

International Journal of Control Theory and Applications

ISSN : 0974-5572

© International Science Press

Volume 10 • Number 26 • 2017

An Android Bug Detector Using Reverse Engineering

¹Kishore Raja K, ²Hemalatha J. and ³Sarahi Varma G. P.

¹Assistant Professor, Department of IT, SRKR Engineering College, Bhimavaram. email:kkrsrkrit@gmail.com

²Associate Professor, Department of IT, SRKR Engineering College, Bhimavaram. email: indukurihemalatha@gmail.com

³Professor, Department of IT, SRKR Engineering College, Bhimavaram. email:gpsvarma@gmail.com

Abstract: In day-to-day life most of the population in this world using Mobile Phones. Like computers, smart phones are likely to under malfunction as long they are in continuous operation. While 81% of the smart phone users are Android users. It is necessary to debug the applications before using them. In this paper authors are interested in Android Reverse Engineering Bug Detector (AREBD) technique to detect and recover the number of bugs by converting binary code to intermediate code. While Debugging we mainly concentrate on computer program, whether it is behaving as expected. Debugging tends to be difficult when various subsystems are tightly bound, as changes in one subsystem may cause bugs to other subsystems. Android Bug Detector is a tool that performs Android Application Debugging using manifest file in .apk file. It is a process of finding the malicious code and reducing the number of bugs in an android application. The proposed bug detector is implemented using Open source and is free of cost. It is slower in detection, but gives efficient results irrespective of capacity of .apk file.

Keywords: Android Apps, Debugging, Malfunction, manifest File, .apk file, Bug Detector

1. INTRODUCTION

In the recent days, popularity of new Android platform has resulted in an extensive rise in the number of reported vulnerabilities as a result number of threats are increased. Based on the estimation has been presented by Kaspersky's, the number of mobile malware aiming the Android platform nearly three times increased in 2013, reaching 98.1% of all mobile malware [1]. Malware is any Malicious code is included in an app will steal the confidential data from the mobile or corrupts it without the consent of the user. Some of the malwares are not dangerous because they created by authorized user to test the mobile apps; however, there have been some malwares may behave unexpectedly and create problems like steal data, services, denial of services, abnormal battery drains etc.

There are few numbers of Android debugging tools that are fully automated. They have their own advantages and limitations. A good number of the tools are used in manual presence. In order to make automated test effective, three issues should be considered. Firstly requirements of the test should be formulated properly, Secondly, select suitable tool for automation test. Lastly, process to test automation. It mainly concentrates on bugs which are caused during the development process and make app unsecure after it is released. APK file is nothing but java compiled file.

We can retrieve the source file back in two ways, one is *Decompiling* and another is *disassembling*. Decompiling is a process of getting the source code from binary code. We cannot get 100% but probably 80%. Disassembling is the process of getting intermediate code from binary code. The proposed system mainly checks if there is any sensitive data in shared preferences, possible SQL content, location info and possible script injection. For better understanding how reverse engineering works need to have knowledge about the how Android apps are built which is shown in Figure 1.

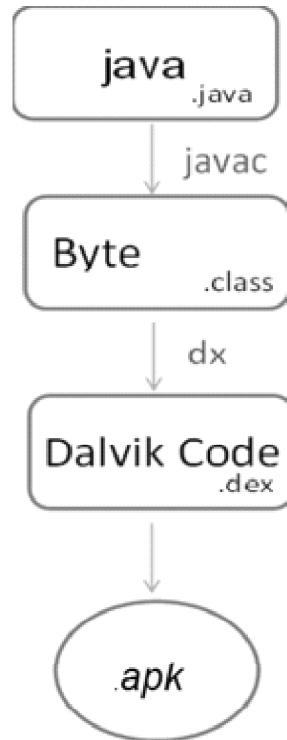


Figure 1: Algorithm for *xx.java* into *xx.apk* file

Reverse Engineering using apktool converts .dex code to the small version of the code this process is called disassembling. We can decompile using dex2jar and jad. The procedure of, reverse engineer of Android apps is done the following two phases.

(i) **Android App Dis-Assembling:** Apk files are extracted using unzip utility and then extract binary resources using ApkTool. It will helps in converting original java code with the help of .apk file use the following apktool command `d [app].apk`

(ii) **Decompiling Android Apps using dex2jar and JAD:** Android apps are generally written in Java language and are compiled to .class files. By using Dalvik virtual machine on an Android platform these java files are converted into a .dex files. The byte code of each app is packaged in an .apk file containing .dex files. The details of the package including the permissions information and other required resource les are described in manifest file called AndroidManifest.xml. The bundle of byte code and the associated data is called as binary container and is maintained in .dex file. It contains Meta-Data section and Data section. Meta-data section includes executable code as a header and also identifier lists that contain references to strings, prototypes, types, fields, classes and methods included in the app. Data section contains the information refereed in the references lists. The attributes and layout of the .dex file is shown in Figure 2. To get the Java Byte code firstly, `sh dex2jar.sh classes.dex` command is executed by giving dex2jar as an argument will generate a new jar file. Secondly, use

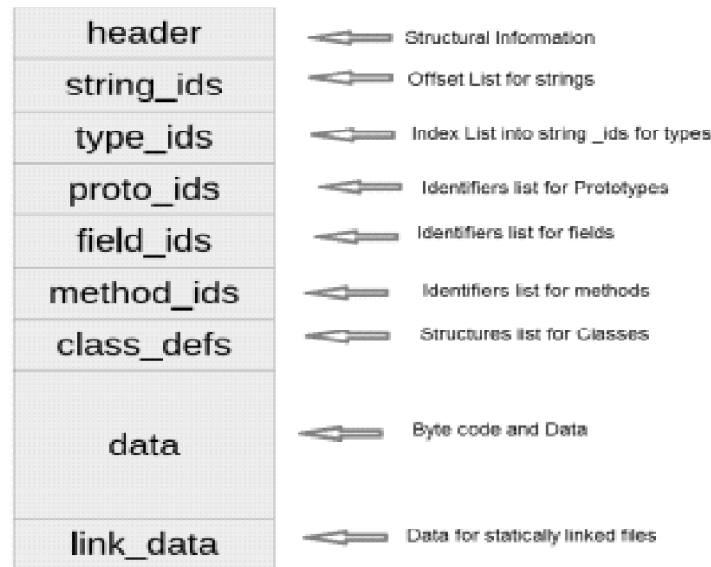


Figure 2: Attributes layout of .dex file

unzip command to derive *.class* files from the jar files. Lastly, decompile them in order to get the *.java* files. JAD is a Multi-platform close source command line decompiler used to get Java files.

The proposed system uses apps in smart phone, all the information is stored in the form of *.xml* files and they are readable. In case if the phone is lost, all the information is taken from these files so the data should be in encrypted form. Same will apply the data stored in the database. For this we check for the following strings like *putString*, *MODE_PRIVATE*, *MODE_WORLD_READABLE*, *MODE_WORLD_WRITABLE* in manifest file generated after disassembling. Secondly, we concentrate on the security during transmission through the network. Generally, we use protocol SECURE SOCKET LAYER (SSL) for secure transaction between client and server. During transmission there is a possibility of middle man attack. There are SSL certificates which tell this is secure connection. By including some tags like *Trust All SSL Socket Factory*, *All Trust SSL Socket Factory*, *Non Validating SSL Socket Factory*, there is a chance to make fake certificates and it tells to browser ignore about certificates and make secure with green https. There is a chance to find a location where you are if we find strings like *getLastKnownLocation*, *requestLocationUpdates*, *getLatitude*, *getLongitude* in manifest file. It also explains about which encryption method is used for transmission. If above mentioned strings are present, then a corresponding block is shown in red color border line. If they are not present, then a corresponding block is shown in green color border line.

2. RELATED WORKS

From last couple of years Mobile phones have become a critical information sources including personal information, images, videos and different online transactions in log files. In the recent days cyber-criminals are targeted on smart phone to steal enormous amount of confidential data through mobile apps. The Vulnerability and error detection of mobile app is an important criteria for the mobile user to overcome this problem [2]. There were different general studies contributing methods for malicious app detection. Malicious detection can be implemented broadly divided into (i) Detection prior to app installation (ii) App behavior monitoring directly on a mobile device. Among the studies in the first group are Risk Ranker [3], Droid-Scope [4] Droid Ranger [5] that enthusiastically observe mobile apps behavior. In view of the fact that, these techniques are computationally expensive for a resource-constraint environment of a mobile platform, they are mostly intended for an offline detection. Reverse engineering is the capability to disassemble the source code from an executable code. This

procedure is used to scrutinize the execution of a piece of code or to avoid security mechanisms, etc. 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.

Konstantin Knorr *et al.* [6] suggested ideal tool for monitoring and tracking long-term health conditions named as Mobile health apps. To implement threat analysis they proposed a testing method for Android mHealth apps.

Taranjeet Kaur Chawla *et al.* [7,8] and projected that Reverse Engineering is a process of converting app in to source code. In this process we use two methods Disassembling and Decompiling. Disassembling is a process of converting .apk to intermediate code which smali version. Decompiling is a process of converting .apk to .java which can be readable. We use APKTOOL for Disassembling and DEX2Jar and JAD for Decompiling.

Based on the analysis of User Interface (UI) information Charlie Sohet. *et. al* [9] has proposed a novel technique of detecting clones of Android application collected at runtime. Without the need of generating relevant inputs and execute the entire application, UI information can be collected easily by manipulating the multiple entry points feature of Android applications. Another advantage of our practice is concealing outfit hard-wearing since semantics preserving complicate technique do not affect runtime behaviors.

To identify known Android malware, Chen J *et. al* [10] examined the application of android clone detector and tested Android applications known to contain malware and a set of benevolent applications. By converting binary code of the applications in to the Java source code and use NiCad, classes of clones in a small subset of the malicious applications are identified by near-miss clone detector. By using these clone classes as a signature, Chen J *et. Al* [10] are become aware of similar source files in the rest of the malicious applications.

Vendors of automatic Bug detecting tools for mobile applications usually offer specific modules or supplementary tools as part of their products design. There are also a couple of open source tools that are used to automate your mobile application tests.

Some tools like Robotium [14] as shown in Figure-3 is a free Android User Interface testing tool. With this tool test case developers can write modules and acceptance test scenarios. The main limitations with this tool is incompatible for interaction with system software; Smart phone or a tablet cannot be locked or unlock. Record and Play utilities are not available in Robotium and it is available commercially.

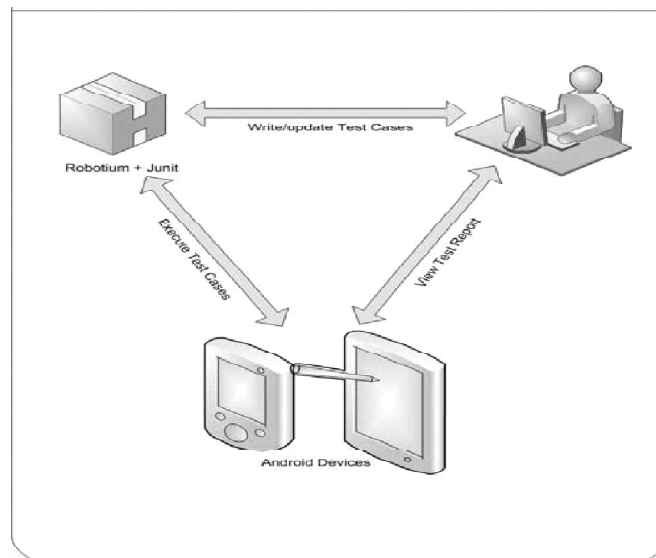


Figure 3: Architecture of Robotium Debugging tool

In-order to automate test, Monkey-Runner [15] as shown in Figure-4 is One does not have to deal with the source code. The tests are implemented in Python. It is low-level approach when compare to Robotum [14]. One may use a recording tool for creating tests. The main drawback is that it is necessary to write scripts for each device. Another drawback is that the tests require adjustments each time when user interface of the tested program is changed.

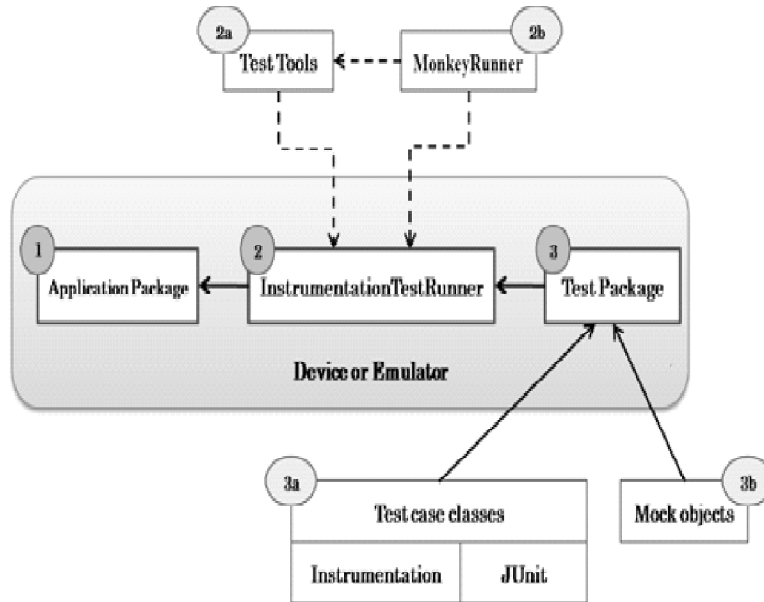


Figure 4: Architecture of Monkey Runner Debugger

One of the advantage of Ranorex [16] as shown in Figure. 5 is that it will generate comprehensive information about bug with screenshots. This tool is compatible with a smart phone or a tablet to the Internet via WIFI. But it is not a open source tool. Ranorex searches elements rather slowly; it takes up to 30 seconds to perform such an operation. One must device APK files for Ranorex. Otherwise, it is not possible to automate tests by mean of tool, as it works only with instrumented APK files.

There are very few debugging tools currently available for developers and the existing tools are also only used for checking general vulnerabilities like security loopholes. The person or organization which developed a particular app cannot check its permissions with the existing debugging tools.

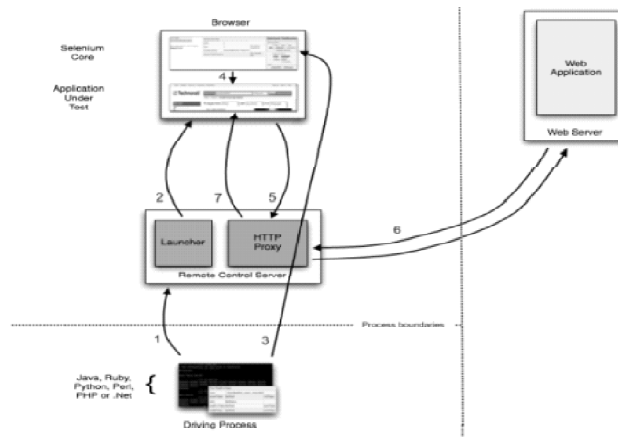


Figure 5: Architecture of Ranorex Debugger

Limitations in Existing Systems

- ❖ Most of the existing Android application debugging tools created are not platform dependent.
- ❖ They are available commercially in market.
- ❖ Once the application is developed and in on use. It is difficult to debug it with these systems that are they are only efficient during development phase.

3. ANDROID REVERSE ENGINEERING BUG DETECTOR (AREBD)

The Android Bug detector is a tool that performs Android Application Debugging. It is a process of finding the section of malicious code called bug and reducing the number of bugs. Debugging is done on a given Android App (APK) by using disassembling technique. It will decode the apk byte code in to java code and checks for malicious code in manifest file. Apk tool is used to convert .apk file into .java file. It is purely version based they can only convert that version supported apps. The main strength is that it is user friendly and will detect the bugs effectively, but it consumes more processing time.

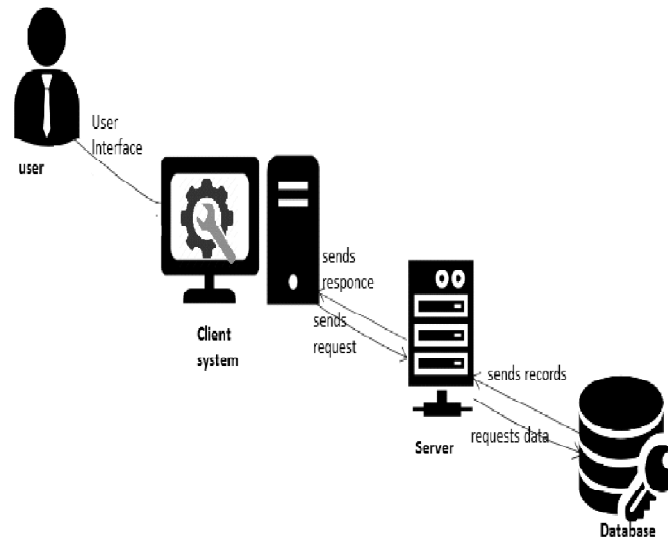


Figure 6: Android Reverse Engineering Bug Detector (AREBD) architecture

AREBD Algorithm

Input: “.apk” file of Android App

Output: Bug Report

Step-1: Client which is user send request to the server for knowing whether his app is secure are not.

Step-2: Server Decompile the binary code to Source code and stores in specific file. For Decompile we use DEX2JAR and JAD.

Step-3: There is a multidimensional array which contains different tags that should not present in the intermediate code.

Step-4: For each String of a particular type in array

{

Boolean b=Search(\$file,String);

```
//checks string of particular type is present
If(b)
{
    Print(string in block with red color border);
    // There is a tag which leads to attack
}Else
{
    Print(string in block with green color border); // safe
}
}
```

Step-5:Store result in database for particular username and send result back to user.

4. EXPERIMENTAL RESULTS & ANALYSIS

The Android Bug detector is a tool that performs Android Application Debugging. AREBD system is tested on four well popular android Apps (i.e. Shareit, whatsApp, Pwndroid and Sony ericsson-notes) and each app's manifest file is having their own memory capacity.

Software Requirements

- Language : PHP.
- Operating System : Ubuntu 14.04.
- SDK : Android SDK.
- Server : Apache tomcat,
- Database : MYSQL.

Hardware Requirements

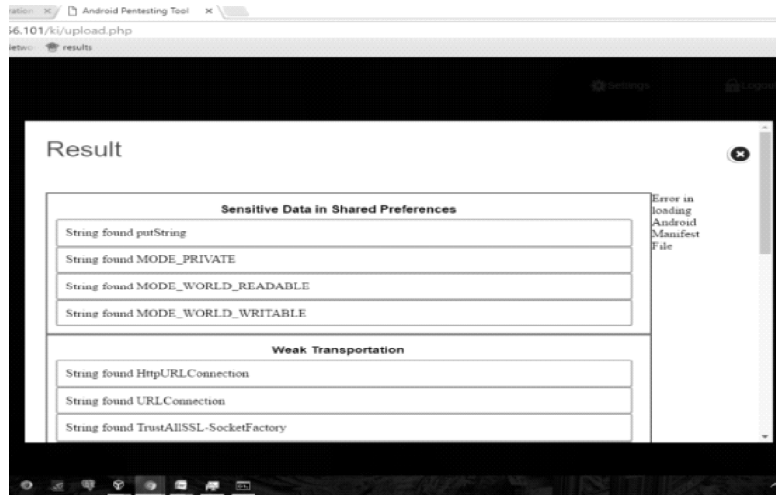
- System Processor : Intel Core 2 Duo
- System speed : 1.2 GHz
- System hard disk : 80 GB
- System RAM : 2GB

The AREDB bug detector mainly implemented in three stages. Firstly, detector converts binary code into intermediate code and is called disassembling. Secondly, bug detector cross checks all the tags in the intermediate code with the standard tags library which may lead to misbehavior of App.

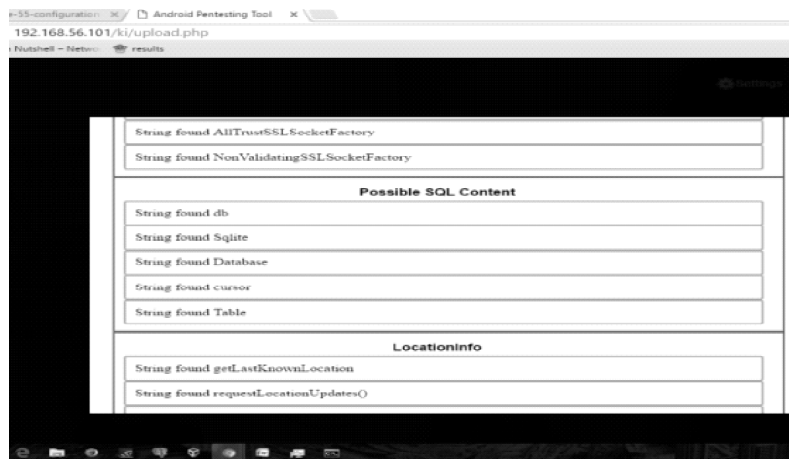
Finally, Detector will report the bug If any unknown tag is identified by it. Most of the Android apps are implemented with standard tag belongs to following libraries,

- o *Sensitive data in shared preferences*
- o *Weak transportation*
- o *Possible SQL content*
- o *Location Info*
- o *Possible Script Injection*

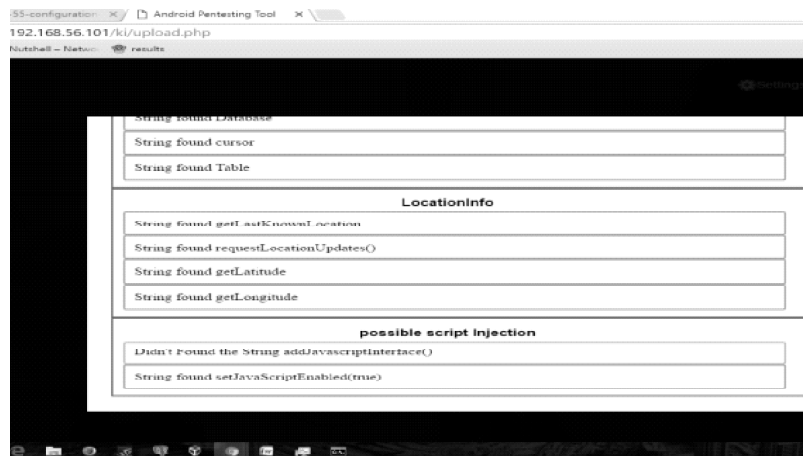
If the App is implemented with tags are not belongs to standard library will misbehave. With a reverse engineering byte code of “.apk “ file is converted into intermediate code to get the tags used in the App.



(a)



(b)







(c)

Figure 6 (a), (b), (c). Debugging results generated by REDB for ShareIt App.

The results obtained from the bug detector for debugging different mobile Apps shown in Table-I. It is observed that if number of tags mismatches will increase debugging time.

Table I
Debugging time of different Android Apps generated by REDB.

APK Details	Size of APK (MB)	Detection Time (Min)
 ShareIt	5.8	5.92
 whatsapp	2.5	4.25
 pwndroid	8.5	4.0
 Sony ericsson.note	2.5	6.2

The results obtained from Android Reverse Engineering Bug Detector(AREBD) it is evident that the time taken to complete the bug detection is not based on the size of the app.

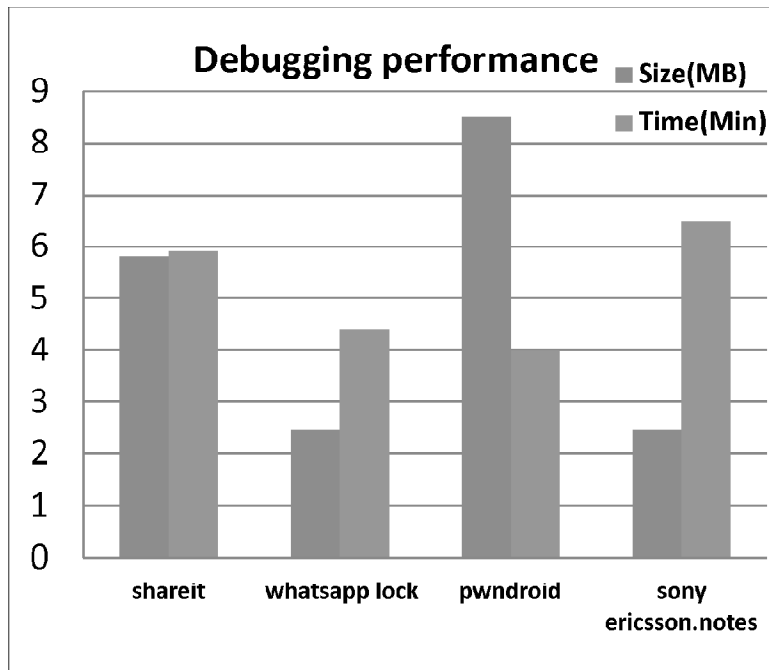


Figure 7: Debugging Time stamp of different Android Apps generated by REDB

The outcome mainly depends on the time taken to *disassembling*. In disassembling detector converting binary code to intermediate code. From intermediate code bug detector cross checks all the tags in the app with standard tags library. If any unknown tag is identified by the bug detector will report the bug

5. CONCLUSION

It is evident that Android Reverse Engineering Bug Detector (AREBD) is an alternative to identify the misbehaving and malfunctioning android apps before installing them in any smart phone. It will detect malicious code in manifest file to identify type of bug in the .apk file. Though it is slower in detection, but gives efficient results irrespective of capacity of .apk file. The proposed bug detector is implemented using Open source and is free of cost. Further, Bug detector efficiency can be improved by reducing the bug detection time.

We bestow our sincere thanks to our beloved family embers and friends for assisting, guiding and motivating us throughout this journey.

Me would love to express our deepest gratitude to Ws. Debashree Ghosh for her unwavering support and mentorship throughout this project.

We would also like to show our appreciation to Ms. Allan Mary George for reinforcing us in all possible aspects. Thank you all.

REFERENCES

- [1] “Extending Existing Android Applications”. Online:<http://blog.inyourbits.com/2012/11/ extending existing- android-applications.html>. 2012.
- [2] J. Boutet, http://www.sans.org/ reading_room/whitepapers/malicious/malicious-androidapplications-risks-exploitation_33578, Date Accessed: 2011 June.
- [3] M. C. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. Risk ranker: scalable and accurate zero-day android malware detection. In The 10th MobiSys, pages 281-294, 2012.
- [4] L. K. Yan and H. Yin. Droid scope: “Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis”. 21st USENIX Conference on Security Symposium, Security’ 12, pages 29, Berkeley, CA, USA, 2012. USENIX Association.
- [5] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You “ *Get off My Market: Detecting Malicious Apps in official and Alternative Android Markets*”, In 19th Annual NDSS, 2012.
- [6] Konstantin Knorr, David Aspinall “Security testing for Android mHealth apps “, Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on 13-17 April 2015, Pg No:1-8, DOI: 10.1109/ICSTW.2015.7107459.
- [7] Taranjeet Kaur Chawla, Aditi Kajala “*Transfiguring of an Android App Using Reverse Engineering*”, IJCSMC, Vol. 3, Issue. 4, April 2014, pg.1204 –1208.
- [8] Rufatet Babakishiyev “*Software Developer, Reverse Engineering an Android Application*”, IEEE International Conference, 2013.
- [9] Charlie Soh, Hee Beng Kuan Tan, Lipo Wang “*Detecting Clones in Android Applications through Analyzing User Interfaces*”, IEEE 23rd International Conference on Program Comprehension, DOI: 10.1109/ICPC.2015.25, Page(s):163 - 173, ISSN : 1092-8138.
- [10] Chen J, Alalfi MH, Dean TR et al. “*Detecting Android malware using clone detection*”, Journal Of Computer Science And Technology 30(5): 942–956 Sept. 2015. DOI 10.1007/s11390-015-1573-7
- [11] Siegfried Rasthofer, Steven Arzt, Marc Miltenberger, and Eric Bodden “*Reverse Engineering Android Apps With Code Inspect*”, Workshop on Innovations in Mobile Privacy and Security IMPS at ESSoS’ 16, London, UK, 06-April-2016.
- [12] Syed Arshad , Ashwin Kumar “*Android Application Analysis using Reverse Engineering Techniques and Taint-aware Slicing* “, International Journal of Computer Applications, Pg No:0975 – 8887, International Conference on Information and Communication Technologies (ICICT- 2014)
- [13] Sudipta Ghosh, S. R. Tandan, Kamlesh Lahre “*Shielding Android Application Against Reverse Engineering*”, International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, www.ijert.org Vol. 2 Issue 6, June - 2013.
- [14] <https://en.wikipedia.org/wiki/Robotium>, Feb 2016.
- [15] <http://antoine-merle.com/introduction-to-the-monkey-runner-tool-2/>, Jan- 2016.
- [16] <http://www.ranorex.com/>, Feb-2016.