INTERNATIONAL JOURNAL OF ADVANCED RESEARCH IN ENGINEERING AND TECHNOLOGY (IJARET)

ISSN 0976 - 6480 (Print)
ISSN 0976 - 6499 (Online)
Volume 5, Issue 4, April (2014), pp. 213-221
© IAEME: www.iaeme.com/ijaret.asp
Journal Impact Factor (2014): 7.8273 (Calculated by GISI)
www.jifactor.com

IJARET
©IAEME

A NOVEL METHODOLOGY FOR ANALYZING MALICIOUS BEHAVIOR OF THE ANDROID APPLICATIONS

Namrata A. Patel[1], **Vijay Ukani**[2], **Nilay Mistry**[3]

[1, 2]Department of Computer Science & Engg, Institute of Technology, Nirma University, Ahmedabad, India,
[3]Asst. Professor, Department of Digital Forensic Division, Gujarat Forensic Sciences University, Gandhinagar, India

ABSTRACT

Smart phones have evolved from simple mobile phones into sophisticated yet compact minicomputers. Mobile gimmicks become vulnerable to various threats such as viruses, worms, trojan horses and malwares. Until some sealed security measures like application signing and validation of developers was introduced, the number of infected applications steadily increased. Smart phones run on Android OS are one that continues to be a prime target for hackers. Our focus is mainly on Android-powered mobile device applications. In these days for smart phones, Android has become a very popular operating system. As we all aware, Android is a Linux-based operating system. With this operating system user can easily share applications via online market store i.e. Google market store. But, this platform includes different types of malicious applications. Because malware on device can create numerous risks, which create problems while connectivity because of security consequences. In this paper, it will be described that using methodology how we can analyze malicious behavior of the Android applications so that users can safely use the android smart phones.

Keywords: Android Mobile Malware, Static Analysis, Dynamic Analysis, Reverse Engineering, Smartphone Security.

I. INTRODUCTION

The speedy gain in smart phone users has enabled the application marketplace to develop extensively. The black market presence has also risen quickly, where paid applications are altered for free transfer. As an outcome, malicious applications are anticipated to disseminate with raising frequency [1].

An Android is an open source operating system. Android smart phones offer advanced computing ability and connectivity as compared to other mobile operating systems. Android was initially originated by Android Inc. and was afterward purchased by Google. Android is a Linux based operating system. It was made for touch screen devices like smart phones, tablets, cameras, set-top boxes, etc. and has attained the hands of millions of users. The architecture of android operating system is designed in such manner so that communication at application level and end user will be quite promiscuous. Android applications are coded in Java, a programming language [7]. But Android has its own virtual machine i.e. DVM (Dalvik Virtual Machine) and to execute the android applications, DVM is used.

Android operating system is also used for business purposes. Risk will be high in business level because people are transmitting sensitive information from one end to another end. As Android allows remote access to official sensitive data e.g. personal information, which can lead to data hack if smart phones are hacked into. For this intension of protection, Google has designed their operating system to execute applications in specified sandboxes, which understate the potentiality of malware attacks to the applications in smart phones [2].

Due to convenient features inherent in smart phones, the number of smart phone users is rapidly increasing. People use smart phones for various purposes such as web brows- ing, calling, SMS, social networking, omnipresent access, and so on. Any technology that is in a lot of hands is a target for malicious applications. Why not? Both official smart phone application markets such as the Apples AppStore and Googles Android Market and nonofficial third party application markets called Black markets have been growing at an eminent rate. However, as the Android Market and Blackmarkets accept applications without any code review or security checks, hackers can easily disseminate their malicious applications into the Android Market. That is, hackers can manipulate any popular application from the market and repackage it to include malicious code. Trojan-SMS. AndroidOS first appeared in August, 2010, Geinimi in December, 2010 and Android/Spyware.PJApps.a in March, 2011, Gingerbreak in August, 2011. This represents the major types of Android malware [6].

The below chart illustrates worldwide smart phone operating system market share from Q1 2010 through Q1 2013[3]. All figures are based on unit sales to end users.

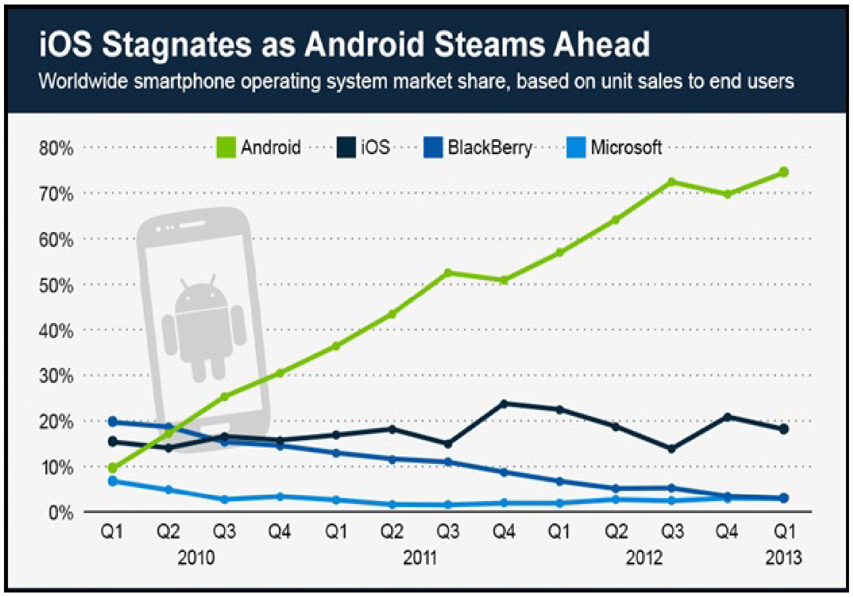


Fig. 1: Android market [3]

II. OBJECTIVE AND DEFINITIONS

The objective of this research paper is Suspicious Activity Assessment. The below figure shows the major parts of malware analysis.

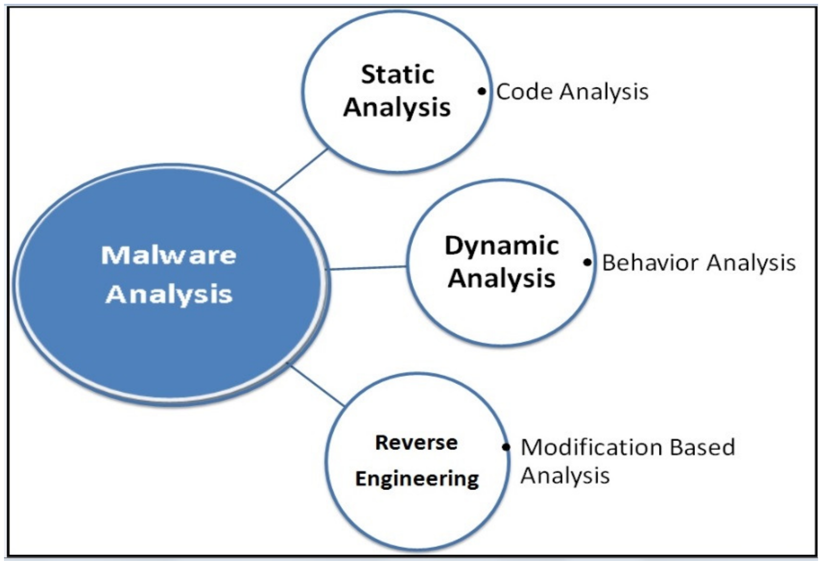


Fig. 2: Malware Analysis

Static analysis involves human work to trace and examine malware, is more common because of the comparatively small number of mobile malware. Static analysis can provide higher quality results and lead to a better understanding [4].

On the other hand, Dynamic analysis involves automated tools to execute the malware in a controlled system environment and check for malicious forms. Therefore a large sample of malware can studied rapidly. Dynamic analysis is not yet common for mobile devices because of the trouble of replicating the numerous mobile hardware, mobile operating systems, and their many different versions [4]. Mobile malware also have different ways of spreading as opposed to desktop malware, which further makes dynamic analysis more difficult. Obfuscation and encryption are quite common for mobile malware, which also demand for extra steps in the analysis. Other sophisticated techniques for evasion used by mobile malware include polymorphism (changing variables and files), and steganography (hiding information in unexpected places). Reverse Engineering is a process of analyzing an existing code or piece of software in order to scrutinize the software for any vulnerability or any errors [5]. "Reverse engineering can therefore be stated as a method or process of modifying a program in order to make it behave in a manner that the reverse engineer desires."

III. DETAILS OF OUR ANALYSIS METHODS

Our intention as a part of this effort is to identify unauthorized actions of applications that are not initiated by users. Such activity may be triggered via background running process or self running script. Our initiation is to characterize programs by statically examining how many such operations (data or system resource access) found in the application. Mobile apps typically have intense user interaction, allowing us to approximate implicit user intention with explicit user inputs and actions. The control flow user interaction procedure is used in this section to describe the analysis. The algorithm performs the following high level steps.

The final conclusion depends on two step analysis as below.

Scenario 1) Check for Behavior and permissions violation of an app

Scenario 1.1) The first analysis is for those applications which are designed to use only in offline mode (like some games) but uses internet besides user knowledge. Hacker may use such behavior to transfer sensitive data. Initially the apk file will be decompiled with the steps described in the static analysis flow. As a result we will have a source code for the application installed. Then with the help of String tokenize algorithm every line of code will scan and check for the specific keyword.

Scenario 1.2) Second attempt is for those applications which runs on internet but connect with more than one service end point (URL). This indicates some malicious activity without user knowledge. Some sensitive data may transfer to hacker via second endpoint.

Below figure displays Flow Diagram of Scenario 1.

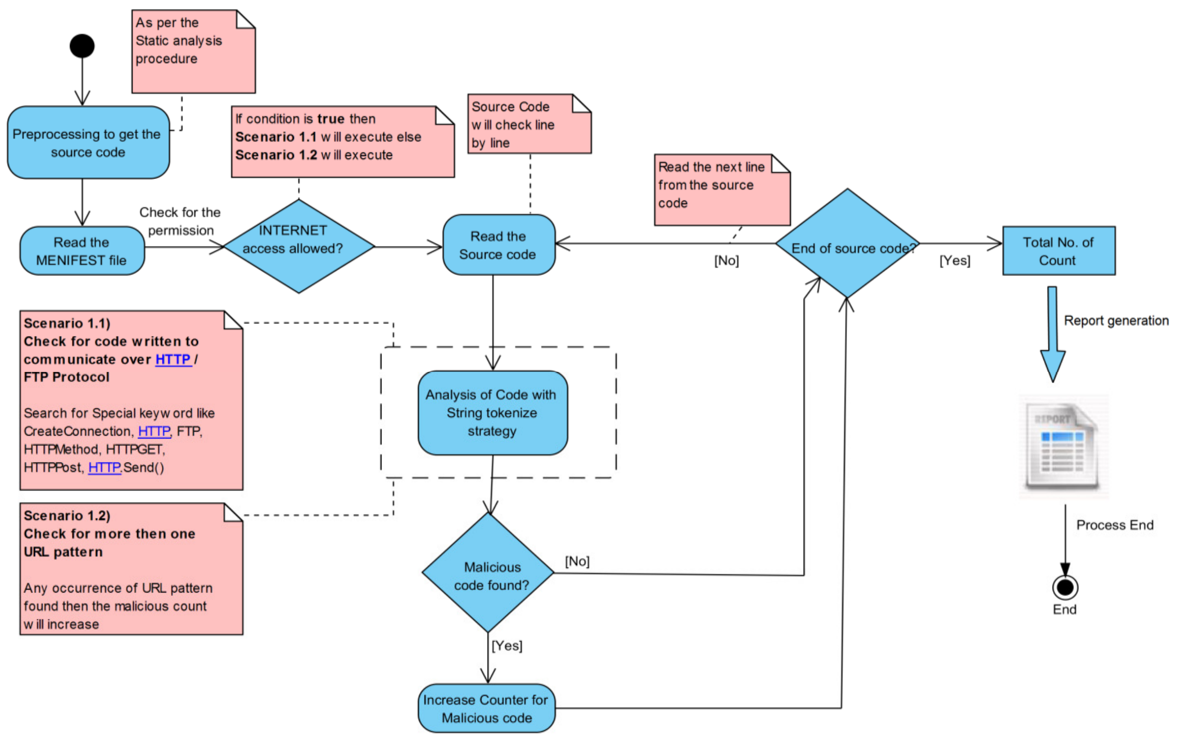

As per the Static analysis procedure
Preprocessing to get the source code
Read the MENIFEST file
Check for the permission
INTERNET access allowed?
If condition is true then Scenario 1.1 will execute else Scenario 1.2 will execute
Source Code will check line by line
Read the Source code
Read the next line from the source code
End of source code?
[No]
[Yes]
Total No. of Count
Report generation
Process End
End
Scenario 1.1) Check for code written to communicate over HTTP / FTP Protocol
Search for Special keyword like CreateConnection, HTTP, FTP, HTTPMethod, HTTPGET, HTTPPost, HTTP.Send()
Scenario 1.2) Check for more then one URL pattern
Any occurrence of URL pattern found then the malicious count will increase
Analysis of Code with String tokenize strategy
Malicious code found?
[No]
[Yes]
Increase Counter for Malicious code

Fig. 3: Flow Diagram of Scenario 1

As a result of the above mentioned processing, the malicious count can be calculated depending on the below equation.

$$\text{Malicious Count}(X\ \%) = \frac{\text{No. of malicious occurrences} * 100}{\text{Total No. of lines}} \qquad (1)$$

Scenario 2) No coupled unused or dead code without User interaction

- Static code analysis which is having some lines of code and which do not have coupling with user action. It means those lines of code which will never going to call by any action.

Below figure shows the Dependence Diagram of Scenario 2.

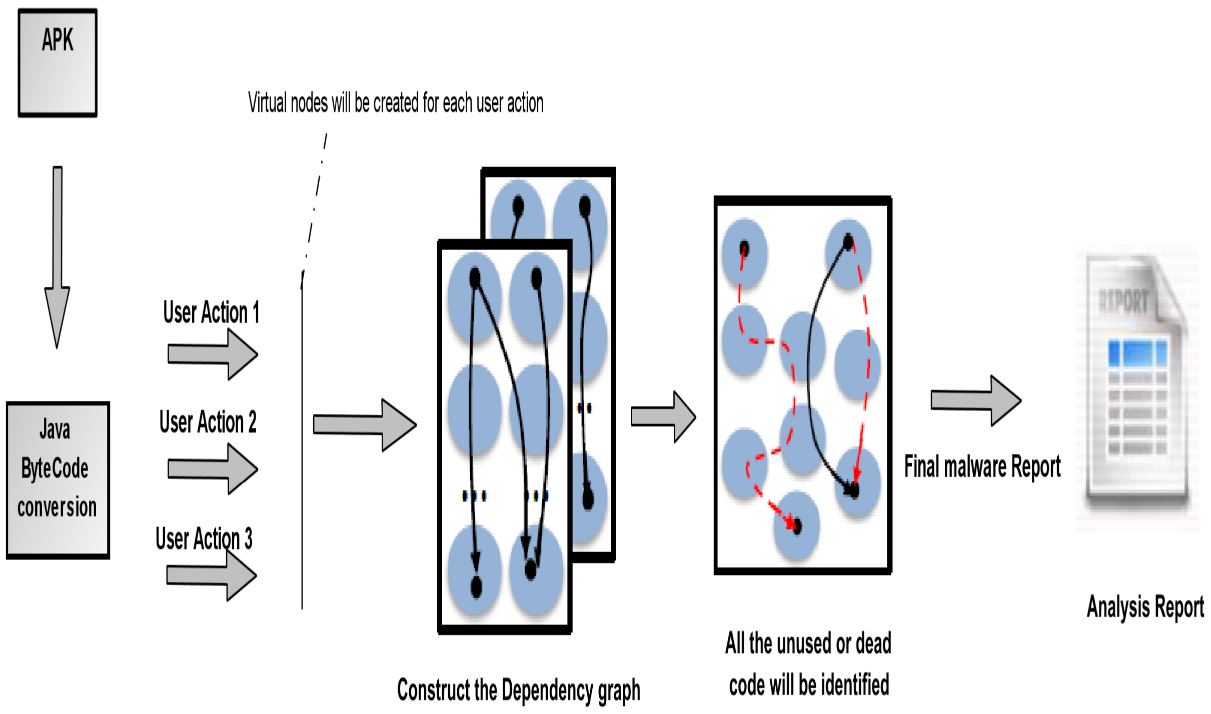

APK
Virtual nodes will be created for each user action
User Action 1
User Action 2
User Action 3
Java ByteCode conversion
Construct the Dependency graph
All the unused or dead code will be identified
Final malware Report
REPORT
Analysis Report

Fig. 4: Dependence Diagram of Scenario 2

- Logically every user action will create method dependent virtual path within the code. These paths are dependent on use-cases of individual application. After every possible virtual path, if we found some classes or methods which is not mapped with any virtual path node then those lines could be used for malicious activities as those were never called by any user action.

Different equation will be used to generate the malicious count from scenario 2. This time we will use the number of dead or unused lines of code which never had been called from any user action.

$$\text{Malicious Count}(Y\ \%) = \frac{\text{No. of unused lines} * 100}{\text{Total No. of lines}} \qquad (2)$$

Finally the average value of the result one and two makes the final output result of any application as below,

$$\text{Final malicious Count } (C\ \%) = \frac{X + Y}{2} \qquad (3)$$

IV. IMPLEMENTATION

Android framework provides simplified utilities to develop quick Android applications. In eclipse version 3.5, Android plug-in(s) have been installed which provides developer friendly environment [10]. All the approaches defined in above sections were implemented in a small application to have a better look on how this approach can work for future industrial use.

The intention behind this project is to generate malware count report for every application installed on any android device. As android executable (.apk) does not give the direct source code, with the help of some technique we have to first generate the source code out of the apk file. As a first step the android framework has to install where the apk file will be process for decompilation. As a result of apk decompilation the artifacts will be generated as below [5].

- Res folder: This folder comprises XMLs specifying the attributes, layout, drawables, languages, etc.
- Android Manifest File: This file is one of the most essential XML file which contains information about the permissions that the application needs or accesses. In other words this file contains the Meta data regarding the application.
- Smali folder: This folder comprises the source code of the application in .smali format.

Then the next step is to convert .dex files into jar files. The Jar file will contain all the third party libraries and source code in .class format. Class files are byte code version of java source which is non human readable. In the next step we will convert that into .java file.
Below figure displays Flow Diagram of the project.

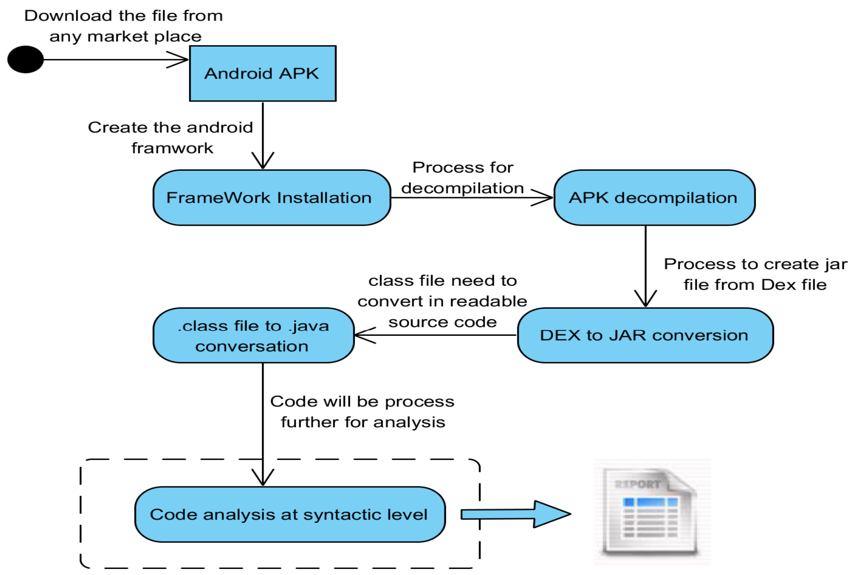

Download the file from any market place
Android APK
Create the android framwork
FrameWork Installation
Process for decompilation
APK decompilation
Process to create jar file from Dex file
DEX to JAR conversion
class file need to convert in readable source code
.class file to .java conversation
Code will be process further for analysis
Code analysis at syntactic level
REPORT

Fig. 5: Flow diagram

Once we have the source code with us, all the approaches which we discussed in section 3 are implemented programmatically to analyze the code for malicious activity. After this process the report will be generated, this indicates the malicious count in percentage for each application installed.

To demonstrate the concept of Malware analysis, a sample application was created. This application will implement the fundamentals of Malware detection and generates malicious count. This application can run on any android version. Below images shows how the application will look like. Figure 6.1 displays the main screen of the application. When user starts processing the application, it will start scanning for malwares. This scenario is shown in Figure 6.2. As a result the output will be shown to the user in well defined format as shown in Figure 7.

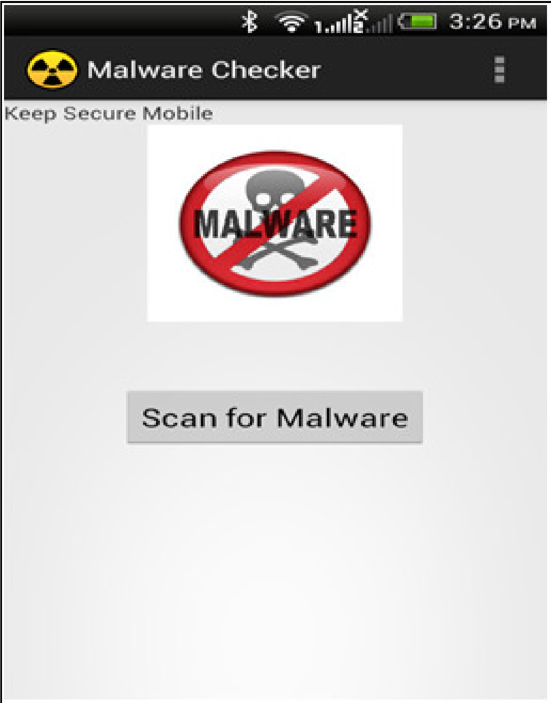


Figure 6.1: Home Screen

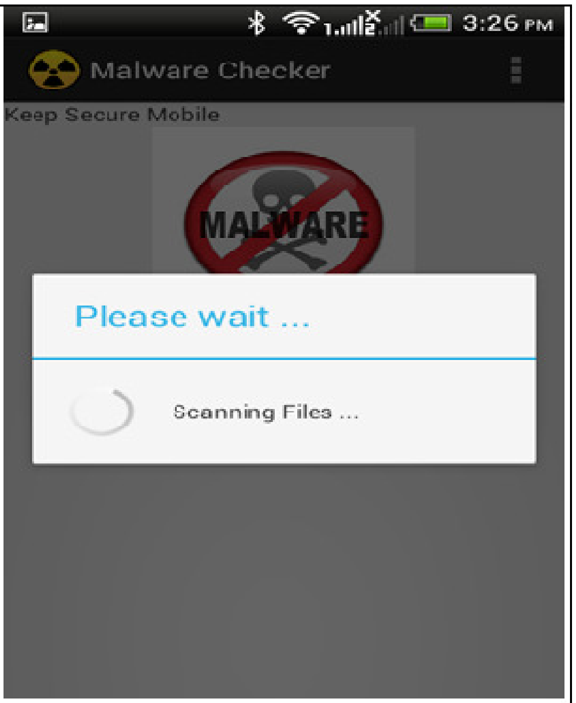
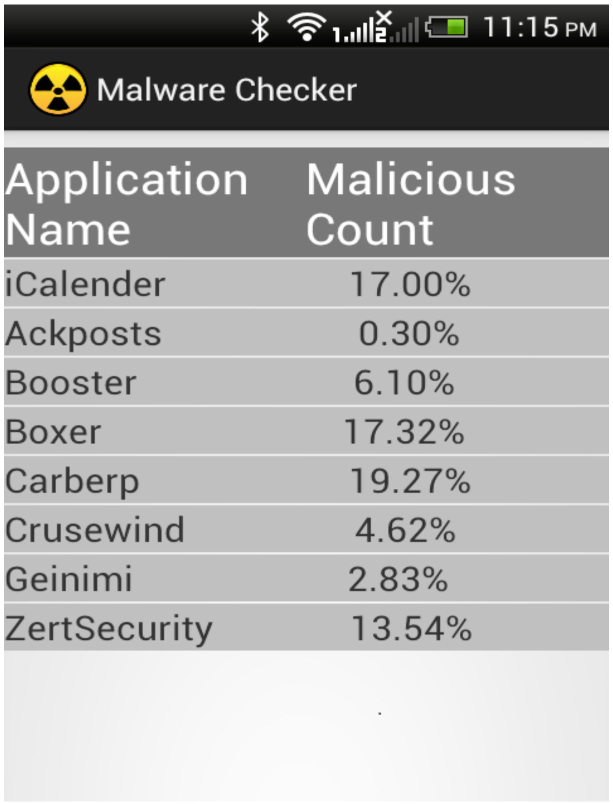


Figure 6.2: Scanning Process



Application Name	Malicious Count
iCalender	17.00%
Ackposts	0.30%
Booster	6.10%
Boxer	17.32%
Carberp	19.27%
Crusewind	4.62%
Geinimi	2.83%
ZertSecurity	13.54%

Fig. 7: Final Output

Below figure displays Comparison of the different applications in percentage.

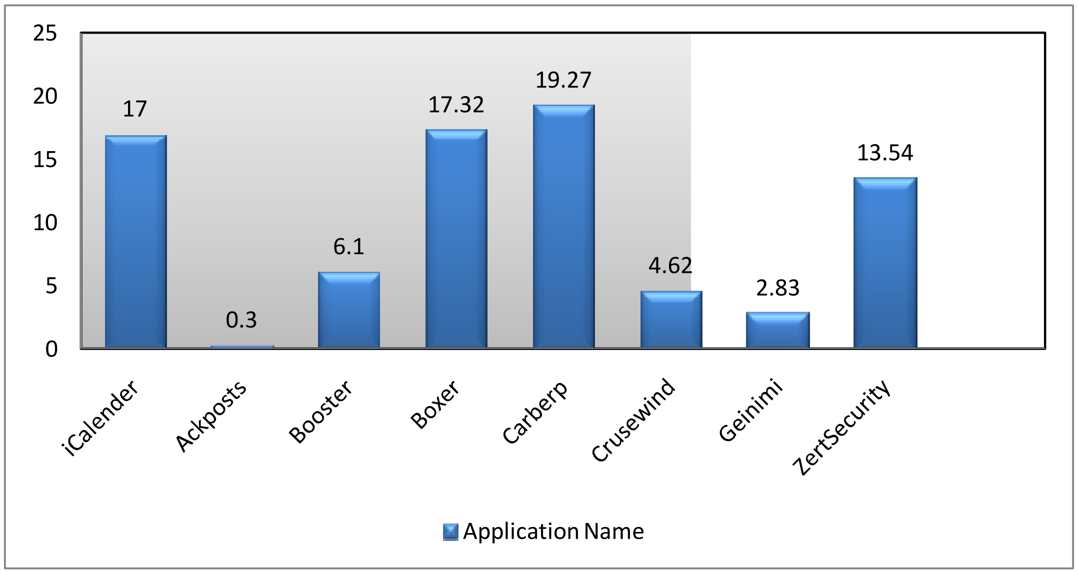


Fig. 8: Graphical Analysis

V. CONCLUSION

Overall, malware analysis is a stimulating and ambitious area of computer security research. The complication which we have gone through in malware analysis is only one area of the security profession that is invariably evolving. The most appropriate way of understanding malwares is by studying the working of an already existing malware. Our initiative with this research is an effort to protect against threats in this huge security issues in mobile device.

VI. ACKNOWLEDGEMENTS

This research was supported by the Institute of Technology, Nirma University, Ahmedabad, India and Gujarat Forensic Sciences University, Gandhinagar, India.

VII. REFERENCES

[1] S.-H. Seo, D.-G. Lee, and K. Yim, “Analysis on maliciousness for mobile applications," in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on, pp. 126-129, IEEE, 2012.

[2] A. G. Kirandeep, “Implementing security on android application," in The International Journal of Engineering and Science (IJES) Volume 2, Issue 3, pp.56-59, IJES, 2013.

[3] “Androidmarket."http://www.statista.com/topics/840/smartphones/chart/1099/smartphone-operating-system-market-share/.

[4] E. Erturk, “A case study in open source software security and privacy: Android adware," in Internet Security (WorldCIS), 2012 World Congress on, pp. 189-191, IEEE, 2012.

[5] Stephen. A.Ridley, "Android Malware Reverse Engineering", Retrieved from: http://dl.dropbox.com/u/2595211/HelloMotoAndroidReversing.

[6] A. Shabtai, “Malware detection on mobile devices," in Mobile Data Management (MDM), 2010 Eleventh International Conference on, pp. 289-290, IEEE, 2010.

[7] A.-D. Schmidt, H.-G. Schmidt, L. Batyuk, J. H. Clausen, S. A. Camtepe, S. Albayrak, and C. Yildizli, "Smartphone malware evolution revisited: Android next target?," in Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on, pp. 1-7, IEEE, 2009.

[8] "Comparison of dalvik and java bytecode." http://forensics.spreitz-enbarth.de/2012/08/27/comparison-of-dalvik-and-java-bytecode/.

[9] T. Eder, M. Rodler, D. Vymazal, and M. Zeilinger, "Ananas a framework for analyzing android applications," in Availability, Reliability and Security (ARES), 2013, Eighth International Conference on, pp. 711-719, IEEE, 2013.

[10] Installing the *Eclipse Plugin*, *Android* Developers *developer.android.com/sdk/installing/installing-adt.html*

[11] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "andromaly: a behavioral malware detection framework for android devices," Journal of Intelligent Information Systems, vol. 38, no. 1, pp. 161-190, 2012.

[12] 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, pp. 15-26, ACM, 2011.

[13] H. Lemoine, "The dyana framework dynamic analysis for android,"

[14] "Reverse engineering definition." http://searchcio-midmarket.techtarget.com/definition/reverse-engineering.

[15] "Android sdk tool." https://developer.android.com/sdk/index.html.

[16] "Jd-gui tool." http://jd.benow.ca/

[17] Alappanavar, P. B., Ms Ankeeta Bhujbal, and Mr Shantanu Deshmukh. "INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY (IJCET)." *Journal Impact Factor* 4.2 (2013): 237-240.

[18] Asokan M, "Android Vs iOS – An Analysis", International Journal of Computer Engineering & Technology (IJCET), Volume 4, Issue 1, 2013, pp. 377 - 382, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.

[19] Kirandeep and Anu Garg, "Implementing Security on Android Application", International Journal of Computer Engineering & Technology (IJCET), Volume 4, Issue 2, 2013, pp. 576 - 589, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.

[20] Anirudha A. Kolpyakwar, Sonal Honale, Piyush M. Dhande and Pallavi A. Chaudhari, "A Review on Cloud-Based Intrusion Detection System for Android Smartphones", International Journal of Advanced Research in Engineering & Technology (IJARET), Volume 4, Issue 6, 2013, pp. 238 - 245, ISSN Print: 0976-6480, ISSN Online: 0976-6499.

[21] A.Edwinrobert and Dr.M.Hemalatha, "Behavioral and Performance Analysis Model for Malware Detection Techniques", International Journal of Computer Engineering & Technology (IJCET), Volume 4, Issue 1, 2013, pp. 141 - 151, ISSN Print: 0976 – 6367, ISSN Online: 0976 – 6375.