Global Journal of Computer Science and Technology: E
Network, Web & Security
Volume 15 Issue 5 Version 1.0 Year 2015
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Extracting Android Applications Data for Anomaly-based Malware Detection

By Joshua Abah, Waziri O.V., Abdullahi M.B., Ume U.A. & Adewale O.S.
University of Technology Minna, Nigeria

Abstract- In order to apply any machine learning algorithm or classifier, it is fundamentally important to first and foremost collect relevant features. This is most important in the field of dynamic analysis approach to anomaly malware detection systems. In this approach, the behaviour patterns of applications while in execution are analysed. The behaviour features that Android as a system allows access permissions to depend on the type of device; either rooted or not. Android is based on the Linux kernel at the bottom layer, all layers on top of the kernel run without privileged mode. Thus, if a behaviour feature vector is created from features of Android (Application Programming Interface) API in unrooted mode, then only system information made available by Android can be used. In this paper, a Device Monitoring system for an unrooted device is developed and used to collect Android application data. The application data is used to build feature vectors that describes the Android application behaviour for Anomaly malware detection.

Keywords: android, anomaly detection, application behaviours, feature vectors, malware detection, mobile device, rooted, unrooted.

GJCST-E Classification : C.1.3 D.4.6

EXTRACTINGANDROIDAPPLICATIONSDATAFORANOMALYBASEDMALWAREDETECTION

Strictly as per the compliance and regulations of:

Extracting Android Applications Data for Anomaly-based Malware Detection

Joshua Abah [α], Waziri O.V. [σ], Abdullahi M.B. [ρ], Ume U.A. [ω] & Adewale O.S. [¥]

Abstract- In order to apply any machine learning algorithm or classifier, it is fundamentally important to first and foremost collect relevant features. This is most important in the field of dynamic analysis approach to anomaly malware detection systems. In this approach, the behaviour patterns of applications while in execution are analysed. The behaviour features that Android as a system allows access permissions to depend on the type of device; either rooted or not. Android is based on the Linux kernel at the bottom layer, all layers on top of the kernel run without privileged mode. Thus, if a behaviour feature vector is created from features of Android (Application Programming Interface) API in unrooted mode, then only system information made available by Android can be used. In this paper, a Device Monitoring system for an unrooted device is developed and used to collect Android application data. The application data is used to build feature vectors that describes the Android application behaviour for Anomaly malware detection. This application is able to collect essential information from Android application such as installed applications and services running within the device before or after the Monitoring application was started, the date/time stamp, calls initiated from the device, calls received by the device, sent short message services (SMSs), SMSs received, and the status of the device as at when the event took place. This information is logged in a comma separated value (.csv) file format and stored on the SDcard of the device. The .csv file is converted to attribute relation file format (.arff); the format acceptable by WEKA machine learning tool. This. arff file of feature vectors is then used as input to the Classifier in the Android malware detection system.

Keywords: android, anomaly detection, application behaviours, feature vectors, malware detection, mobile device, rooted, unrooted.

I. Introduction

Android is one of the most used Smartphone's operating System in the World (Srikanth, 2012). Android is open source with huge user community and documentations as a result of these, it allows any programmer to develop and publish Applications to both the Official or Unofficial market. There are over seven hundred thousand Applications published via the Official Android market, the Google Play Store (Zack, 2012). Malware attack is a challenging issue among the Android user community. This is due to its open source and a very huge adoption and market penetration, making it a target for most malware developers. Android is predicted to be the most used mobile Smartphone platform by 2014 (You, Daeyeol, Hyung-Woo, Jae &Jeong,2014) which has become a reality. This ubiquitous gains of Android brings along with it security risks in terms of malware attacks targeted at this platform. It therefore becomes necessary to make the platform safe for users by providing defence mechanism especially against malware.

There are basically three approaches according to (Burquera, Zurutuza&Nadjm-Tehrani,2011);(Aswathy, 2013); (Lovi&Divya, 2014) to mobile malware detection approaches; static, dynamic and manifest file analyses. While Static analysis focused on the use of patterns of strings called signatures to detect malware presence, dynamic analysis approach to malware detection uses the behaviour pattern of Applications while in execution. The third approach involves the analysis of Android Manifest file. This paper presents a model for mining Applications behaviours for detecting malware on the Android platform using dynamic analysis.

The malware detector attempts to help protect the system by detecting malicious behaviour (Aswathy, 2013). The malware detector performs its protection through the manifested malware detection Approaches.Detection methods for attacks on mobile devices (Burquera, Zurutuza&Nadjm-Tehrani2011);(Wei, Mao, Jeng, Lee, Wang& Wu, 2012); (Wu, Mao, Wei, Lee & Wu, 2012);(Ham, Choi, Lee, Lim & Kim, 2012) have been proposed to reduce the damage from the distribution of malicious applications. However, a mechanism that provides more accurate ways of determining normal applications and malicious applications on Android mobile devices must be developed and a procedure for obtaining the features well defined. This paper developed a model for extracting Android application behaviours through events of normal applications and malicious applications, using a customized approach.

The research employs Anomaly-based detection in a host-based manner to monitor activity that occurs on the target host system. This system is capable of monitoring features of the Android system such as calls received, calls initiated, system calls invoked by running applications, Short Messaging

Author α ρ ¥ : Department of Computer Science, Federal University of Technology Minna, Nigeria. e-mails: jehoshua_a@yahoo.com, abah@unimaid.edu.ng, el.bashir02@futminna.edu.ng, adewale@futa.edu.ng
Author σ : Department of Cyber Security Science, Federal University of Technology Minna, Nigeria. e-mail: victor.waziri@futminna.edu.ng, onomzavictor@gmail.com
Author ω : Department of Information and Media Technology, Federal University of Technology Minna, Nigeria. e-mail: drarthurume@gmail.com

Services (SMSs) received, SMSs sent and screen status of the target device. Anomaly-based detection systems use a prior training phase to establish a normality model for the system activity. In this method of detection, the detection system is first trained on the normal behaviour of the application or target system to be monitored. Using this normality model of behaviour, it becomes possible to detect anomalous activities by looking for abnormal behaviour or activities that deviate from the defined normal behaviour occurring in the system. Though this technique look more complex, it has the advantage of being able to detect new and unknown malware attacks. Anomaly-based detection requires the use of feature vectors to train the classifier before subsequent classification can be carried out. These feature vectors are obtained from features or data collected from the system.

The objective of this work is to extract Android applications data from an unrooted android device and using them to effectively describe the system behaviour. The structure of this paper is given as follows: section one provides a brief introduction; section two gives related literatures; section three discuss the Experimental procedures and setup; section four provides the discussion of result; section five provide the hardware and software used for the experimentation and finally, section six gives the summary and conclusion of the work.

II. Related Works

Android malware detection systems available currently employs static approach to malware detection by scanning files for byte sequences of known malware Applications. Anomaly-based detection is still in a developmental stage and researches are ongoing. As a result, the current approaches are not able to detect unknown attacks. Unknown malware attacks also referred to as 'zero day attacks' are attacks carried out by unknown malware whose signatures have not been analysed and obtained. Several approaches with different metrics for defining Android application behaviours have been developed and are discussed.

You Jounget *al*. (2014);You Joung&Hyung-Woo, (2014)presented an approach for determining malicious attack on Android using System Call Event Pattern Analysis. In their work, system calls invoked by executing Applications of different categories and their frequency of occurrences is used as the metrics for defining Applications behaviour. Their analysis was carried out on Linux system rather than on mobile device. Abelaet *al.*(2013) developed AMDA an automated malware detection system for the Android platform. The core modules of the system included the Feature Extraction Module and the Behaviour Analysis Module. The Feature Extraction Module generates activity log from running applications retrieved from the application repository of the system. The activity log contains the system calls from application activity which are the features that the module retrieves.

Mohammed *et al.* (2014) in the Automatic Feature Extraction part of their work proposed and implemented an approach to detect malicious applications statically through a set of well-defined APIs. Similarly, Tchakounté, &Dayang (2013) used a static approach to analyse System calls of malware on the Android platform.Lin *et al*, (2013)proposed SCSdroid, which uses the thread-grained system calls sequences, because these sequences can be regarded as the actual behaviour of the application. Their approach is a step further from just system calls of Applications to carter for malware repackaged applications. Luoxu & Qinghua, (2013) presented a static approach to their Runtime-based Behaviour Dynamic Analysis System for Android Malware Detection. They used Loadable Kernel Module hooking to hook the Android system and then collect *data.* The collected data consist of IMSI, SIM, IMEI, TEL, call log, SMS, MAIL and so on. The technology of analysis is semantic analysis and regular expression.

Yousra, Wenliang&Heng,(2013) used APIs as the feature for describing Android behaviours used for detecting malware. To select the best features that distinguish between malware from benign applications, API level information within the bytecode were used since it conveys substantial semantics about the apps behaviour. More specifically, they focused on critical API calls, their package level information, as well as their parameters. Dini, Martinelli, Saracino&Sgandurra, (2012)employed two-layer applications behaviour features in order to properly described Android malware behaviours. These include System calls from the kernel layer and other features from the Applications layer. This approach tend to provide a better description of the system than a monolithic view of just a single layer as it considered both the Operating System layer behaviours and the Applications layer behaviours.

It is observed from all the reviewed literatures that System calls pattern analysis played a critical role in providing Android Applications behaviour pattern. It is therefore clear that System calls as features could best be used either singly or in addition to other features to describe Application behaviours not just in Android but any mobile platform.

III. Experimental Procedures and Setup

In this section, the various activities carried out and the different modules implemented to ensure application feature behaviours are intercepted for use in malware detection process are discussed. But before then we show the big picture of the entire malware detection system in a schematic form as in Figure 1.0.

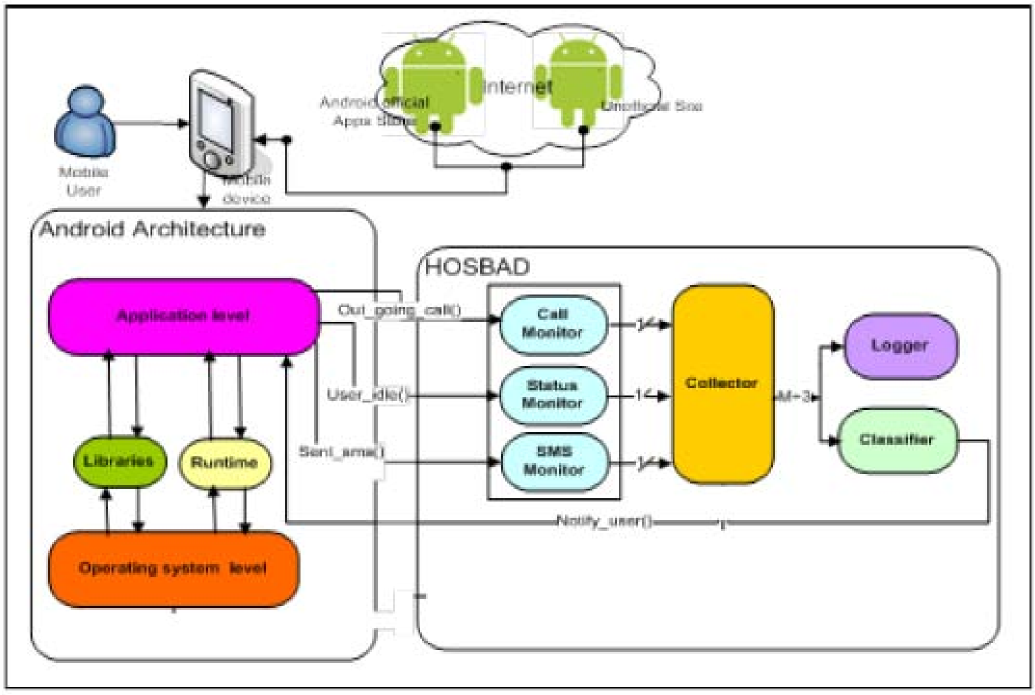

Figure 1. 0 : Architecture of the Android Malware Detection System (HOSBAD)

a) *Application Acquisition Process*

The Application Acquisition process involves downloading applications from Android Markets and storing them into the application repository folder. Applications which could be normal or malicious are downloaded both from the Official Android market and unofficial Android markets. Figure 1.1 shows the Application acquisition processes.

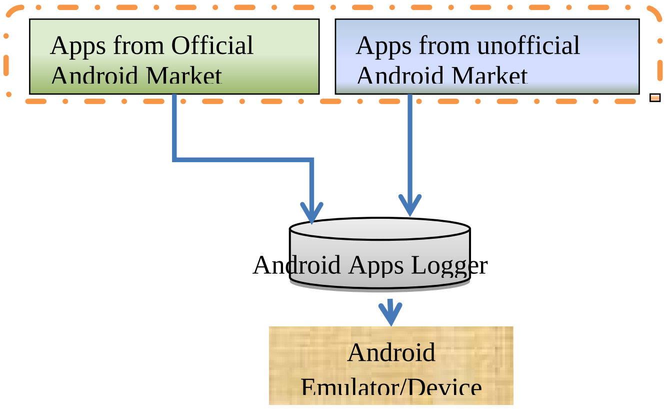
Apps from Official Android Market
Apps from unofficial Android Market
Android Apps Logger
Android Emulator/Device

Figure 1. 1: Schematic of Application Acquisition Process

Each of these Applications is executed in an instrumented Android emulator via Android Virtual Device (AVD). An Android 2.3.3 software development kit (SDK) emulator is used to run the Android applications because this is the only medium to automate the generation of application system activity logs without using an actual mobile device. There is no much actual difference to using human input to be able to activate the behavioural activity of an application. However, the log data contains activities which are irrelevant for detection of malicious activity. With this problem of noise in the log data, the system utilizes a self-developed parser which is customized as to which features are to be collected.

b) *The Data Collection Processes*

In order to collect the Android Applications data, the various monitors described are implemented as Android java programs in the Device Monitoring Application. This application is actually just a module in the complete detection system called HOSBAD. The application will serve as the feature mining model which will run on the Android device to collect the features while the user interacts with Applications on the device. The feature mining model will monitor Android application activities implemented using a broadcast receiver and record on going activity taking place on the device. Figure 1.2 shows the data collection stages by the feature mining model.

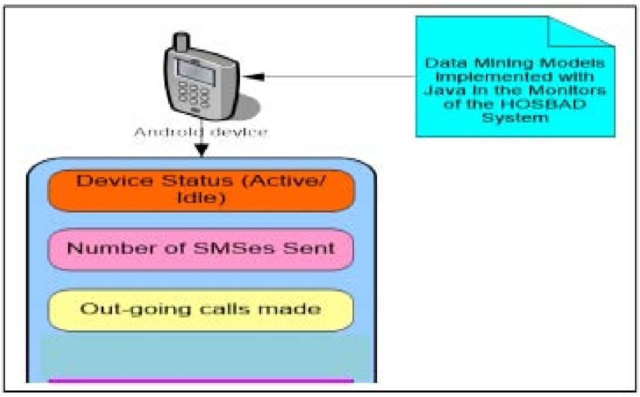
Data Mining Models Implemented with Java in the Monitors of the HOSBAD System
Android device
Device Status (Active/ Idle)
Number of SMSes Sent
Out-going calls made

Figure 1. 2 : The Data Collection Process

The *collector* module in conjunction with the monitors will help to collect as much information as possible from the Android Applications installed on the device. This information include the Date/Time stamp, the application and services running on the device, out-going calls, incoming calls, out-going SMS, incoming SMS, and Device screen status. This information is collectively referred to as feature of application or behaviours . For each .apk file, the device user interaction is created or the emulator simulates user interaction by randomly interacting with the application interface. It should be note that due to the numerous Android Applications available in the Android market, it is not possible for one to monitor and record all Applications for the numerous available Android Applications, doing this will require the researcher to spend many years collecting all of the information about Applications available in the Android market. For this reason, few of the Applications were selected.

c) Android Feature Collection

In order to apply any machine learning algorithm or classifier, it is fundamentally important to first and foremost collect relevant features. The features that Android as a system allows access permissions to depend on the type of device. The type of device here implies whether the device has been rooted or not. Android is based on the Linux kernel at the bottom layer, all layers on top of the kernel layer run without privileged mode. That is, all applications and system libraries are inside a virtual application sandbox. As a result of this architecture, applications are prohibited from accessing other application data (unless explicitly granted permission by other applications called the rooting applications). Thus, if a feature vector is created from features of Android API in unrooted mode, then only system information made available by Android can be used. On the other hand, having a rooted device allows one to install system tools that could gather features from underlying host and network behaviour but doing this subject the device to serious security vulnerabilities as the entire device file system will be opened up to attacks.

In this Work, an unrooted device is used in order to collect Android application data. To be able to do this, a feature mining model which is a self-developed application module that will be part of the detection system is used. This application is able to collect essential information from Android application such as installed applications and services running within the device before or after the Monitoring application was started, the date/time stamp, calls initiated from the device (outCalls), calls received by the device (InCalls), sent SMSs (OutSMS), SMSs received (InSMS), and the status of the device (Screen) as at when the event took place. This information is written into a log file and stored on the SDcard of the device. This log file is a comma separated value in .csv format. Parsing these data with another self-developed code module will produce the feature vectors which is in .arff file format; the format acceptable by WEKA. This self-developed code module that serves as a feature mining model for application enable us to create a folder were all monitored/recorded application logs in csv file format will be stored. This csv file will be parsed by another parserto make feature vector file in arff. This arff file of feature vectors will be used as input to the Classifier in the Android malware detection system.

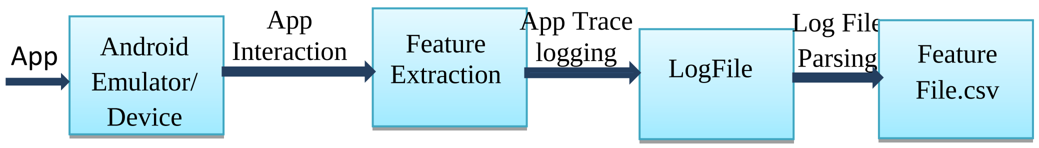

Figure 1. 3 : Features Extraction Processes

The data extraction application performs the following major task as it runs either in foreground or background. This is represented in Figure 1.3: the features extraction processes.

i. First, the Android application runs either on the emulator or real device, the Device Monitoring which implements the feature extraction model; a self-developed module that implements the monitors runs in the background to intercepts and records the specified features (out call, in call, out SMS, in SMS, and device status).
ii. Secondly, the log stream is input to the parser in the Device Monitoring application and is parsed by filtering and formatting the log data to a readable form in a comma separated value (csv) format.
iii. Finally, the csv file will then be parsed by another parser to generate a .arff file that will be used by the classifier.

i. *Implementation Details*

Although the code for the Device Monitoring application which is the data extraction model cannot be given here, the skeletal description of the different modules representing the respective monitors is presented. The broadcast receiver class for the calls and receiving incoming SMS record the calls and SMS events into app preferences, there is no proper receiver for the outgoing SMS so special observer class is used in the service class. When this receiver is started in service, it doesn't work on real device, so it is registered in the manifest and the preferences is used. The structure of the public class; ReceiverCallSms that implements the calls and the SMS is given as;

```
public class ReceiverCallSms extends
BroadcastReceiver {
```

Within this class, the methods for the calls (out-going and in-coming calls) and the in-coming SMS are implemented in a single method with a nexted*if ..else* statement.

The Inner broadcast receiver for monitoring the screen condition is implemented with the class ScreenReceiver which implements the onReceive method using special observer "intent".

The service monitoring is implemented by a class Service Monitoring with a method that records the services running on the device and the features to be extracted. The Binder function initiates the monitoring process when the start button is clicked and to stop the monitoring when the stop button is clicked. All monitored events and activities are written to a file in a comma separated value format. The method checks for the presence of an SD card and create a folder there where the file will be stored or setup a Gmail account where the file will be sent to without user interference. The file is named using the device date/time stamp.csv.

```
07.10.2015 20:33:55,Monitoring Started
Time,AppName,OutCall,InCall,OutSMS,InSMS,Screen,Class
 before,YouTube,0,0,0,0,1,?
 before,Launcher,0,0,0,0,1,?
 before,Torch,0,0,0,0,1,?
 before,Opera Mini beta,0,0,0,0,1,?
 before,Contacts,0,0,0,0,1,?
 before,Phone,0,0,0,0,1,?
```

Figure 1.4 shows a screenshot of the feature mining model application for the malware detection system.

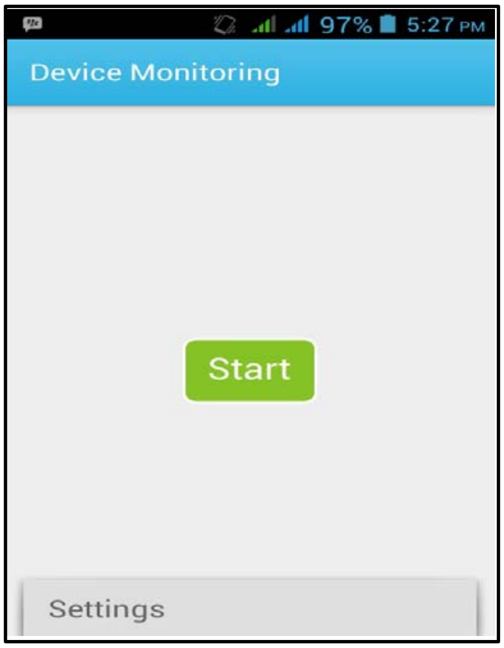

Figure 1.4 : Feature Mining Model Application

The settings menu provides the avenue for creating folder where reports will be stored on the SD card and to also specify a Gmail account and mail subject if the report is to be sent to a remote recipient or possibly server for analysis.

d) Feature Vectors

Analysing activities of the system will give an accurate representation of the behaviour of the applications. The aim of intercepting these activities is to create an output file containing the events generated by the Android applications. This file provides useful information such as opened and accessed applications, running applications, running services, timestamps, received SMSs, sent SMSs, calls received, calls initiated and device status as at the time of occurrence of the activity. This information generated by the Device Monitoring application is used to represent the behaviour of applications.

IV. Discussion of Result

A sample report obtained from a single run of the feature extraction model implemented as a Device Monitoring application is given and discussed here.

```
before,Facebook,0,0,0,0,1,?
before,Messages,0,0,0,0,1,?
before,com.mediatek.voicecommand.service.VoiceCommandManagerService,0,0,0,0,1,?
before,com.mobogenie.service.WifiUpdateService,0,0,0,0,1,?
before,ua.com.doublekey.devicemonitoring.ServiceMonitoring,0,0,0,0,1,?
before,com.mobogenie.service.CommonService,0,0,0,0,1,?
before,com.mediatek.CellConnService.PhoneStatesMgrService,0,0,0,0,1,?
before,com.tecno.ime.IME,0,0,0,0,1,?
before,com.mediatek.filemanager.service.FileManagerService,0,0,0,0,1,?
before,com.mobogenie.service.MobogeniePushService,0,0,0,0,1,?
before,com.afmobi.palmchat.LaunchService,0,0,0,0,1,?
before,com.whatsapp.messaging.MessageService,0,0,0,0,1,?
before,com.mediatek.FMRadio.FMRadioService,0,0,0,0,1,?
before,com.facebook.push.mqtt.service.MqttPushService,0,0,0,0,1,?
before,com.mobogenie.service.MobogenieService,0,0,0,0,1,?
before,com.mobogenie.plugin.cys.cleaner.service.BackgroudCheckService,0,0,0,0,1,?
07.10.2015 20:36:47,com.facebook.fbservice.service.DefaultBlueService,0,0,0,0,1,?
07.10.2015 20:36:47,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:36:50,com.facebook.vault.service.VaultManagerService,0,0,0,0,1,?
07.10.2015 20:36:52,com.facebook.analytics.service.AnalyticsService,0,0,0,0,1,?
07.10.2015 20:37:50,com.facebook.fbservice.service.DefaultBlueService,0,0,0,0,1,?
07.10.2015 20:41:07,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:41:15,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:42:24,Launcher,0,0,0,0,1,?
07.10.2015 20:42:36,YouTube,0,0,0,0,1,?
07.10.2015 20:45:35,Launcher,0,0,0,0,1,?
07.10.2015 20:45:38,Google Play Store,0,0,0,0,1,?
07.10.2015 20:45:44,Launcher,0,0,0,0,1,?
07.10.2015 20:45:48,Gallery,0,0,0,0,1,?
07.10.2015 20:47:55,Launcher,0,0,0,0,1,?
07.10.2015 20:48:02,Torch,0,0,0,0,1,?
07.10.2015 20:48:05,Launcher,0,0,0,0,1,?
07.10.2015 20:48:18,Contacts,0,0,0,0,1,?
07.10.2015 20:48:32,?,1,0,0,0,1,?
07.10.2015 20:48:32,Phone,0,0,0,0,1,?
07.10.2015 20:49:27,Contacts,0,0,0,0,1,?
07.10.2015 20:49:29,Launcher,0,0,0,0,1,?
07.10.2015 20:49:32,Email,0,0,0,0,1,?
07.10.2015 20:51:12,Launcher,0,0,0,0,1,?
07.10.2015 20:51:20,SendSMS,0,0,0,0,1,?
07.10.2015 20:51:20,?,0,0,0,1,0,?
07.10.2015 20:51:32,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:51:40,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:51:57,?,0,1,0,0,1,?
07.10.2015 20:51:57,Phone,0,0,0,0,1,?
07.10.2015 20:52:01,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 20:58:48,Launcher,0,0,0,0,1,?
07.10.2015 20:58:49,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 21:07:04,WhatsApp,0,0,0,0,1,?
07.10.2015 21:08:52,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 21:09:40,com.facebook.conditionalworker.ConditionalWorkerService,0,0,0,0,1,?
07.10.2015 21:13:08,Tecno Input,0,0,0,0,1,?
07.10.2015 21:13:10,WhatsApp,0,0,0,0,1,?
07.10.2015 21:16:12,Launcher,0,0,0,0,1,?
07.10.2015 21:16:28,SendSMS,0,0,0,0,1,?
07.10.2015 21:16:53,Launcher,0,0,0,0,1,?
07.10.2015 21:16:59,Device Monitoring,0,0,0,0,1,?
```

```
07.10.2015 21:17:00,Monitoring Stopped
Tally:,,out calls: 1,in calls: 1,out sms: 0,in sms: 1
```

The report shows the date and time the Monitoring Device application was started. Immediately after that line is the field or attributes of the collected information in a CSV manner. After the attributes are the attribute values entered in the order of the specified attributes. The first attribute is the Date/Time, followed by AppName, OutCall, InCall, OutSMS, InSMS, Screen, and finally the Class in that order. For applications and services running before the Monitoring Device application was started, the Date/Time stamp is indicated as "before" while the applications and services started after the Monitoring Device application was started, the date/time stamp is indicated.

It is indeed very difficult to know which application performs a given activity since certain tasks are deprecated at application layer. Therefore, any activity that occurred without knowing which application perform the activity is given '?' as the value for the AppName attribute at that point. For the OutCall, InCall, OutSMS, InSMS and Screen attribute, the attributes have Boolean values; the value 0 is entered to represent the absence of the attribute and 1 is entered to represent the presence of that attribute. For the Screen attribute that represents the device status which is either idle or active, the value 1 means that the screen is in 'ON' or active state while 0 imply 'OFF' or idle state. Finally, the last attribute Class is not actually extracted from the applications or services by the Device Monitoring application but appended to the log file to indicate the class after classification is done using the classifier. Since the classification has not yet beencarried out on the data, the classes of the instances are undetermined and so they all have the value of '?' that means unknown class (normal or malicious).

When the Device Monitoring application is stopped, the event together with the Date/Time stamp of the event is registered and finally the report gives a summary of all the events in the form of count or tally.

V. Hardware and Software

The experiments were run on a laptop machine with the Intel Core-i3 -370M Processor, 3GBof available memory and 500GB Hard Disk Drive (HDD). This machine runs Windows 7 Operating System while Android Studio 1.2.2 Integrated Development Environment (IDE) was used as the Software Development Kit (SDK).

VI. Summary and Conclusion

In this paper, we describe the development of a feature extraction model that is used to extract Android application behaviour for anomaly malware detection. The type of information that can be extracted depends on whether the device has been rooted or not. Our focus is on unrooted Android devices and the information that were extracted and used to describe Android application behaviours include date/time stamp of the running application and services given as Time, Application and service name (AppName), Outbound call (OutCall), Inbound call (InCall), Outbound SMS (OutSMS), Inbound SMS (InSMS) and the device status (Screen). The device status indicates whether there is an active interaction with the device by the user or not. When the screen is active (value of 1), it means there is active interaction with the device by the user and when the screen is idle or hibernated, it implies no active user interaction. Activities like sending SMS and initiating calls requires active user interaction. If these attributes have values of 1 when the screen state is idle (value of 0) implies a suspicious or malicious behaviour is taking place on the device by an application.

Although other features could be added, these were used as a test base to realise the concept of anomaly detection system. As earlier stated, the type of information that can be intercepted depends on whether the device is rooted or not. Rooting a device is a bridge of security and therefore opens up the device to attacks. Since the aim is to improve security of mobile devices and applications with Android platform, an unrooted device is used. To be able to access more information that could be used to describe application behaviour for anomaly detection purposes, it is recommended that access to certain information like system calls, network traffic etc. which are presently deprecated in unrooted Android systems should be allowed access by Google in some ways.

References Références Referencias

1. Abela Kevin, Joshua AngelesL., Don Kristopher E., Delas Alas, Jan Raynier P., Tolentino, Robert Joseph and Gomez, Miguel Alberto N. (2013). An Automated Malware Detection System for Android using Behavior-based Analysis AMDA. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)* 2 (2), pp 1-11 *The Society of Digital Information and Wireless Communications.*
2. You Joung Ham and Hyung-Woo Lee (2014). Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events. *International Journal of Computer and Communication Engineering,* 3 (2), pp 149 -154, March 2014.
3. Ham Y.J., Choi W.B., Lee H.W., Lim J.D. and Kim J.N. (2012), Vulnerability monitoring mechanism in Android based smartphone with correlation analysis on event-driven activities" *2012 2nd International Conference on Computer Science and Network Technology*, pp. 371-375.
4. Wu D.J., Mao C.H., Wei T.E., Lee H.M. and Wu K.P. (2012), Droid Mat: Android Malware Detection

through Manifest and API Calls Tracing, 7th *Asia Joint Conference on Information Security*.

5. Wei T.E., Mao C.H., Jeng A.B., Lee H.M., Wang H.T. and Wu D.J. (2012), Android Malware Detection via a Latent Network Behaviour Analysis, *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*.
6. Burquera I., Zurutuza U. and Nadjm-Tehrani S. (2011). Crowdroid: behavior-based malware detection system for Android, Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15-26.
7. You Joung Ham, Daeyeol Moon, Hyung-Woo Lee, Jae Deok Lim and Jeong Nyeo Kim (2014). Android Mobile Application System Call Event Pattern Analysis for Determination of Malicious Attack. *International Journal of Security and Its Applications* 8(1), pp.231-246, http://dx.doi.org/10. 14257/ijsia. 2014.8.1.22
8. Zack, Islam (2012). *Google Play Matches Apple's iOS With 700,000 Apps.* Businessweek, 30 October 2012. Retrieved fromhttp://www.tomsguide.com/us/Google-Play-Android-Apple-iOS,news-16235.html
9. Dini, G., Martinelli, F., Saracino, A. and Sgandurra, A. (2012). MADAM: A Multi-level Anomaly Detector for Android Malware. *Computer Network Security, Lecture Notes in Computer Science*, 7531, 240-253.
10. YousraAafer, Wenliang Du, and Heng Yin, (2013). Droid APIMiner: Mining API-Level Features for Robust Malware Detection in Android. pp 1-18. Retrieved from http://www.google.com
11. Luoxu Min and Qinghua Cao, (2013).Runtime-based Behaviour Dynamic Analysis System for Android Malware Detection. pp. 1-4. Retrieved from http://www.google.com
12. Ying-Dar Lin, Yuan-Cheng Lai, Chien-Hung Chen, and Hao-Chuan Tsai (2013). Identifying Android Malicious Repackaged Applications by Thread-grained System call Sequences, *Elsevier:Computers & Security*, pp 1-11, (2013), http://dx.doi.org/10.1016/j.cose.2013.08.010
13. Lovi Dua and Divya Bansal (2014).Taxonomy: Mobile MalwareThreats and Detection Techniques. *Dhinaharan Nagamalai et al. (Eds) : ACITY, WiMoN, CSIA, AIAA, DPPR, NECO, In WeS*–2014 pp. 213–221.
14. Aswathy Dinesh (2013). An Analysis of Mobile Malware and Detection Techniques. pp 1-13. Retrieved from http://www.google.com
15. Tchakounté F. and Dayang P. (2013). System Calls Analysis of Malwares on Android. *International Journal of Science and Technology* 2(9), pp 669-674 September, 2013.
16. Muhammad ZuhairQadir, AtifNisar Jilani, and Hassam Ullah Sheikh (2014). Automatic Feature Extraction, Categorization and Detection of Malicious Code in Android Applications. *International Journal of Information & Network Security (IJINS)*3(1), pp. 12~17, February 2014.
17. Srikanth, R. (2012). Mobile Malware Evolution, Detection and Defense. *Unpublished Term Survey Paper*, *Institute for Computing, Information and Cognitive Systems*, University of British Columbia, Vancouver, Canada.