ENHANCING ANDROID OPERATING SYSTEM SECURITY BY BLOCKING SYSTEM CALLS OF MALICIOUS APPLICATIONS USING REFLECTION

A Dissertation Proposal
submitted
By

Sabir Ali

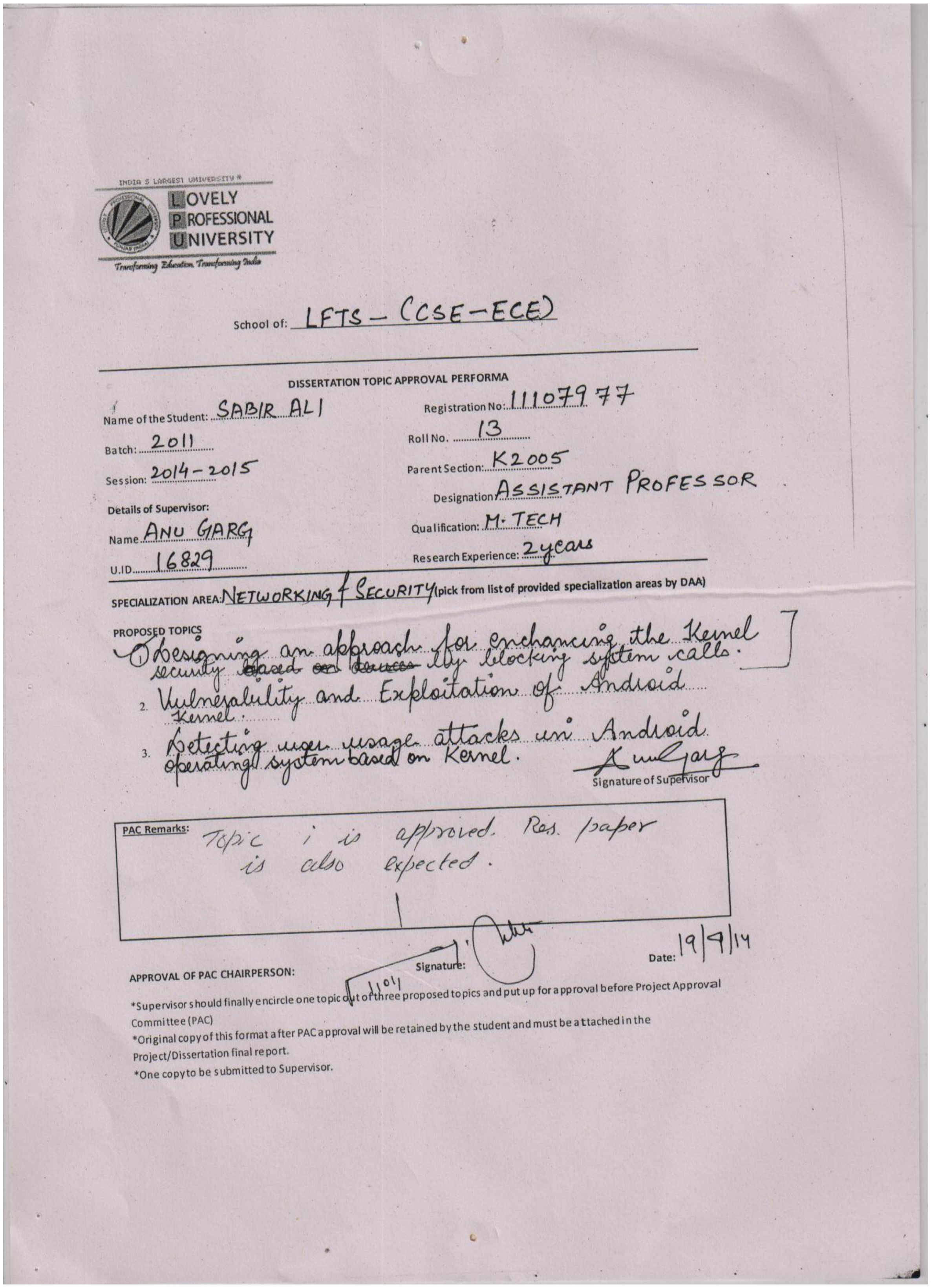
To
Department of Computer Science Engineering

In partial fulfillment of the Requirement for the
Award of the Degree of

Master of Technology in Computer
Science and Engineering

Under the guidance of
Ms. Anu Garg

(May,2015)



INDIA'S LARGEST UNIVERSITY*
LOVELY PROFESSIONAL UNIVERSITY
Transforming Education, Transforming India

School of: LFTS – (CSE–ECE)

DISSERTATION TOPIC APPROVAL PERFORMA

Name of the Student: SABIR ALI	Registration No: 11107977
Batch: 2011	Roll No. 13
Session: 2014–2015	Parent Section: K2005
Details of Supervisor:	Designation: ASSISTANT PROFESSOR
Name: ANU GARG	Qualification: M. TECH
U.ID: 16829	Research Experience: 2 years

SPECIALIZATION AREA: NETWORKING & SECURITY (pick from list of provided specialization areas by DAA)

PROPOSED TOPICS

1. Designing an approach for enchancing the Kernel security ~~based on~~ ~~[illegible]~~ by blocking system calls.
2. Vulnerability and Exploitation of Android Kernel.
3. Detecting user usage attacks in Android operating system based on Kernel.

[illegible]
Signature of Supervisor

PAC Remarks: Topic 1 is approved. Res. paper is also expected.

APPROVAL OF PAC CHAIRPERSON: Signature: [illegible] Date: 19/9/14

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

DECLARATION

I hereby declare that the dissertation proposal entitled, "**Enhancing Android Operating System Security by Blocking System Calls of Malicious Applications Using Reflection**" submitted for the M.Tech Degree is entirely my original work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: **Investigator**

Regn No.:- 11107977

ABSTRACT

Android Operating System is emerged as most popular mobile Operating System in this decade. Millions of Android devices are activated each day. The popularity also come across thousands of malwares spread from various Appmarkets. Android Operating System consists of various layers. The application layer relies upon the Linux Kernel layer in various security aspects. We analyzed system call pattern of various applications in four category like gaming, utility, social applications and media based applications. The analysis is based upon the Reflection Method call and reflection field access parameter. We found a pattern between privacy leakage level and reflection method call percentage of any application. It leads to detect malware in Android platform.

Keywords: Malware, Reflection, System Call, RMC,RFA

CERTIFICATE

This is to certify that **Sabir Ali** has completed M.Tech dissertation proposal titled **"Enhancing Android Operating System Security by blocking system calls of malicious Applications using Reflection"** under my guidance and supervision. To the best of my knowledge, the present work is the result of his original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma.

The dissertation proposal is fit for the submission and the partial fulfillment of the condition for the award of **M.Tech in Computer Science and Engineering**.

Date: 27th April,2015

Signature of Advisor
Name: Ms. Anu Garg
UID: 16829

ACKNOWLEGEMENT

The report has been written with the kind guidance and support of my mentor. The satisfaction and happiness that accompany the successful completion of any task would be incomplete without mentioning the names of people who made it possible, whose constant guidance and encouragement crowns all efforts with our success.

I would like to express my deep gratitude and thanks to my mentor **Ms. Anu Garg,** Asst.Professor, as well as **Mr**. **Dalwinder Sir**, HOD, School Of CSE, Lovely Professional University, Phagwara, Punjab for their help and guidance throughout my dissertation work. I have received an enormous amount of valuable advice and knowledge from her that helps me a lot in my work. Thank you, **Ms. Anu Garg**, for providing me motivating support in my research work.

Table of Contents

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In last few years, Android has emerged as the most popular mobile operating system. In 2013 it crossed 1 billion mark and millions of Android devices are activated each day throughout the world. It reaches almost 75% market share in 1st quarter of 2014. With the increasing popularity, Android also became a soft target for malware and viruses. Android's current security model is roughly categorized as Application level security and Kernel level security. In Application level, security is based on Android permission model and Application Sandboxing. These are enforced with controlled access of Application components, system resources etc. While Android permission model is used by the Application developers to specify the permission their Applications are going to use.

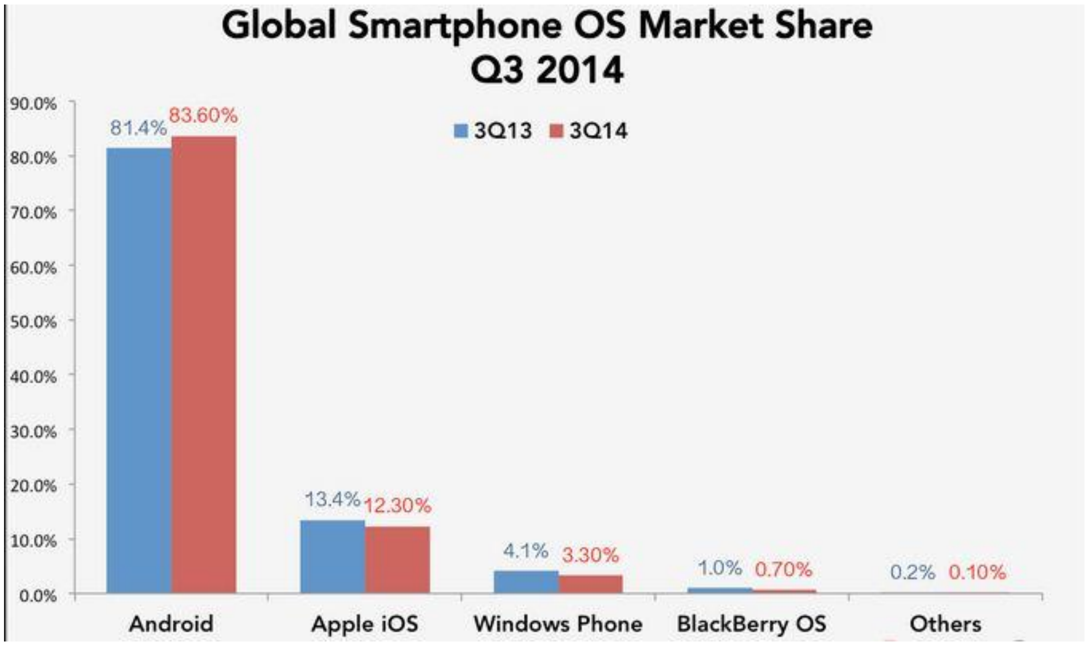


Figure 1. Market share of Android Operating System

In Kernel level, Application Sandboxing is achieved by Linux Kernel. In Kernel level, the Linux Kernel DAC (Discretionary Access Control) which is responsible for restricting the access facilitated by an Application. It authorizes the system resources to an Application and isolates the Applications from each other. In Kernel level, the interaction mechanism between various layers while forking a system call is exposed too much vulnerability. In The Android security framework within Linux Kernel, the Zygote Socket shows vulnerability by allowing malicious Application to fork new process, there is no such mechanism to checking requesting process id.

In this research we suggest solution of malicious inter-process communication between Linux Kernel and Android Application Frame work. The module provided by Alessandro et al is KernelCallController which checks and rules insecure and harmful interaction between the Android Application Framework and Linux Kernel .But it cannot completely detect the malicious interplays if the requesting process have sufficient permissions . As the Activity Manager Service allows any requesting Application to launch new process which have permissions to do so.

The malicious applications spread through Google Play Store and other third party app markets. Earlier check mechanism was not there for publishing an application to the app market .Google introduced a Bouncer tool to put check on the malicious application but it was not effective. The online application makers tools like AppMkr, Appypie are also injecting adware codes to applications .These possess real threat to the user data. The rapid increasing of the malware is because of the popularity of the Android devices and openness of the Android platform. When a malware in installed in a device it may not activated instantly, some malwares are activated with user interactions, some need to update themselves and some malware simply activated with installation.

In this paper we provide patterns of system calls of malicious applications as well as goodware or normal applications. In our research we took 100 Android applications of four categories like gaming, social, utility and media based applications where we applied reflection field access method and reflection method call percentage to determine the system call pattern and malicious interplays of an application.

1.1 Android

Android is an open source platform, designed for handset devices. They are widely spread in the market today and found mostly in mobile phones and tablets. Android Inc led by Andy Rubin started Android Project in 2006 later on Google acquired Android Inc and started ASOP project with OHA group led by Google. The unveiling of the Android platform on 5 November 2007 was announced with the founding of the Open Handset Alliance, a consortium of 34 hardware, software and telecom companies devoted to advancing open standards for mobile devices. The OHA allows phone makers to run Android on a suitable handset, without charge. In September 2008, T-Mobile released the first smart phone based on the Android Platform as well as a Software Development Kit (SDK). The open nature of Android and easiness to create custom application makes it market leader and popular one. A survey of mobile platforms market share early in 2014 indicates that the largest market share is contributed by Android phones. For developer, it gives opportunity to develop apps for a fairly new market that is booming on a daily basis. With so many users, it's never been easier to write an application that can be downloaded and used by real people. Android gives developers a way to develop unique, creative applications and get those applications in the hands of user's. Users don't have to go searching the Internet to find an app to install. They just simply go to the Android Market that is preinstalled on their device, and they have access to all apps .Google also provides online tutorials and Application Programming Interfaces , which makes Android very developer-friendly and easy to develop. Along all these Google also provides equal opportunity on neutral platform for all developers from large companies to small and medium enterprises and also for freelancers.

1.2 Android Architecture

Android is not just Mobile Operating System but is a software stack that includes the middleware and a number of layers.

- **Application Layer (A).** Application layers reside at the top of the stack and comprised of both user and system applications which have been installed and execute on the device. Each Android application is created with set of components which performs a different role in the logic of the application.
- **Application Framework Layer (AF).** By providing an open development platform, the Application Framework provides the main services of the platform that are exposed to applications as a set of APIs. This layer provides the System Server that is a component containing the main modules for managing the device (e.g. Activity Manager and Package Manager) and for interacting with the underlying Linux drivers.
- **Android Runtime Layer (AR).** This layer consists of the Dalvik Virtual Machine (DVM), i.e. the Android runtime core component that executes application built in the Dalvik Executable format (.dex) which is converted from .class file of Java.
- **Libraries Layer (L).**
  The next layer is the Android's native libraries. It is this layer that enables the device to handle different types of data. These libraries are written in C or C++ language and are specific for a particular hardware.
  Some of the important native libraries include the following:
  - ➢ **Surface Manager:** It is used for compositing window manager with off-screen buffering. Off-screen buffering means you cannot directly draw into the screen, but your drawings go to the off-screen buffer. There it is combined with other drawings and form the final screen the user will see. This off screen buffer is the reason behind the transparency of windows.

- **Media framework:** Media framework provides different media codecs allowing the recording and playback of different media formats.
- **SQLite:** SQLite is the database engine used in Android for data storage purposes.
- **WebKit:** It is the browser engine used to display HTML content.
- **OpenGL:** Used to render 2D or 3D graphics content to the screen.

- **Kernel layer (K).** . The whole Android OS is built on top of the Linux 2.6 Kernel with some further architectural changes made by Google. It is this Linux that interacts with the hardware and contains all the essential hardware drivers. Drivers are programs that control and communicate with the hardware .These functionalities includes accessing to physical resources (i.e. device peripherals) and ii) the Inter-Process Communication (IPC) which is further performed via Intents. Device peripherals (e.g. GPS antenna, Blue- tooth/Wireless/3G modules, camera, accelerometer) are accessed through Linux drivers installed as kernel modules.

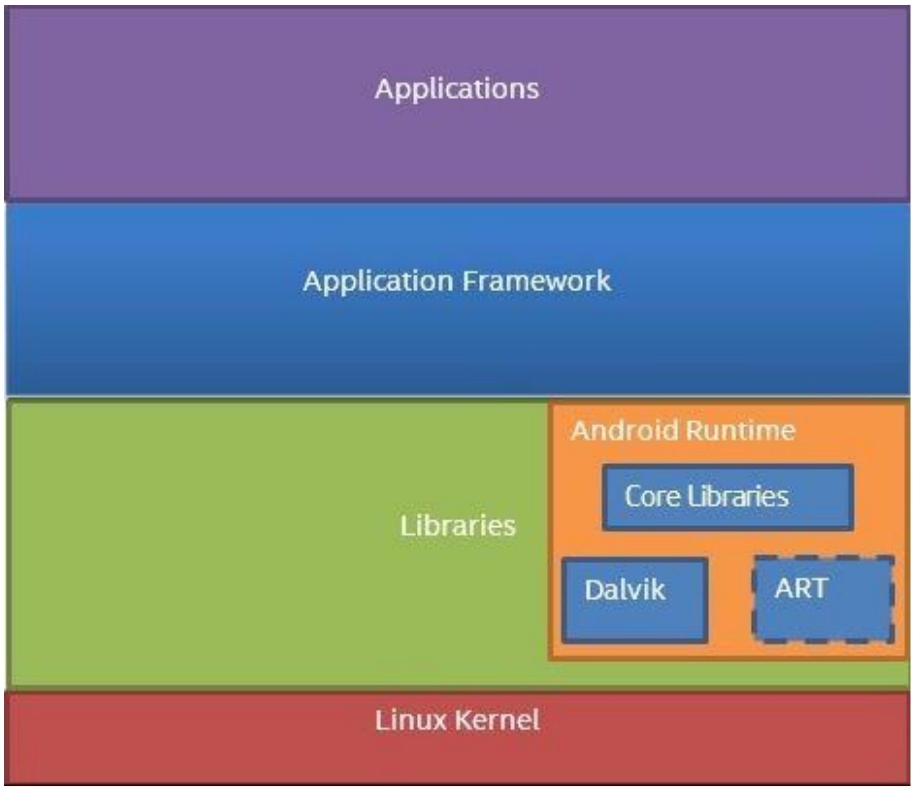


Figure 2. Android Framework

for process management Android Runtime uses DVM instead of JVM.Because JVM needs heavy computing resource which will have serious impact on system performance. To solve this problem, DVM is specially designed to replace JVM only for the Android platform. DVM provides process isolations via application sandboxing which are viewed as one of security enforcements points in the Android platform.

The Android Application framework defines the system services and developer APIs. The system service is responsible for low level functionalities while the Application framework APIs makes system components reusable. Android Applications defines its own static permissions at Manifest file which is used to govern the access between application components and resources.

1.3 Android Platform

Android platform implements the permission based security that develop a framework in order to address the security issues to prevent applications from stealing private data, maliciously disrupting other applications or the operating system itself. Each application can expose a subset of its functionalities to other applications if they have been granted the corresponding permissions. Developers are encouraged to take full advantage of Android's features when writing their own applications.

1.4 Components of the Android Application

An Android Application is built up from various types of components. We discuss the main components used in Android Application. These are also called building blocks of an Android Application. These components are Activity, Content Provider, Broadcast Reciever, Services and Intent. Acitivities are responsible for reacting to the operations and user actions that a user have performed on the interface

1.4.1 Activities

Activities represent the single screen which the user interacts with. It is also responsible for reacting to the operations and user actions that a user have performed on the interface. The activity life cycle include different phases. It starts with onCreate() and ends while onDestroy() method is invoked. After creation, onStart() is the first point which user can see an activity on the mobile screen. onResume() also shows a phase the activity is visible but it recovers the old state. onPause() represents a state that the current activity is shifted to background and can be on focus at any time .In onStop() phase the activity is still alive but window manager detached it and it cannot be on focus further. If an application has more than one activity, then one of them should be marked as the main activity which represents the start of the application .Main activity is the activity which starts with the launch of the application. An application can create another activity thus an application can contain numbers of activity inside. Whenever a new activity is created and enter into start phase, the old one will not be destroyed its state is pushed into the stack. The old activities are restored by retrieving its state and regain the focus whenever user navigates back.

1.4.2 Services

Services work quite similar to activities .A service is a component that runs in the background to perform long-running operations. It doesn't have any graphical interfaces. A service can be invoked in two different manners. First Directly invoking the method, startService(), which starts the service exits whenever the background task is finished. The other way to start a service is with application bindings. A bound service is linked to an application so the application has to decide the fate of the service.

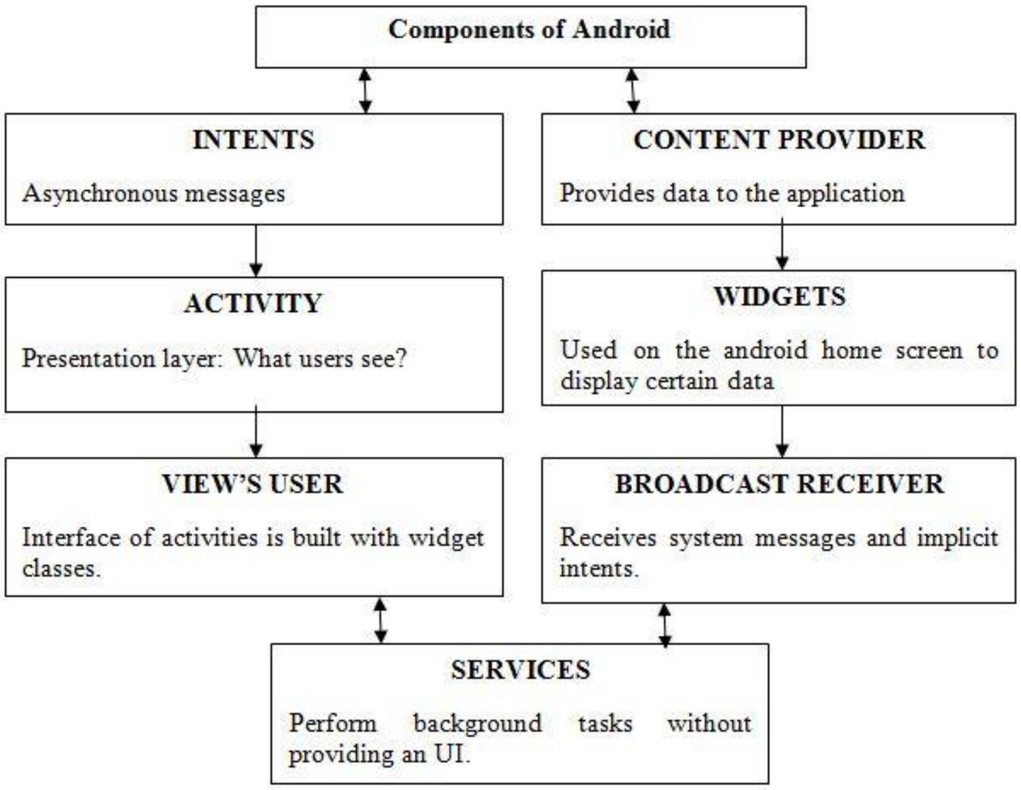
1.4.3 Content Providers

A content provider component acts database provider from one application to others on request. Such data access requests are handled by the methods of the *ContentResolver* class.

The applications are also eligible to access public contents provided by Google. While storing data to a content provider, the developer needs to specify the naming of the data by the Uniform Resource Identifier (URI) scheme so that the data can be easily retrieved by name.

1.4.4 Broadcast receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. They are used to activate services at some point. For example, an application want to let other applications that a new media file is added in SD card or phone memory and that is available for them to use, so only the broadcast receiver who will intercept the communication and initiate proper action. The notification message sent between to components is called intent, which is the communication medium in Android Application.. Intent filters used for filtering unwanted intents so that activities are informed by required ones only.



Components of Android
INTENTS
Asynchronous messages
CONTENT PROVIDER
Provides data to the application
ACTIVITY
Presentation layer: What users see?
WIDGETS
Used on the android home screen to display certain data
VIEW'S USER
Interface of activities is built with widget classes.
BROADCAST RECEIVER
Receives system messages and implicit intents.
SERVICES
Perform background tasks without providing an UI.

Figure 3. Components of Android Application

1.5 Configurations of Android Applications

The AndroidManifest.xml main configuration files of an Android Application. It states al the permissions the application is going to use as well as the resource configuration. Android having a security model, permission based that by default denies access to features or functionality that could negatively impact the user experience, the system, or other applications installed on the device. As to the Android permissions, it states the permissions it requests for installation as well as permissions that are defined to protect the application components.

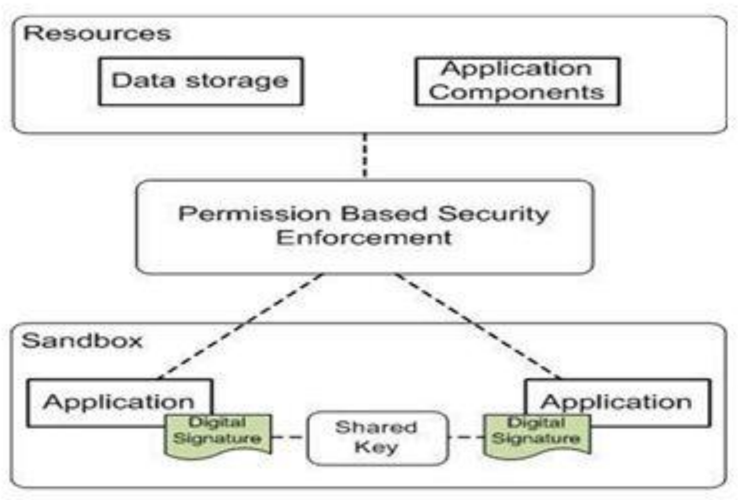


Figure 4. Android Security Configuration.

1.6 The Android security

The Android security combines of Linux kernel and in the application framework layer. The security feature inherited from Linux kernel is the userID, which is assaigned for each application running in the Android device. Apart from this kernel level, at the framework level, a mandatory access control is used by Android permissions to restrict access between components.

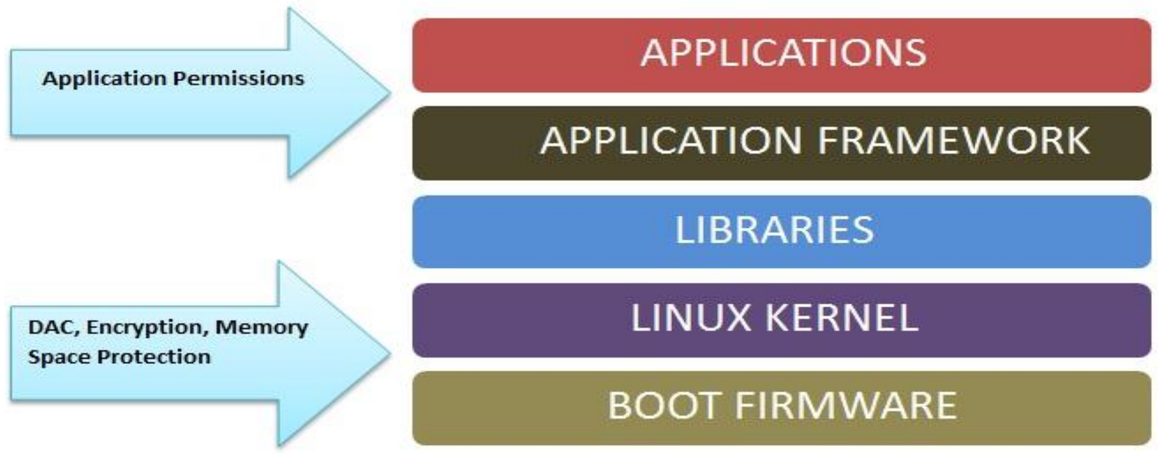


Figure 5. Security Aspects in Android Layers.

1.6.1 Linux Kernel

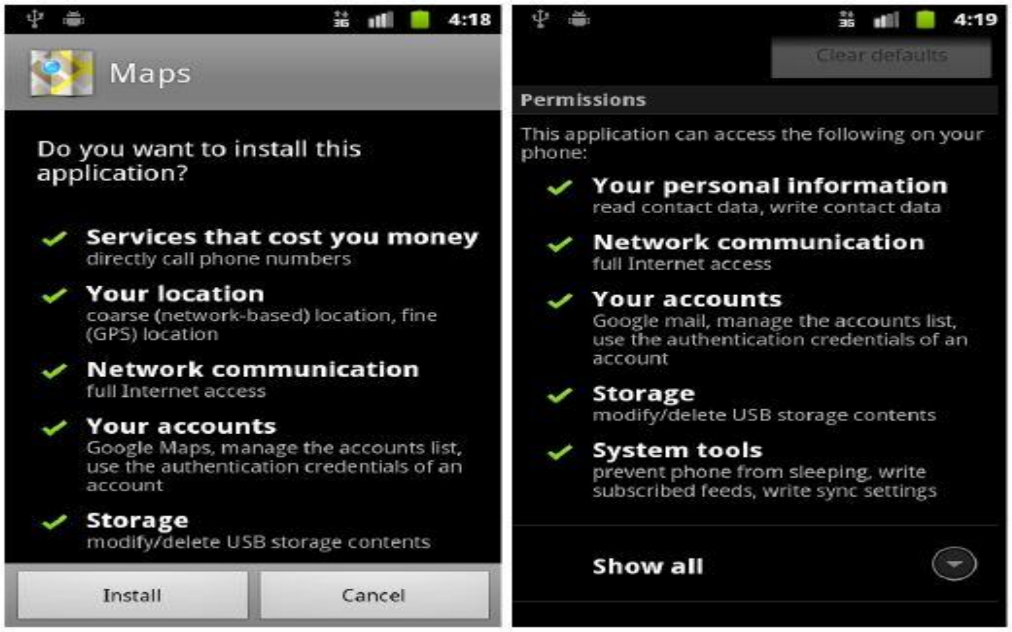
Each application is treated as individual process in Linux kernel. Linux introduces some useful features inside android system. It does not use JVM, instead it uses DVM. The DVM runs executable file in dex format. The Android application compiled by the Java compiler, to run in an instance of DVM, the system required to convert the compiled Android program to dex executable file. A tool called dex in Android Software development kit (SDK) is present inside. The DVM provides code isolation to mitigate the damage that can be caused by vulnerable applications. For multiple packages inside an application, a shared UserId is provided to all, which determines a shared process for them to run.

1.6.2 Android Applications Signings

Android System requires each and every application to be signed. The main purpose of application signing is to protect a application from its repackaging. Each developer who want to publish their application signs the application using own private key. These keys are kept secret. Once a signed application is installed on the device, the system is capable to use its signature information to differentiate it from other application.

1.6.3 Android permissions

Permissions are the main characteristics of the Android security. They are mainly used to control the access of resources and states from one application component to another. All the permission required by an application is granted at the time of installation. For granting permission to an Application, it should be requested via Android manifest file when specifying the properties for an application. The system



Maps

Do you want to install this application?

- Services that cost you money
  directly call phone numbers
- Your location
  coarse (network-based) location, fine (GPS) location
- Network communication
  full Internet access
- Your accounts
  Google Maps, manage the accounts list, use the authentication credentials of an account
- Storage
  modify/delete USB storage contents

Install Cancel

Clear defaults

Permissions

This application can access the following on your phone:

- Your personal information
  read contact data, write contact data
- Network communication
  full Internet access
- Your accounts
  Google mail, manage the accounts list, use the authentication credentials of an account
- Storage
  modify/delete USB storage contents
- System tools
  prevent phone from sleeping, write subscribed feeds, write sync settings

Show all

Figure 6. Permissions of Android Application

then take decision to granting or denying the permission. After the application has been started, permission checks are enforced before the actual component and resources are accessed take. The name of permission must be unique and descriptive so that other components are able to know and request it by name.

1.6.4 Vulnerabilities

Android do cover a significant range of the vulnerabilities and risks that may be exploited on the Android OS:

1.6.4.1 Anybody can make an app

As the Android platform is free and developing an app is easier so the developer can free over their Android domain without restriction.

1.6.4.2 The Android Market

The official app market called “Play Store” is the largest distributor of malicious application; earlier Google introduced Bouncer application to control and check the number of malware-infected apps.

1.6.4.3 Malicious code insertion

The data transfer between two applications is governed by a protocol of implicit and explicit intents .If the resource is targeted and ask for user action as the target application then the gets the data dictionary in data loss.

1.6.4.4 Third party applications

The Android Application are mainly downloaded from play store which is the official Android Application market .The other third party app developers are prone to virus and can be potential malware . This happens despite the development of such security measures for web app versions.

1.6.4.5 Rooting

Rooting features allows the user to custom the device. The process of getting into full access is compromised with security. Additionally, root is a very basic exploit used by malicious applications to gain system-level access into your Android device.

We designed a check mechanism on Application requesting to AMS for binder call to fork new process. If the contents and logs of the Application is properly matched with the permission it has been granted then only it will be allowed to start the launching process. Through this we reduce the malicious interplay started by any Application in Kernel layer.

We will check the logs and contents verily before allowing it to start a new process. The NotepadTest first check the appropriateness of the contents and logs of the Application with algorithms and match it with the new process and permission it acquired.

CHAPTER 2
LITERATURE REVIEW

There are extensive research on Android Operating System is conducted in recent few years. Most of these are based on the Android Application Security. Enck et al started research on Application security with TaintDroid which checks sensitive information flow in Android Apps. Access control also checked in StowAway,PSCount and Barrena et al analyzes third party apps with organizing maps .Many Applications are built for vulnerability analysis of Android like Pios' AppFence, TISSA' CleanOS. Several modules and research has done to check inter-process mechanism and runtime interplay between apps to avoid privilege escalation problem

Xiali Hei et al found vulnerabilities in Android Honeycomb. Where location based on and offs are not checked, which may create Kernel overflow and accessing arbitrary memory. Other vulnerability is found with no checking mechanism of IOC_Size(cmd) in driver package which can create DOS attack on Kernel .The authors fixed the vulnerability with proper cheek and also released patch for the same.(Xiali Hei et al,2009).

Alessandro et al found vulnerability in IPC of Android Application in Kernel layer .It is the Zygote Socket which doesn't check the UID of the requesting process as long as it has originated from a valid static class. Authors have shown it is possible to force the zygote process to fork, generate dummy processes which are kept alive in Linux Kernel .Thus flooding the Zygote socket with requests of large number of dummy process can be made until all memory resources are exhausted. It can make the device unresponsive. The proposed counter measures by checking the UID of requesting process and reducing the permission of Zygote Socket (Alessandro et al ,2011).

Shewale at el analyzed various Android vulnerabilities and modern mitigation techniques in their paper. They classified the vulnerabilities found in the Android according to the layers of Android architecture. They also assessed the external drivers for exploitation. The vulnerabilities found in the Application Framework layer are mainly of DoS attack, escalation of privilege and permissions and unauthorized access to code of various drivers.

The Application layer shows vulnerabilities mainly with browsers and WebView. Many Applications contain bugs and are unable to restrict users to inject codes in cookies, sessions in browsers. The external drivers from various OEM manufacturers also contain exploits inside their Kernel driver codes. Generally CPU drivers and Graphic card drivers from companies Qualcomm shows memory corruption issue (Shewale at el,2012).
In case of **SELinux**, which uses Android Application Sandboxing and MAC (Mandatory Access Control) in Linux Kernel ,is first updated in Android Operating system in Jelly Beans, Android 4.3 version. This enforces security in Kernel module by removing "setuid" feature which restricts the low privileged processes to execute high privilege modules.

Erika Chin et al examined Android's Inter Application message passing system is vulnerable for surface attack. In their paper, the shows the risks of inter application communication and identified insecure practices from developer end. If sender does not specify the recipient correctly, then any malicious attacker can interpret the message can do harmful actions with it. If the system does not restrict or verify who having privilege to send a message, then an attacker could modify the message also. The paper specifically checked Intent based attacks on communication surfaces and chosen external intents as vulnerable for attacks like intent hijacking, intent spoofing. They developed ComDroid tool for detecting vulnerabilities inside an Android Applications. ComDroid checks the parsed dex file output and records potential component and vulnerabilities of external intent. It shows that many useful applications are also vulnerable to attack and developers should be more precautious.
(Erika Chin et al, 2012).

Iker Burguera et al analyzed behavior of an application as a means for identifying malware. Their dynamic analysis approach is based on crowdsourcing of application data onto a remote server which analyze the system call pattern for each user .The traces are used to distinguish between a benign application and a malware application. Each user who downloaded the CrowDroid application generates behavior related data of each application the use. Then they are clustered using partitioning clustering algorithm with self-written malware and real malware like Streamy Window and Monkey Jump. Crowdroid is capable to distinguish between malware and benign apps of same name and version but having different behavior(Iker Burguera et al,2013).

Hyoung-Woo Lee et al perform relevance pattern analysis of events that occur when an application is launched. Their research provides drawing similarity between system call events of similar type of applications by which we can distinguish malicious applications. They analyzed the system call events with the help of a customized strace tool and a system call event monitoring procedure. This are required to retrieve the activated services and call events from Android Kernel which further transfer all the data in a remote database server. They categorize the malicious system call events and found a mechanism to determine a malicious application which is better than previous CrowDriod implementation (Hyoung-Woo Lee et al ,2014).

M. Karami et al proposed behavioral analysis of applications where black box testing is adopted using Fuzz tool. Authors attempted to automate the user interactions with Android Applications in GUI based interface. They tried to expose the functionality of a target Android Application, by automatically generating which copies the behavior of the user. They used strace tool to view the logs and analyze the file I/O and network I/O event of an application. They found some malware are activated based on events which are further independent of the user interaction and other malwares are activated with user activities (M.Karami et al,2014).

Seonho Choi et al developed an API tracing tool trace view for Android smartphones. The tool can remotely monitor and record API calls from an app running from a mobile device. The trace data can be effective for analyzing the behavior of an application and can even find the system call pattern of the application without the intervention of the user. The working of tool traceview starts with decompilation of an Android Application and further rebuilding it with a new signkey. The traceview output is further converted to CSV file for to enhance portability and other library support. The method of analysis system call is lack on graphical and visualization tool and recognition pattern(Seonho Choi et al,2014).

You Joung Ham et al analyzed system call event patterns of normal benign apps with the most popular game apps in the Android Play Store with malicious system call event patterns distributed by Android MalGenome Project. Authors used strace tool for category based System call event analysis. Their method based analysis addressed four categories like Repackaging, Update Attack, Drive-by-Download and Standalone. They compared the normal and malicious app events with sequence analysis and found consistent pattern of system call function while an app is running.(You Joung Ham et al,2014)

Martina Lindorfer et al presented Andrubis tool for comprehensive analysis system for Android apps. ANDRUBIS can perform static, dynamic and auxiliary analysis on Android applications .Authors collected Android Applications from various markets as well as submission from various users on their online submission page. The static analysis is performed on the decompiled dex file and manifest file while dynamic analysis follows stimulating of Activities, Services, Broadcast receivers etc. Auxiliary analysis includes analysis of network services. The importance of Andrubis lies on the huge sample it has like more than 100000 apps are scanned which includes goodware and malware both. The result analysis opens new direction of data leakage, cross platform malware and activated malware etc(Martina Lindorfer et,2014).

Yu Feng et al provided semantics-based approach for detecting a new types of Android malware. The malware signature matching algorithm of Apposcopy uses a combination of taint analysis and Inter-Component Call Graph for detecting Android applications which contains Inter Process Call flow data. They checked Apposcopy with Android applications from real user and it shows that it can effectively and reliably detect malicious applications that come under certain malware category(Yu Feng et al,2015).

Android Kitkat(version 4.4.2) also incorporated more enhanced version of SELinux Kernel(version 3.10.x or above with API Level 19) where various feature protect the Kernel level by detecting the modification in file system. In this version the MAC implementation also mitigates a large number of exploits in Application Framework Layer. The Application

layer have more secure WebView which uses Chromium Engine (version 30 and above).It also removed OpenSSL middleware feature and third party battery statistics access feature. **Android Lollipop** (version 5.0) released on Nov 12,2014 having more secure Kernel in terms of encrypting user data with improved garbage collection and Ahead of Time(AOT) compilation feature in Android Runtime.

CHAPTER 3
PRESENT WORK

3.1 Problem Formulation:

Currently malware detection mechanism in Android OS is prone to relevancy of the malware from the virus signature exists. In our research we formulated the problem in two parts, first detecting a malware and secondly blocking its system call.

The main problem lies on detecting a unknown malware or distinguishing between good ware and malware .The system call pattern analysis of application is used to determine whether the application is a potential malware or not.

At Linux kernel the system call are invoked by zygote socket .There are 300 system calls are provided in Linux Kernel. These system calls are relevant to the permission assigned to an application. Among the method calls, the application used inside from various packages, some unknown objects are also invoked during runtime. Reflection helps the application package to invoke unknown caller object .the percentage of reflective method called indicates how many unknown objects are invoked by the application. Reflection field access shows how much private field and data are accessed .We have integrated the signature and pattern based approach with reflection method. Here we used apposcopy to determine the reflective method call percentage .Analyzing it with the privacy risk and dangerous permissions inside the application. We tried to draw a pattern for system calls and reflection based method access.

3.2 Objective

There are various malware detection tools and mechanisms are available in market like DroidScope, DroidMat, CrowDroid, Apposcopy and Andrubis. Some of these tools are using semantic based approach, static analysis of manifest file , Clustering based method tracing APIs etc. Individually these tools are effective but lacks in consuming resources and time. They are not effective for naïve users who cannot distinguish a new application installed as potential malware. We have integrated some of these approaches with Java reflection methods where unknown objects are accessed internally. We calculated the RMC and RFA of an individual application. The reflection method call percentage shows the level of privacy leaks an application possesses.

3.3. Research Methodology

An application is downloaded from app market and then the .apk file is decompiled using apktool. The .dex file is generated which is further decompiled using dex2jar tool. The jar file contains the classes, dex file and resource folder. We use jd-GUI tool to check the java code and manifest file to check the permission stated.

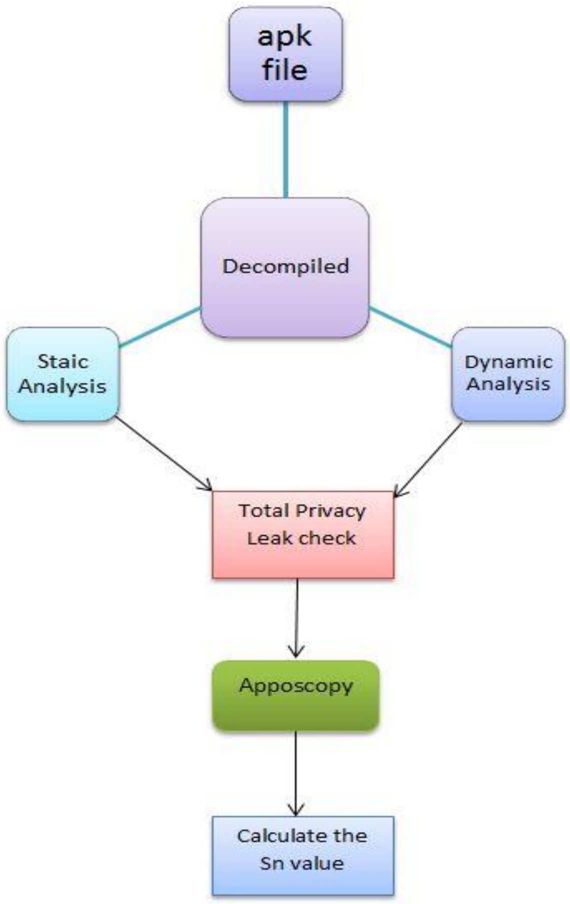


Figure 7. Flowchart of research methodology.

3.3.1. Static Analysis

In static analysis we check the permission associated with an Application. Usually static analysis is done with reverse engineering of an application without executing the application on a device. It is performed using various reverse engineering tool like Apktool, AapInspector, Dex2jar, JAD, jd-GUI,Androguard etc. Androgurad is useful tool to detect malware signature present in an application.

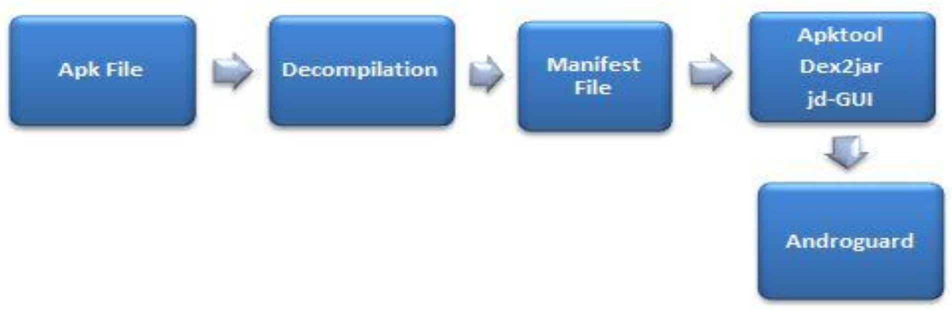


Figure 8. Static Analysis of Android Application.

3.3.2. Dynamic Analysis

In dynamic analysis we decompile the application and check run time behavior by tracing the system calls using tool like Procrank, Strace, BusyBox, Andrubis, Apposcopy etc.Its more powerful and accurate analysis of an unknown application to determine whether an application is malware or goodware.

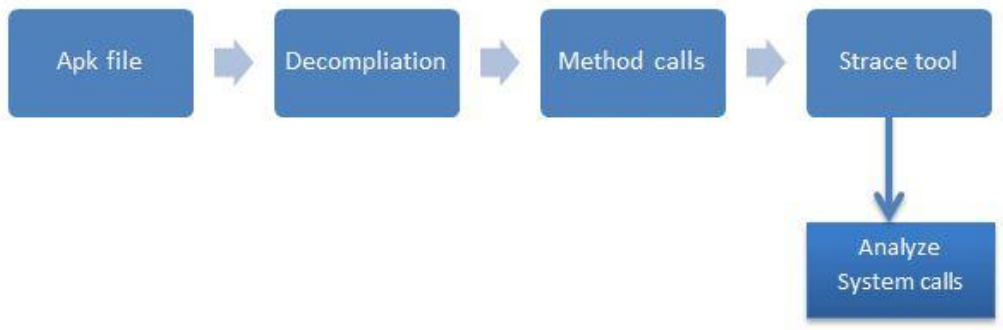


Figure 9. Dynamic Analysis of an Application.

3.4. Tools used

We used various tools for analyzing the applications thoroughly and rigorously. These following tools are used for static and dynamic analysis. These are discussed here.

3.4.1 Android SDK

The Android SDK (software development kit) is a collection of development tools required to develop an Application on Android platform. It has following components:

- Libraries
- Debugger
- An emulator to run application.
- Documentation for using Android Application Programming Interfaces (APIs)
- Guide through Tutorials for the Android OS and sample code.

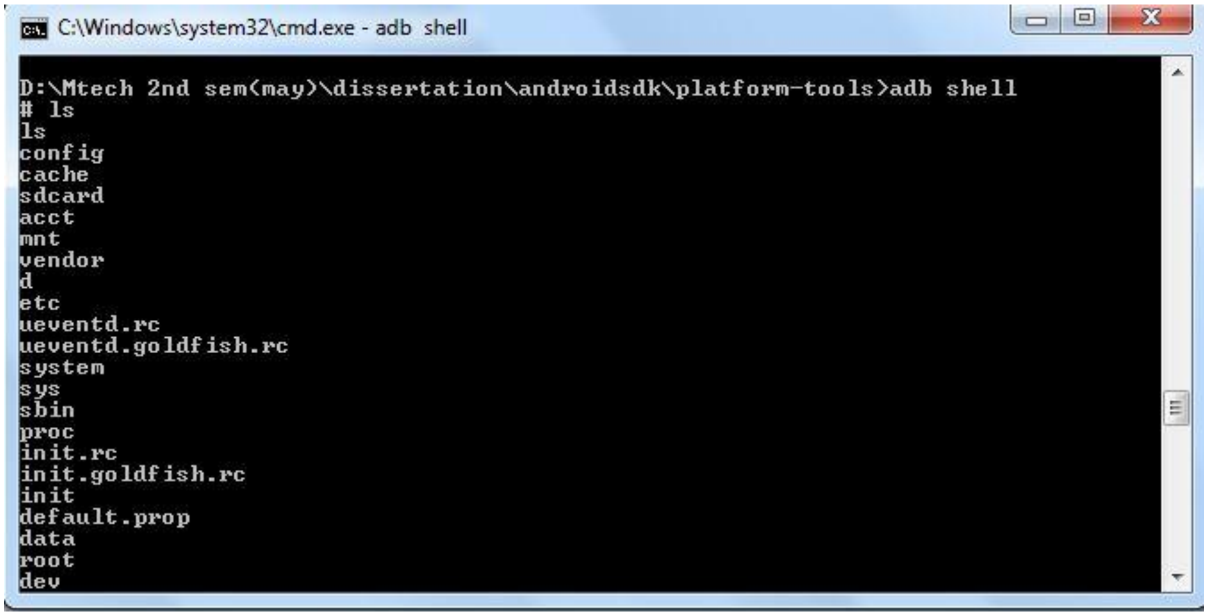
Whenever a new Android version is released by Google,the relevant SDK also released. Developers need to download the version and APIs to build application supporting latest Android version. These APIs can also be downloaded with Eclipse plugin.

The SDK supports all Operating system like Windows (XP and later releases), Mac OS(10.5 and later),Linux (all current distributions).These can be downloaded separately or using any third party software.

The SDK is generally used with a IDE (Integrated Development Environment) for developing an Application. All IDEs available can be integrated with SDK but the Eclipse with ADT plugin is mostly used. Others IDE like IntelliJ and NetBeans are also used.

3.4.2 ADB Shell

The Android Debug Bridge (ADB) is a tool that is included inside Android SDK. It is generally used to access the command line interface of the device attached and it shows the client-server communication. We access the file system, install applications, and check databases using ADB. The ADB is usually accessed with Command Line Interface abut graphical user interface is also available in ADB.

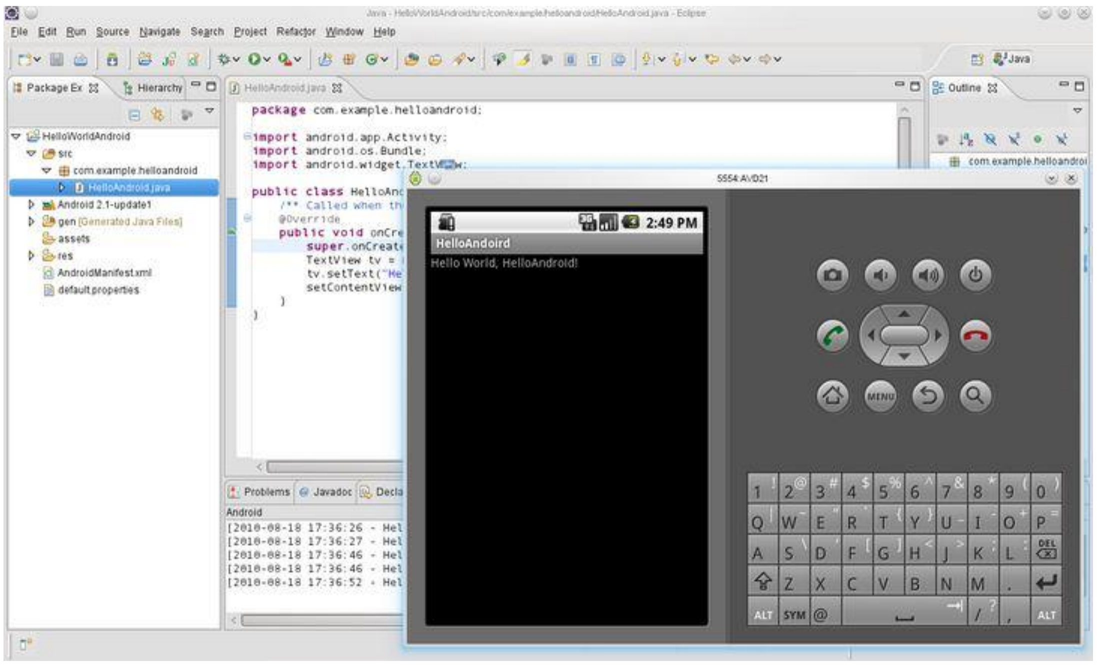

C:\Windows\system32\cmd.exe - adb shell

```
D:\Mtech 2nd sem(may)\dissertation\androidsdk\platform-tools>adb shell
# ls
ls
config
cache
sdcard
acct
mnt
vendor
d
etc
ueventd.rc
ueventd.goldfish.rc
system
sys
sbin
proc
init.rc
init.goldfish.rc
init
default.prop
data
root
dev
```

Figure 10. ADB shell.

3.4.3 Eclipse

Eclipse is an IDE for developing Java based application. It offers customized IDE for developers using various plugins. Android offers a custom Eclipse IDE plug-in called Android Development Tools (ADT) designed to give a powerful, integrated environment in which to build Android applications. It extends the capabilities of Eclipse to let quickly set up new Android projects, create an application UI, debug applications using the Android SDK tools, and even export signed (or unsigned) Android Package (APKs) in order to distribute the application .Eclipse is managed and directed by the Eclipse.org Consortium.



```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAnd
    /** Called when th
    @Override
    public void onCre
        super.onCreate
        TextView tv = 
        tv.setText("He
        setContentView
    }
}
```

```
Android
[2010-08-18 17:36:26 - Hel
[2010-08-18 17:36:27 - Hel
[2010-08-18 17:36:46 - Hel
[2010-08-18 17:36:46 - Hel
[2010-08-18 17:36:52 - Hel
```

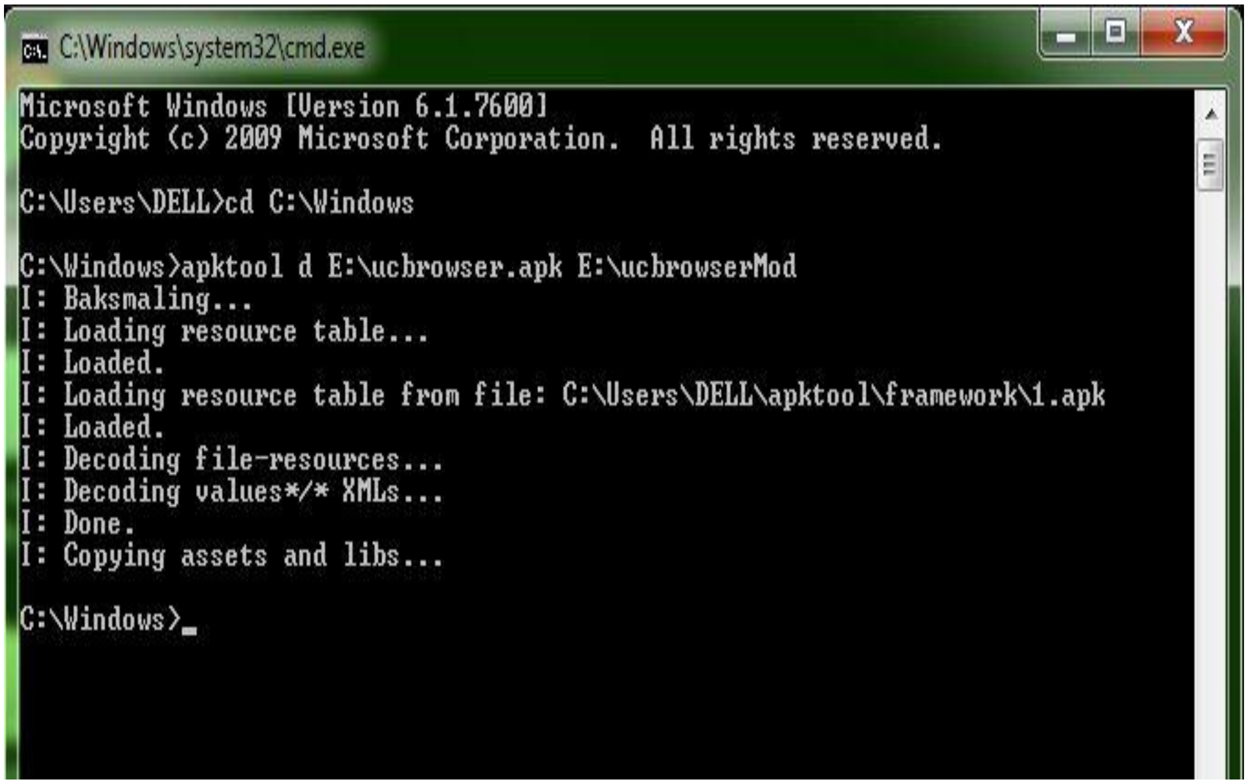
Figure 11. Eclipse IDE.

3.4.4 Strace

Strace is a debugging tool available in Linux. It is used to monitor the IPC (inter process communication) among various running processes while checking their system calls, signal deliveries, and changes of process state. Ptrace runs in the background while running strace command. We generally use strace to list the system call made by a process using process id. Strace is useful tool to get the low level details of a process running.
As strace only details system calls and easier to use than a code debugger. It is an extremely useful tool for the penetration testers and system administrators. It is also used by researchers to generate system call traces and find pattern among them.

3.4.5 Apktool

Apktool is a tool used for performing reverse engineering of an Android Application. It can decode the resources used by an application to the original extend which makes possible to debug the code. It can also repackage the Application after modification. We use apktool for features of displaying project like file-structure, easily repackaging an application etc.



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\DELL>cd C:\Windows

C:\Windows>apktool d E:\ucbrowser.apk E:\ucbrowserMod
I: Baksmaling...
I: Loading resource table...
I: Loaded.
I: Loading resource table from file: C:\Users\DELL\apktool\framework\1.apk
I: Loaded.
I: Decoding file-resources...
I: Decoding values*/* XMLs...
I: Done.
I: Copying assets and libs...

C:\Windows>_
```

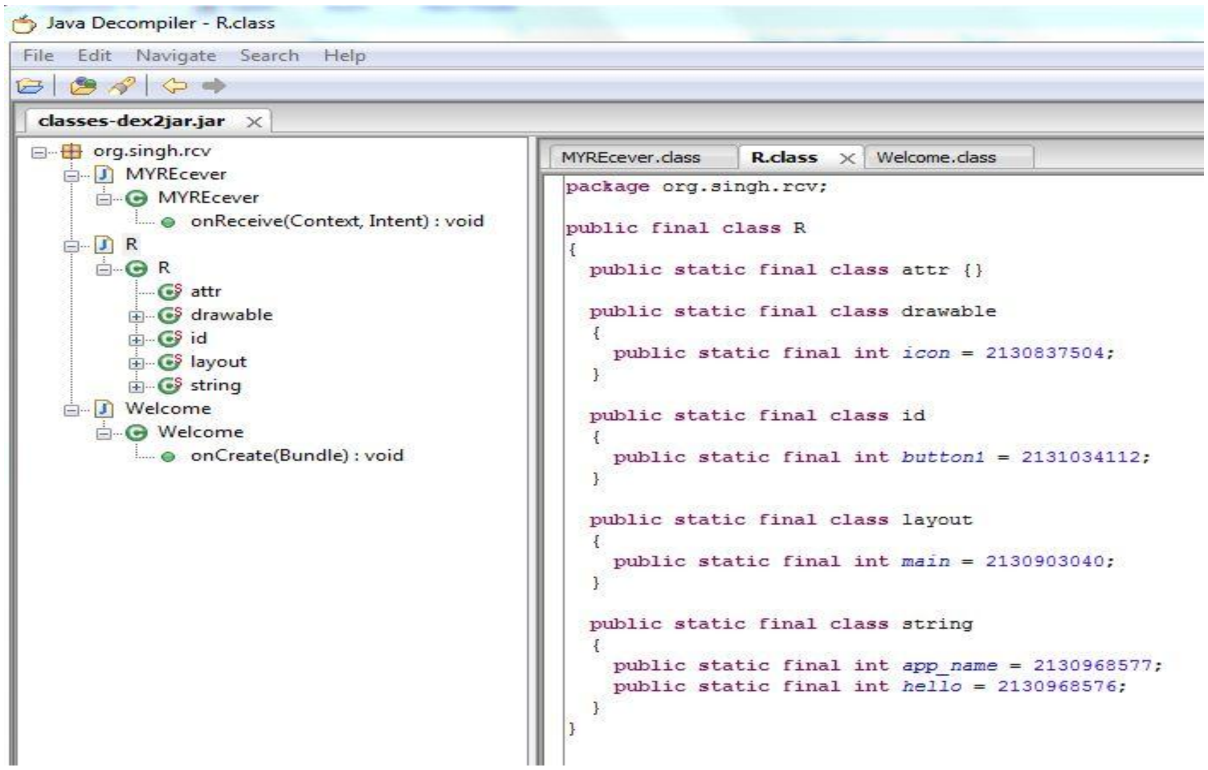
Figure 12. Apktool

3.4.6 Dex2Jar

Dex2jar is a tool for converting one binary file to another binary file like converting the **.**dex file of Android Application to .class file in java. This is used for reverse engineering and extracting source code of an application from apk file.

3.4.7 JD-gui

JD-GUI is a used to displaying the source code of a java class file. One can browse the extracted source code of .class file which is reconstructed. JD-GUI is a freeware for non-commercial use.

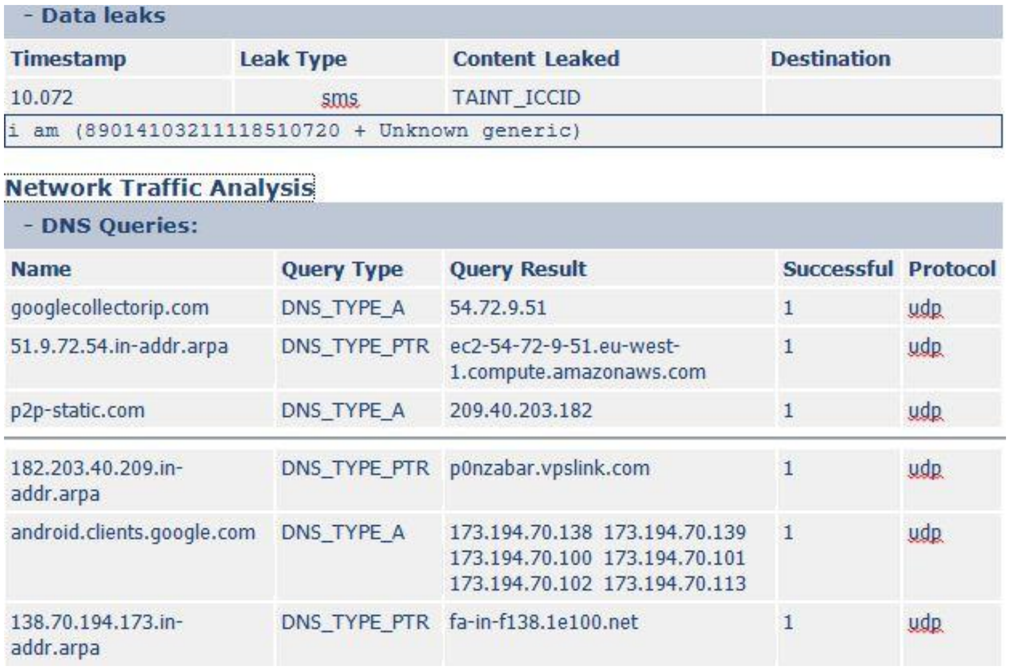


```
package org.singh.rcv;

public final class R
{
  public static final class attr {}

  public static final class drawable
  {
    public static final int icon = 2130837504;
  }

  public static final class id
  {
    public static final int button1 = 2131034112;
  }

  public static final class layout
  {
    public static final int main = 2130903040;
  }

  public static final class string
  {
    public static final int app_name = 2130968577;
    public static final int hello = 2130968576;
  }
}
```

Figure 13. jd-Gui tool

3.4.8. Andrubis

Andrubis is the popular analysis tool for analyzing malware and goodware. It checks the behavioral aspects and properties of an application using stimulated environment. It uses both static and dynamic approach to check an application submitted over Andrubis. It checks the behavior of an application, including file access, network access, dangerous permissions and privacy leaks. In addition to the dynamic analysis in the sandbox, Andrubis also performs static analysis, showing information regarding services, activities, and permission provided.



- Data leaks

Timestamp	Leak Type	Content Leaked	Destination
10.072	sms	TAINT_ICCID	

```
i am (89014103211118510720 + Unknown generic)
```

Network Traffic Analysis

- DNS Queries:

Name	Query Type	Query Result	Successful	Protocol
googlecollectorip.com	DNS_TYPE_A	54.72.9.51	1	udp
51.9.72.54.in-addr.arpa	DNS_TYPE_PTR	ec2-54-72-9-51.eu-west-1.compute.amazonaws.com	1	udp
p2p-static.com	DNS_TYPE_A	209.40.203.182	1	udp
182.203.40.209.in-addr.arpa	DNS_TYPE_PTR	p0nzabar.vpslink.com	1	udp
android.clients.google.com	DNS_TYPE_A	173.194.70.138 173.194.70.139 173.194.70.100 173.194.70.101 173.194.70.102 173.194.70.113	1	udp
138.70.194.173.in-addr.arpa	DNS_TYPE_PTR	fa-in-f138.1e100.net	1	udp

Figure 14. Andrubis Sample Report.

3.4.9 Apposcopy

A semantics-based tool for detecting malwares having privacy leaks. Apposcopy incorporates a model language for specifying signatures and pattern that describe semantic behavior of malware and also performs static analysis for determing whether an application contains a malware signature. It uses taint analysis and ICCG (Inter Component Call Graph) to efficiently detect malware having control- and data-flow properties.

3.5 Reflection

Java Reflection makes it possible to inspect classes, interfaces, methods and fields at runtime, without knowing the names of the classes, methods etc. at compile time. Reflection makes it possible to instantiate new objects; invoke methods and retrieving/setting field values. Java Reflection is useful and can be powerful to inspect classes. For instance, when mapping objects to tables in a database at runtime it is used to check the method access.

Here Method class is obtained from Class object. For example:

```
Method[] methodsarray = MyObj.class.getMethods()
   for(Method method : methodsarray)
   {
   System.out.println(" The  method  name is = " + method.getName());
   }
```

We used to access private fields and methods of other classes via Java Reflection when the objects names are unknown while invoking. This only works when running the code as a standalone Java application as Android Application runs in standalone sandboxing environment .We use Class.getDeclaredField(String name) or Class.getDeclaredFields() method to access the private fields.

CHAPTER 4
RESULT AND DISCUSSIONS

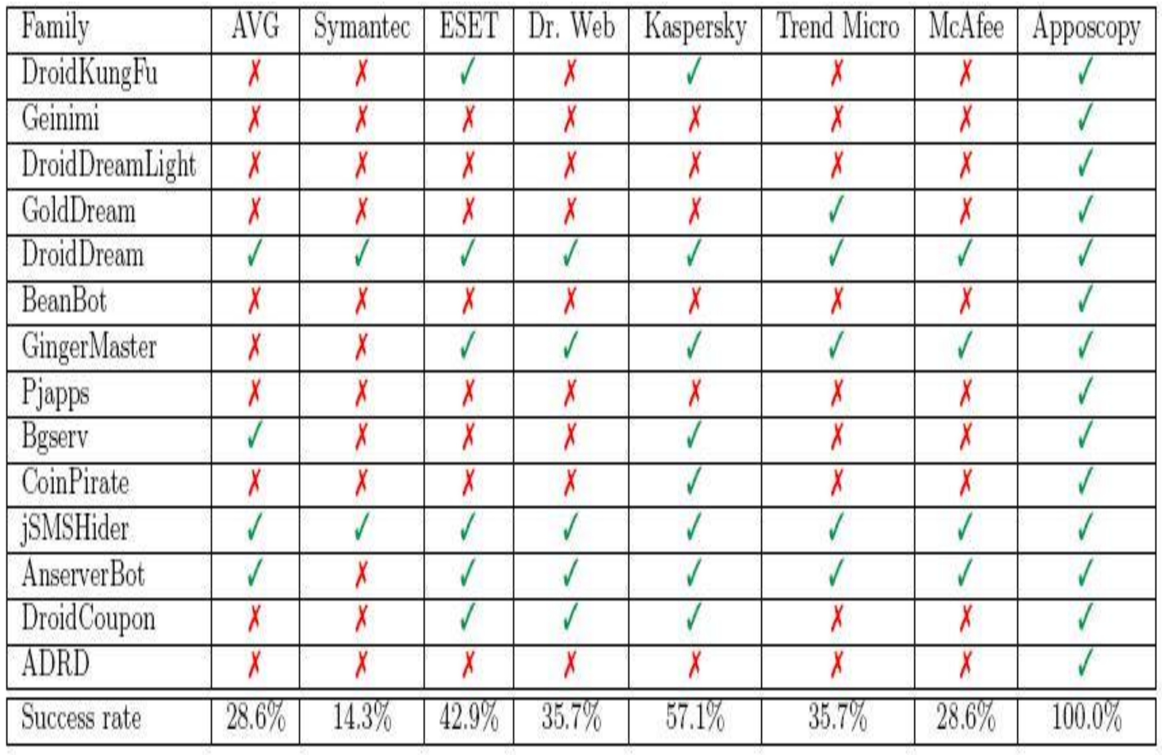
We categorized 100 popular Android Applications in four categories gaming, social, utility and media based application. Then we calculate strace and apposcopy based reflect field access and reflection method calls percentage of total system system calls made by the application .The other parameter we consider are number of dangerous permission and privacy risk exposed by the application .These data are calculated using DroidMat , Apposcopy and other tools. We didn't consider auxiliary analysis other than static and dynamic analysis. We noticed Social Media applications are 50% prone to privacy risk and having more dangerous permissions than other gaming and utility applications. We found a pattern between rmc and privacy leaks .Where Similarity to normal (*Sn*), the lower value indicates its goodware and higher value indicates the malware.

$$Sn = (1 - rmc) * k, k = \sum_{i=0}^{n} dp$$

dp=dangerous permission ,

rmc=reflection method call percentage.

We classified the threat level of these application in 4 tiers where tier 1 shows the lowest level of threat, that signifies the application is having less similarity to a malware signature whereas the tier 4 shows the application is a potential malware. The ranges are decided upon the clustering of *Sn* value of an Android Application. The *Sn* value is calculated with the above stated formula using rmc value and total dangerous permission exists. The reason we depends upon the apposcopy is it's accuracy to detect malware signature than other aviailable techniques and antivirus.



Family	AVG	Symantec	ESET	Dr. Web	Kaspersky	Trend Micro	McAfee	Apposcopy
DroidKungFu	✗	✗	✓	✗	✓	✗	✗	✓
Geinimi	✗	✗	✗	✗	✗	✗	✗	✓
DroidDreamLight	✗	✗	✗	✗	✗	✗	✗	✓
GoldDream	✗	✗	✗	✗	✗	✓	✗	✓
DroidDream	✓	✓	✓	✓	✓	✓	✓	✓
BeanBot	✗	✗	✗	✗	✗	✗	✗	✓
GingerMaster	✗	✗	✓	✓	✓	✓	✓	✓
Pjapps	✗	✗	✗	✗	✗	✗	✗	✓
Bgserv	✓	✗	✗	✗	✓	✗	✗	✓
CoinPirate	✗	✗	✗	✗	✓	✗	✗	✓
jSMSHider	✓	✓	✓	✓	✓	✓	✓	✓
AnserverBot	✓	✗	✓	✓	✓	✓	✓	✓
DroidCoupon	✗	✗	✓	✓	✓	✗	✗	✓
ADRD	✗	✗	✗	✗	✗	✗	✗	✓
Success rate	28.6%	14.3%	42.9%	35.7%	57.1%	35.7%	28.6%	100.0%

Figure 15. Accuracy of Apposcopy

The above table shows that Apposcopy is far better than the currently available anti-virus application to detect a malware signature.

The below figure 16 shows the average dangerous permission we obtained from the sample set of 100 applications of four categories. The social media based application BBM, SnapChat, TextySms shows higher number of dangerous permission while the utility application like widget and wallpaper applications shows lower number of dangerous permissions.

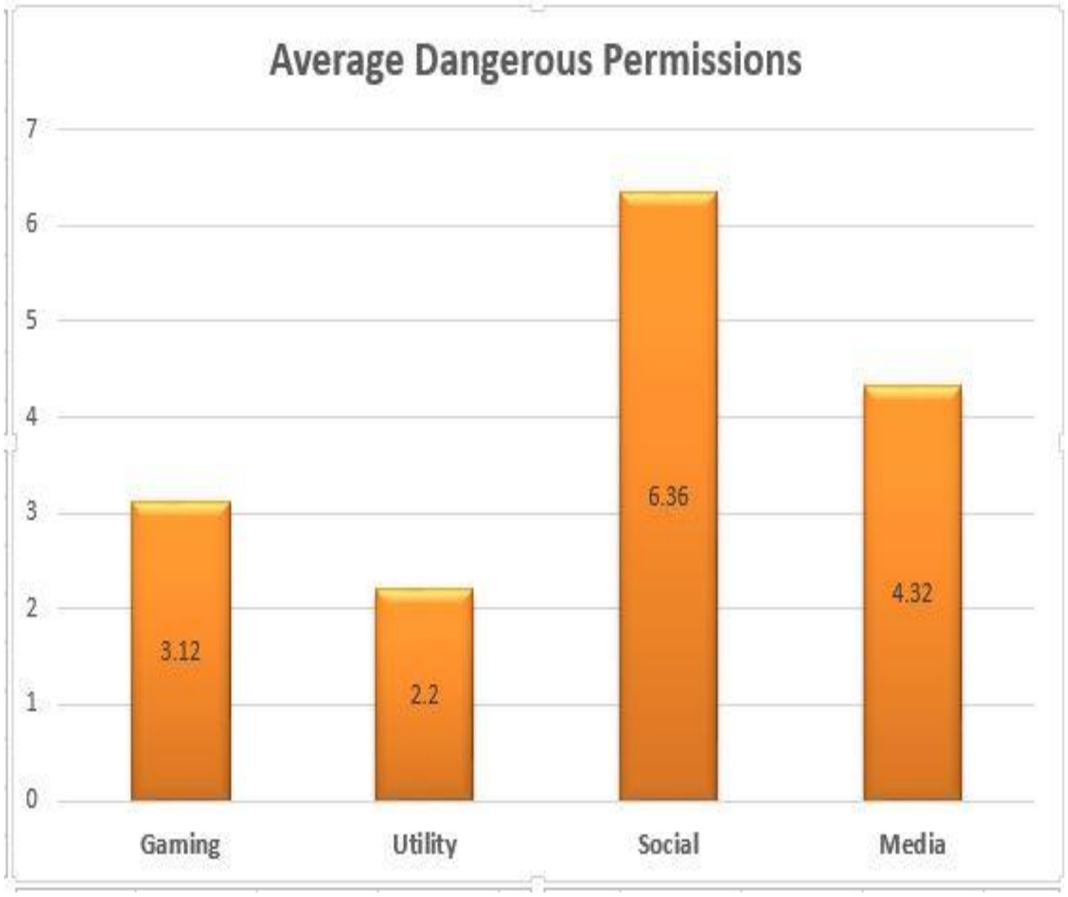


Figure 16. Average Dangerous permissions

The next figure 17 shows the average Privacy leaks inside the applications which we obtaianed from the sample set of 100 applications. Here also he social media based application BBM, SnapChat, TextySms shows higher number of privacy leaks while the utilty application like widget and wallpaper applications shows lower number of privacy leaks. The average privacy leaks of social application are more than 4.

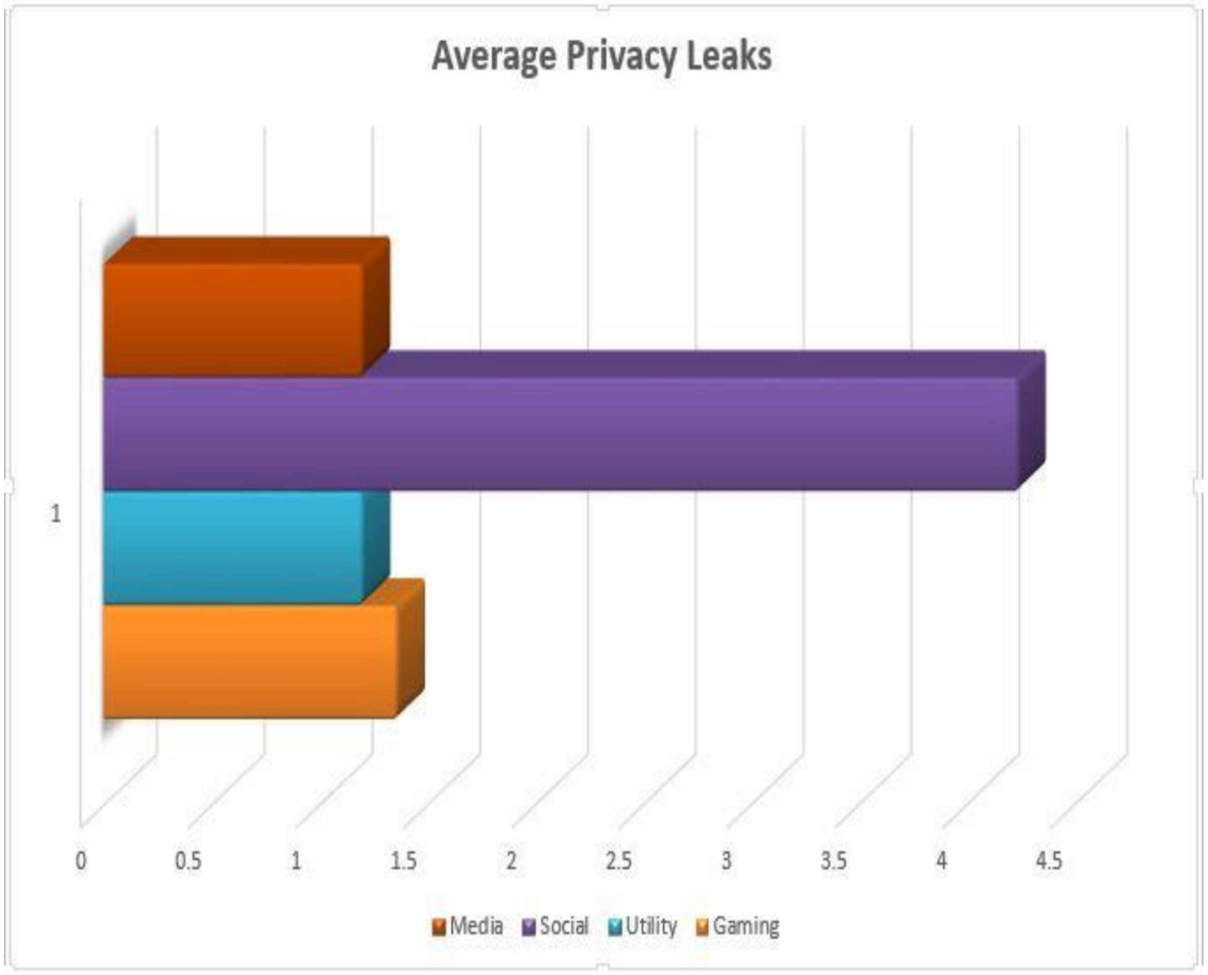


Figure 17. Average Privacy Leaks

Here figure 18 shows the average RFAvalue we obtaioned using apposcopy from the sample set of 100 applications of four categories. The social media based application BBM, SnapChat, TextySms shows higher number of dangerous permission while the media based application like MXvideo Player,Shazam,HBO shows lower number of RFA value.The average RFA value of social applications are 0.0198588 where the media based application having average RFA value 0.00747096.

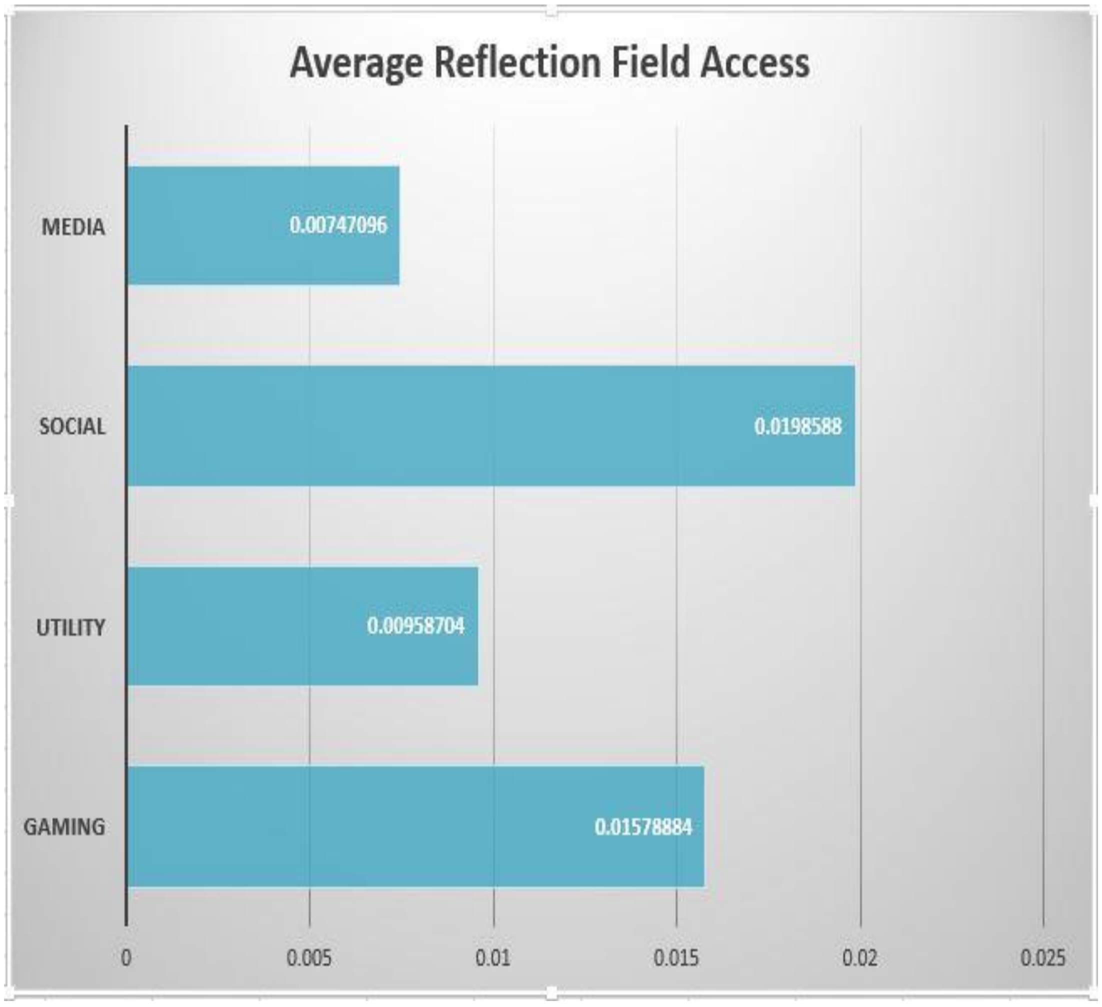


Figure 18. Average RFA value

Next figure 19 shows the average RMC value we obtained using apposcopy from the sample set of 100 applications of four categories. The Gaming applications like Temple Run2,Ninja Run,HeyDay,Bowling 3D shows higher RMC value while the media based application and utilty application shows lower number of RMC value. The average RMC value of Gaming applications are 0.021076 where the utilty based application having average RMC value 0.0121556.

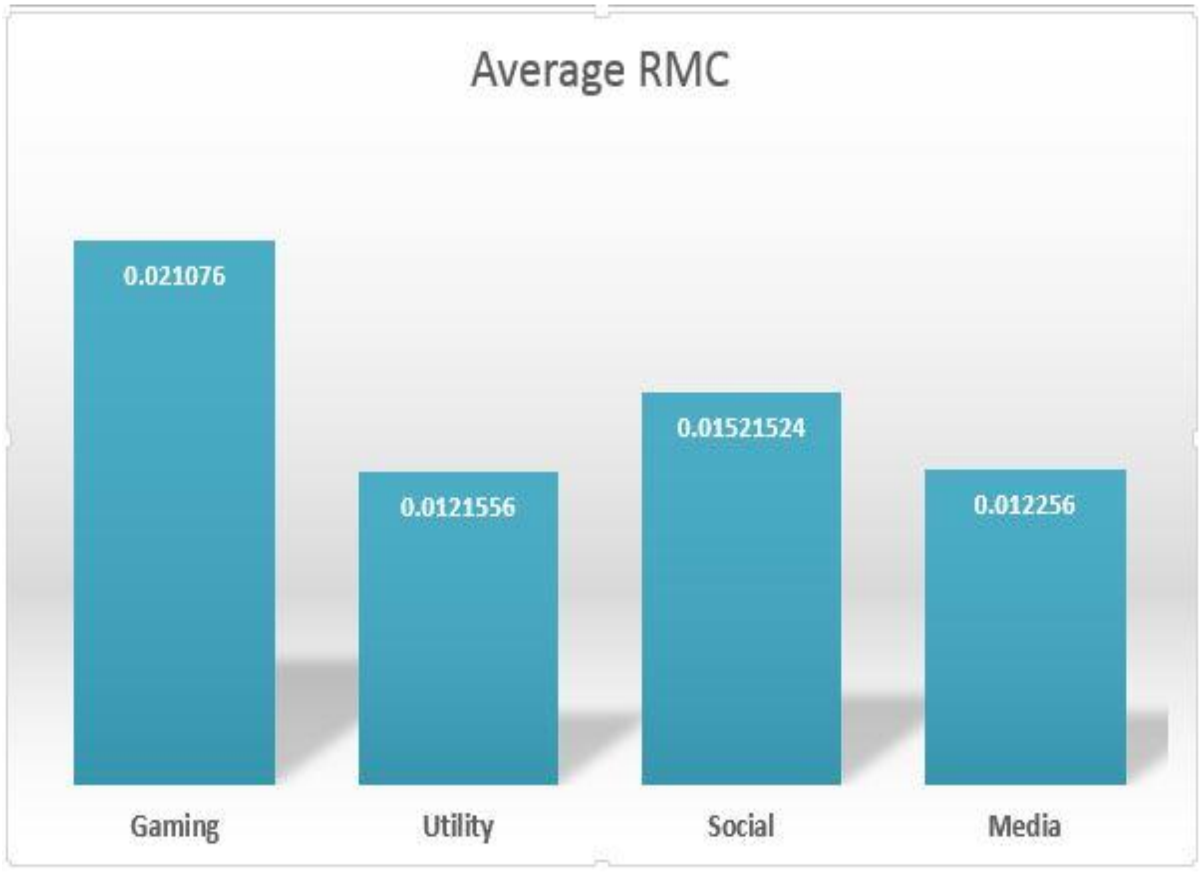


Figure 19. Average RMC value.

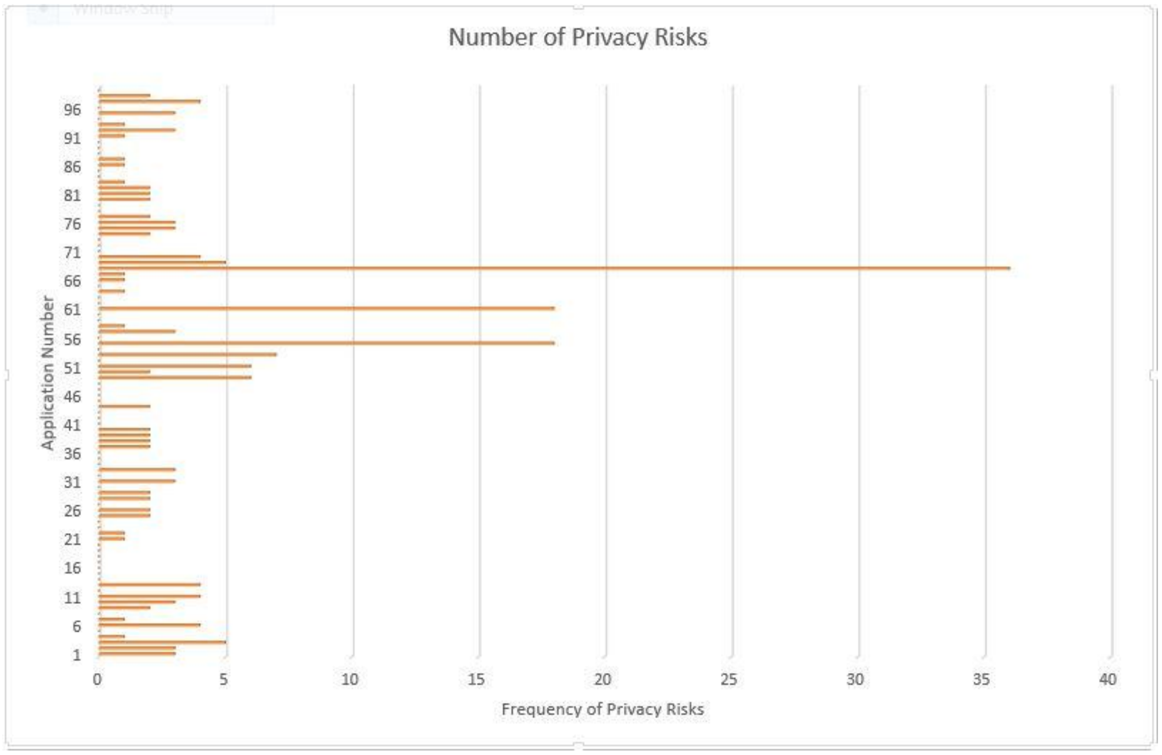


Figure 20. Number of Privacy Leaks

Above figure 20 depicts the number of application risks in each application ranging sample number 1 to 100. It shows most applications having privacy risks below 4

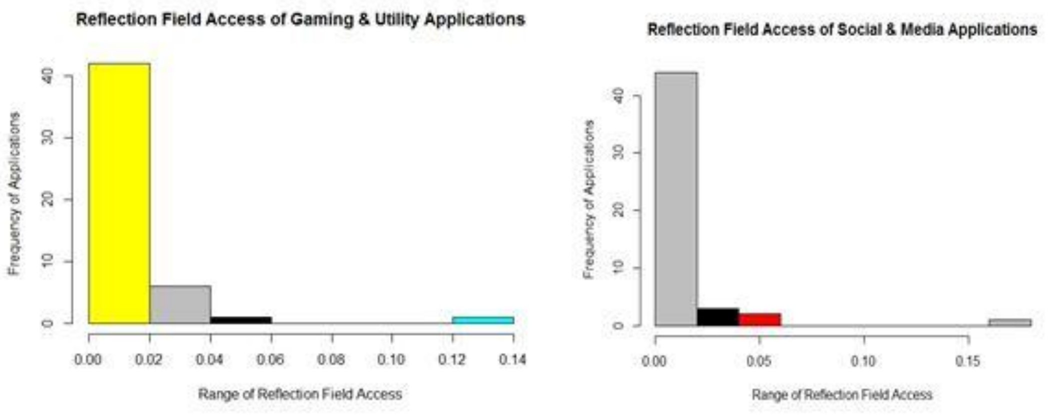


Figure 21. Frequency of Application With RFA values.

The figure 21 shows the frequency with range of RFA values in all four categories of applications. It shows most application having RFA values lies under 0.01 to 0.02 .

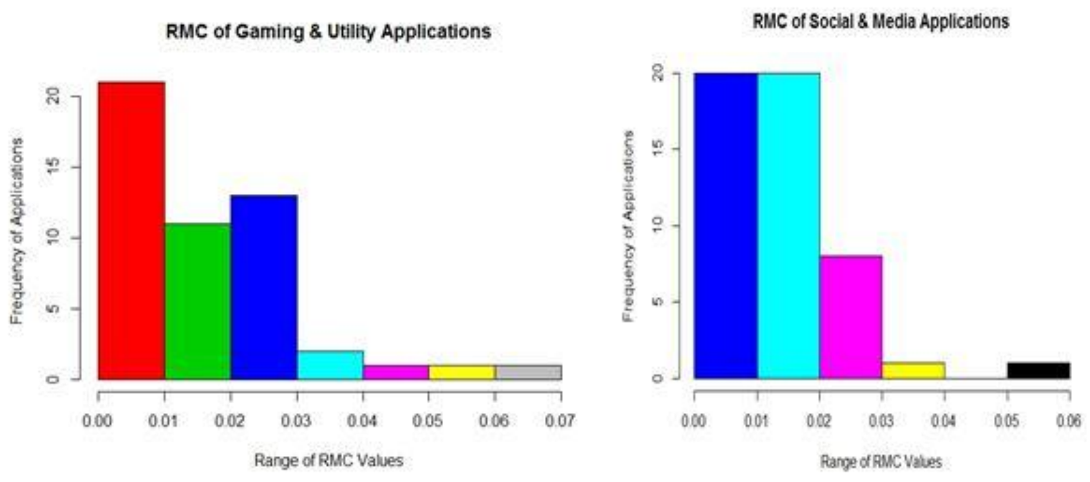
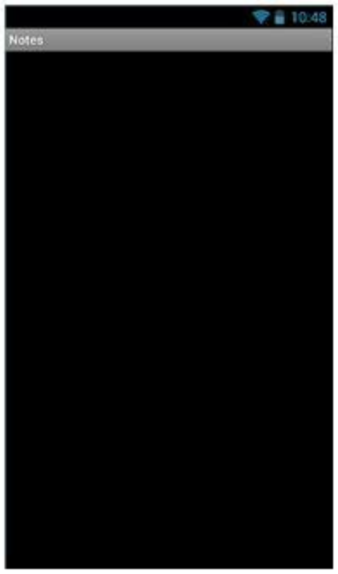


Figure 22. Frequency of Application with RMC values.

The figure 22 shows the frequency with range of RMC values in all four categories of applications. It shows most application having RMC values lies under 0.01 to 0.03.

We created a test application to block system calls of an application in which we injected a malware code segment. The NotePad application is run with NotePadTest application which blocks some system calls in NotePad application.



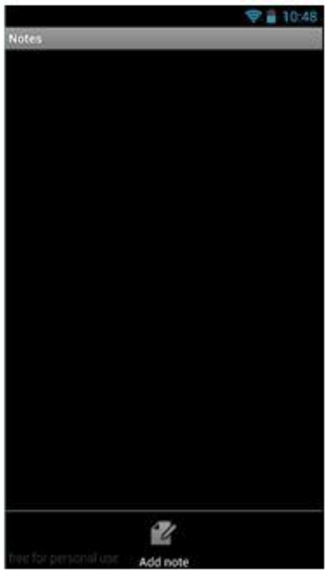
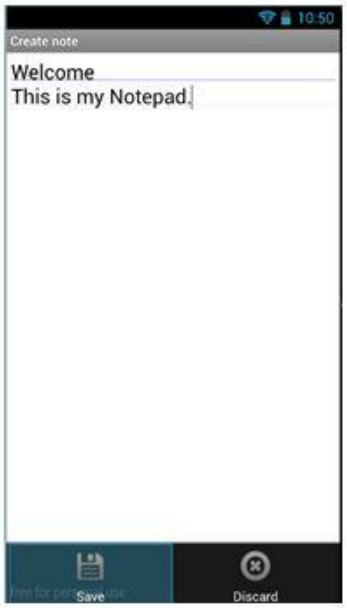
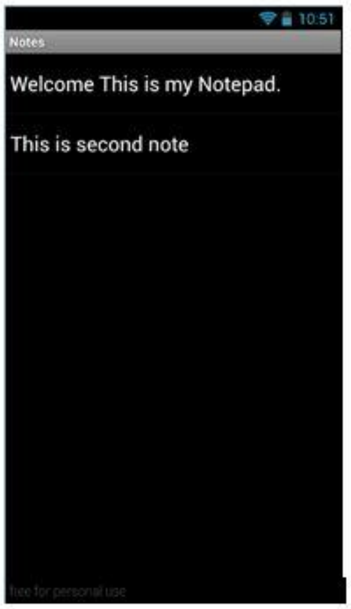


Figure 23. Screenshot 1 and 2 of NotePadTest.



Welcome
This is my Notepad.



Welcome This is my Notepad.

This is second note

Figure 24. Screenshot 3 and 4 of NotePadTest

CHAPTER 5
CONCLUSION AND FUTURE WORK

In this paper, we detailed how we can relate system call patterns with reflection and then find out a potential malware. We introduced RMC(Reflection Method Call) and RFA (Reflection field access) to dynamically the system call pattern of an application .This approach integrate various prevailing mechanism and tools to quick detection of a malware .This reflection based approach shows an application can be easily detected as malware if it shows reflective behavior and invoke reflection methods internally.

The detection of Malicious Android Application by its behavior with reflection based dynamic analysis will reduce the security attacks on Android devices. Users always have tendency to install an Application which is downloaded, overlooking the permission request for that application. Our mechanism will help them to identify the malicious application and blocking them. It may lead further research on this aspect where we may create such mechanism where application are temporary allowed as trail run to analysis their behavior .If found harmful and compromising with user data they might be uninstalled from devices with user consent. This may further incorporated with future releases of Android Operating System.

REFERENCES

A. Armando, A. Merlo, M. Migliardi, and L. Verderame, “Would you mind forking this process? A denial of service attack on Android (and some counter measures)”, *27th IFIP International Information Security and Privacy Conference , IFIP Advances in Information and Communication Technology*, 376, pages 13_24. Springer, 2012.

A. P. Felt, E. Chin, S. HaD. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji,“A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android*”*. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS ’10, 2010.*

A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, “ MockDroid: Trading Privacy for Application Functionality on Smartphones”, In *Proceedings of the 12th International Workshop on Mobile Computing Systems and Applications*, HotMobile ’11.

A. Shabtai and Y. Elovici, “Applying behavioral detection on Android-based devices,” in Mobile Wireless. Middleware, Operating Systems, and Applications. pp. 235-239.Springer, 2010.

Alessandro Armando, Alessio Merlo and Luca Verderame, “An Empirical Evaluation of the Android Security Framework” , Springer, 2012.

D. Arp, M. Spreitzenbarth, M. H ubner, H. Gascon,K. Rieck, and C. Siemens. Drebin: Effective andexplainable detection of android malware in your pocket. 2014.

D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, “A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android*”*. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS ’10.

D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Androidmalware detection through manifest and API calls tracing. In Proc. of Asia Joint Conference on Information Security (Asia JCIS), pages 62–69, 2012.

E. Chin, A. P. Felt, K. Greenwood, and D. Wagner,Analyzing inter-application communication in Android. In MobiSys, pages 239-252, 2011.

Erika Chin,Adrienne Porter Felt, Kate Greenwood ,DavidWagner,AnalyzingInter-Application Communication in Android,ACM,2011.

Himanshu Shewale, Sameer Patil, Vaibhav Deshmukh and Pragya Singh, "Analysis of Android Vulnerabilities and Modern Exploitaion Techniques", *ICTACT*, Volume: 05, Issue: 01, ,March 2014.

I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In Proc. of ACM Worksgop on Security and Privacy in Smartphones and Mobile Devices (SPSM), pages 15–26, 2011.

K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android Permissions". In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.n Specification. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, CCS '12, 2012.

Karami, M., Elsabagh, M., Najafiborazjani, P. and Stavrou, A. (2013) Behavioral Analysis of Android Applications Using Automated Instrumentation. IEEE 7th International Conference on Software Security and Reliability Companion, Washington DC, page 182-187,18-20 June 2013,.

L.-K. Yan and H. Yin. Droidscope: Seamlessly recon-structing os and dalvik semantic views for dynamic an-droid malware analysis. In Proc. of USENIX Security Symposium 2012.

M. Egele, C. Kruegel, E. Kirda, and G. Vigna., “PiOS: Detecting Privacy Leaks in iOS Applications”. In *Proceedings of the 18th Annual Symposium on Network and Distributed System Security*, NDSS ’11, 2011.

M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, “Andrubis - 1,000,000 Apps Later: A View on Current Android Malware Behaviors,” in Proceedings of the the 3rd International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), 2014.

M. Ongtang, K. Butler, and P. McDaniel, “Porscha: policyoriented secure content handling in Android*”*, In 26th Annual Computer Security Applications Conference (ACSAC’10), December 2010.

Ongtang, M., McLaughlin, S., Enck, W., McDaniel, P.: Semantically Rich Application-Centric Security in Android. In: Computer Security Applications Conference, 2009. ACSAC ’09. Annual.(Dec 2009) .

P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “These Aren’t the Droids You’re Looking For: Retrofitting Android to Protect Data from Imperious Applications”, In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS ’11, 2011.

S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.R. Sadeghi, and B. Shastry, “Towards Taming Privilege-Escalation Attacks on Android” , In *Proceedings of the 19th Annual Symposium on Network and Distributed System Security*, NDSS ’12, 2012.

Seonho Choi, Michael Bijou, Kun Sun, and Edward Jung, API Tracing Tool for Android-Based Mobile Devices, International Journal of Information and Education Technology, Vol. 5, No. 6, June 2015.

Stephen Smalley, Chris Vance, and Wayne Salamon, "*Implementing SELinux as a Linux Security Module*", *NAI Labs Report #01-043*.

Stephen Smalley, Robert Craig, "Security Enhanced (SE) Android: Bringing Flexible MAC to Android", Trusted Systems Research, *National Security Agency*.

W. Enck, M. Ongtang, and P. McDaniel, "On lightweight mobile phone Application certification" , In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, 2009.

W. Enck, P. Gilbert, B.G. Chun, L. P. Cox, J. Jung, P. McDaniel, A. N. Sheth, "TaintDroid: an information-ow tracking system for realtime privacy monitoring on smartphones". In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, 2010.*

Xiali Hei, Xiaojiang Du and Shan Lin, " Two Vulnerabilities in Android OS Kernel" Department of Computer and Information Sciences,Temple University.

Y Feng, S Anand, I Dillig, A Aiken. Apposcopy: Semantics-based detection of Androidmalware through static analysis In SIGSOFT FSE,2014.

Y. J Ham and H. W. Lee, "Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events," International Journal of Computer and Communication Engineering , vol. 3, no. 2, pp. 149-154, 2014.

Y. Tang, P. Ames, S. Bhamidipati, A. Bijlani, R. Geambasu, and N. Sarda. " CleanOS: Limiting Mobile Data Exposure With Idle Eviction", In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, USENIX OSDI '12.

Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, “Taming Information-Stealing Smartphone Applications (on Android)”, In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, TRUST ’11, 2011 .

Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you,get off my market: Detecting malicious apps in offficial and alternative android markets. In NDSS,2012.

You Joung Ham, Daeyeol Moon1, Hyung-Woo Lee, Jae Deok Lim and Jeong Nyeo Kim. "Android Mobile Application System Call Event Pattern Analysis for Determination of Malicious Attack", International Journal of Security and Its Applications, vol. 8, no. 1, pp. 231-246, 2014.

WEBSITES

- Gartner Says Worldwide Traditional PC, Tablet, Ultra mobile and Mobile Phone Shipments On Pace to Grow 7.6 Percent in 2014.
  Available at: http://www.gartner.com/newsroom/id/2645115.
- Enabling the Kernel's DMESG_RESTRICT feature,
  Available at: https://lists.ubuntu.com/archives/ubuntu-devel/2011-May/033240.html
- Arxan's Annual Report: '*State of Mobile App Security'* Reveals an Increase in App Hacks for Top 100 Mobile Apps, November 17, 2014,
  Available at: https://www.arxan.com/arxan-annual-report-state-of-mobile-app-security-reveals-an-increase-in-app-hacks-for-top-100-mobile-apps/?CategoryId=5.
- Andrubis: A tool for analyzing unknown Android Applications.
  Available at: http://anubis.iseclab.org/.

APPENDIX

LIST OF ABBREVIATIONS

OHA: Open Handset Alliance

OS: Operating System

ID: Identifier

API: Application Programming Interface

DVM: Dalvik Virtual Machine

UI: User Interface

SDK: Software Development Toolkit

GPS: Global Positioning System

SD: Secure Digital

HTTP: Hyper Text Transfer Protocol

IPC: Inter-process Communication

UID: User Identifier

URL: Uniform Resource Locator

JD-GUI: Java Decompiler – Graphical User Interface

SSL: Secured Socket Layer

ADB: Android Debug Bridge

IDE: Integrated Development Environment

ADT: Android Development Tool

JDK: Java Development Tool

APK: Android Package

RMC: Reflection Method Call

RFA: Reflection Field Access