Constraints on Permission for Making Safe App Execution

Submitted in partial fulfillment of the requirements of the degree of

Master of Technology (M.Tech)

by

Dharmendra Kumar

***Roll no.* 163050032**

Supervisor:

Prof. R.K. Shyamasundar

Department of Computer Science & Engineering

Indian Institute of Technology Bombay

2018

Dissertation Approval

This project report entitled **"Constraints on Permission for Making Safe App Execution"**, submitted by **Dharmendra Kumar** (Roll No. **163050032**), is approved for the award of degree of **Master of Technology (M.Tech)** in Computer Science & Engineering.

RKShyamasundar

Prof. R.K. Shyamasundar
Dept. of CSE, IIT Bombay
Supervisor

[illegible]

Prof. G. Sivakumar
Dept. of CSE, IIT Bombay
Internal & External Examiner

[illegible]

Prof. Bernard Menezes
Dept. of CSE, IIT Bombay
Internal Examiner

Date: 29 June 2018

Place: IIT BOMBAY, MUMBAI

Declaration of Authorship

Signature: Dharmendra Kr

Dharmendra Kumar

163050032

Date: 30 June 2018

Abstract

In recent past smart-phones have become very popular due to their utility in obtaining online services and increasing level of user convenience. The cost of smart-phones is falling and increasing number of laymen are using such devices installed with utility applications. Android operating system runs on nearly 82% of these smart-phones available today. As in the case of desktop operating systems, mobile operating systems too are prone to viruses and malware and there is a steep increase in malicious Apps on Android's application store from where the users install their applications. There exists techniques and methods that are developed to check whether an application is benign or malicious. Applications are checked before they are uploaded to the applications store. The statistics shows that the existing approaches are falling short. Most of the works classify applications into categories based on certain models. Accuracy of these models has direct impact on the security of the user data these applications have access to on a smart-phone. Sometimes malicious applications are also classified as benign. In such cases, there can be serious security breaches as the application makes its way to users' smart-phones via application store. The main idea of our approach is to restrict the malicious behaviour of applications to provide the security to user's private data and device itself. We have done dynamic and static analysis of applications to know the behaviour of applications. We have build classification models using various machine learning techniques to classify the applications as benign or malicious. If applications are classified as malicious, we have restricted its malicious behaviour without modifying apk file. We have used wrappper around the applications such that applications work normally even after restrictions.

Contents

List of Figures

Chapter 1

Introduction

Android mobile devices is becoming very popular due to ease of use and its user-friendliness User Interface (UI). There is monopoly of Android in mobile device market. Nearly 82% of smartphones run Android Operating System [15] . Due to such a large market share, malware authors are targeting Android mobile devices. The Android is the fastest growing mobile operating system. Unlike Apple's App Store, deploying an application in Google Play Store is much easier. Google takes minimal steps for inspection of applications this allows anyone to publish applications on the Android market. If an application is reported as malware by users, it will then be removed. Due to large market share and easy deployment policy, malware authors are targetting Android. Malware authors are creating malicious application to compromise Android security. According to AO Kaspersky Lab, following Android security issues were in trend in 2016:

- Third party applications using super user permissions.
- Development of new mechanism to pass security system.
- Ransomware i.e encrypting mobile devices.
- Continuous improvement in Mobile Banking Trojans.
- Deploying malicious application in Google Play Store.

In 2016, Kaspesky Lab detected the following:

- 8,534,221 malicious installation packages

- 136,786 mobile banking Trojans
- 251,114 mobile ransomware Trojans

Malicious applications frequently easily passes the security suite of Google's Play Protect security suite, and some applications attracts millions of downloads before Google can find and remove them. Recently the security firm Check Point discovered a new type of Android malware called "Expensive Wall" hidden in 50 apps in the Play Store. They have been cumulatively downloaded nearly 4.2 million times. Even after Google removed all these applications, Check Point discovered new sample of the malware in the Google Play store. So, we can not only rely on Google Play Protected security suite.

Antivirus software is designed to detect, prevent and take action against malicious software. It can either disarm or remove the malicious software. Antivirus scans the file comparing specific pattern in the code against the signature of viruses which already stored in database. If pattern matches, then it is considered as malicious. If a new type of malicious application will come, then it will not be identified by the current antivirus. In that case, from security perspective, application may access user's private data and perform security-sensitive operations on the device.

In the current scenario, Android security revolves around the permissions. Applications ask for all the permission that it needs. If required permission is not given to the application, applications does not work properly. Since, applications need all the permission to work we can not restrict particular feature of applications that is causing malicious activity.

For providing the security to user's private data and device, our approach is to restrict the behaviour change of Android application and also to restrict them from doing any malicious activity. To achieve our goal we are doing the following things:

- We are doing the static and dynamic analysis of the application to gather information about permissions and Application Program Interfaces (APIs). Based on permissions and APIs, we are trying to figure out what can be the possible behaviour of application.

- We are building App Classification Model to check whether application is benign or malicious.
- We try to restrict the malicious behaviour of application by wrapping the application.

Organisation: In the following chapter, that is Chapter 2, we provide background and related work. In Chapter 3, we provide details about the Android software stack architecture and describe only the components of the stack that are relevant to our work. In Chapter 4, we present our approach and also provide analysis of some prominent applications on application store of Android. In Chapter 5, we present our classification models that help us determine/classify whether an app is benign or malicious. In Chapter 6, we show how to restrict malicious intent of an application and also talks about case studies of various applications and we conclude with possible future work in Chapter 8.

Chapter 2

Background & Related Work

Android operating system has five layer in its software stack. There are multiple applications running on the top of operating system. To provide the security to one application from other applications and to Android operating system from applications Android uses uses virtual machines. Every Android application runs in its own process, with its own instance of the Dalvik virtual machine (DVM) [18] / Android Runtime (ART) [5]. Since, all Android applications run in its own virtural machine, they can not affect the execution of each other and also they can affect Android operating system functionality.

To check the behaviour of Android applications various technique and tools have been developed. Androwarn is a tool to analyze the applications [4]. Androwarn does the static analysis of the application's Dalvik bytecode, represented as Smali and provides the detail about possible malicious behaviour of applications. Flowdroid uses call graphs of the application to analyze the behaviour of the application [1]. Static analysis of application can be done on the basis of permissions it requires [11]. Another way of static anaysis is to check the behaviour of kernel to provide the behaviour of applications [9].

Droidmate [10] does the dynamic analysis of application by clicking on various UI element and traces the API calls made. Crowdroid [2] detects the malicious behaviour by analyzing data collected from two sets: one from those from articial malware created for test purposes, and those from real malware found in the wild. To detect malware in Android applications various methods and techniques have developed. Some of those approaches are:

Permission based approach

To stop malware installation permission based system has been developed. While installation applications ask for permission to access sensitive resources. Based on requested permission a classifier has been developed to identify benign and malicious application.

Two-layered permission based approach

In this method, they analyzed data from two types of permissions namely *requested permissions* and *used permissions.* After analyzing requested permissions, they make classifier to classify applications int benign and malware. After this classification, they make another classifier by analyzing used permissions to identify malware.

Malware detection using static feature-based analysis

This approach uses various static information such as permissions, deployment of components, intent message passing and API calls for characterizing the Android application behaviour. After extracting the above information one can

1. applies K-means algorithm to classify applications into benign and malware.

2. can generate a model using KNN classifier for detecting malware.

Malware detection using hybrid of static and dynamic analysis

By static analysis, it extracts expected activity switch paths by analyzing both Activity and Function Call Graphs. Then it uses dynamic analysis to traverse each UI elements and explore the UI interactions paths towards sensitive APIs. Using these data, they create a model to detect malware.

Behaviour-based malware detection system

They firstly gather test data and actual processed data, it processes the data and try to figure out some patterns in the data that can be used to differentiate benign applications and malicious applications. They use different machine learning techniques to find the pattern in the data. After finding the pattern, they make classifier to identify the malware.

By analyzing usage of sensitive data

In this method, they firstly analyze the flow of sensitive data in benign applications and then try same for malicious applications. From these data, they implements different type of classifier to detect malicious applications.

Chapter 3

Android: Architecture, Permissions, APIs

Android is widely used operating system for moblie devices. In section 3.1, we will discuss about the architecture of android. In section 3.2, we will discuss permissions required by any application to access the sensitive resources. In Section 3.3, we will give brief idea about APIs.

3.1 Architecture

Android is opensource. Android operating system is based on Linux based-software stack [8]. There are mainly five components in Android software stack. These are:

1. Linux Kernel
2. Native Libraries
3. Android Runtime
4. Application Framework
5. Applications

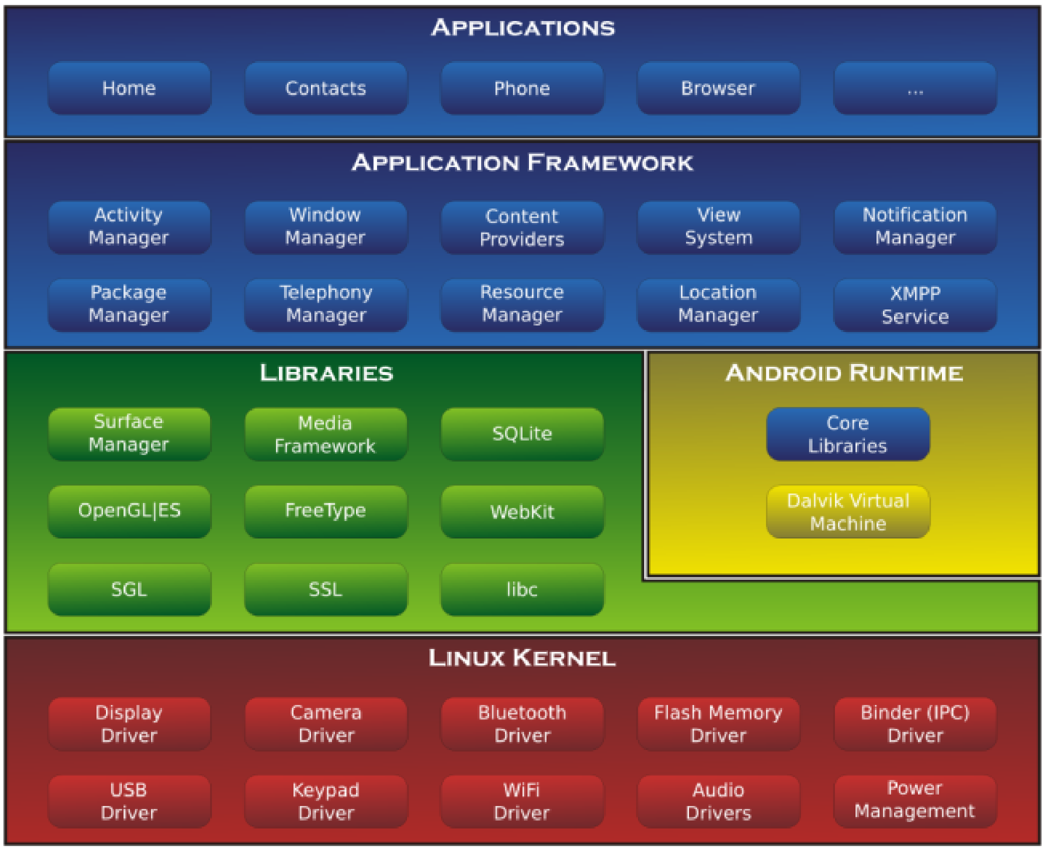


Figure 3.1: The Android Software Stack

3.1.1 Linux Kernel

Linux kernel is root/foundation of the Android platform. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. Linux kernel includes display driver, camera driver, bluetooth driver, Flash memory driver, Binder (IPC) driver, USB driver, kepad driver, WiFi driver and Audio Driver.

3.1.2 Native Libraries

Native libraries sits on the top of Linux kernel. Some of native libraries are WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc), etc. The WebKit is used for browser support, SQLite is used for database, FreeType is used for font support, Media is used for playing and recording audio and video formats.

3.1.3 Android Runtime

Android Runtime contains some core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is similar to JVM but it is more optimized for mobile devices. It comsumes less memory and provides fast performance. ART is written to run multiple virtual machines on low-memory devices. Some of the major features of ART includes the following:

- Ahead-of-time (AOT) and just-in-time (JIT) Compilation.
- Optimized garbage collection (GC).
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields.

3.1.4 Android Framework

Android Framework sits on the top of Native libraries and Android runtime. Android framework contains Content provider, View system, Activity manager, Window manager, Telephony manager, resource manager, location manager and XMPP service.

3.1.5 Applications

Last layer of the Android Platform Architecture is applications. It contains applications. All applications such as home, contact, settings, email, dialer, calendar, camera, etc. are using Android framework that uses Native libraries and Android runtime. Android runtime and native libraries are using linear kernel.

3.2 Permissions

If an Android application wants to use any resources such as camera, internet, location, etc., they need permission from the Android operating system to use them. Android marshmallow operating system has more than 200 permissions [7]. The main purpose of permission is to protect the user data. Android apps must request permission to access sensitive user data as well as the system features. Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request. Permissions are divided into several protection level. There are three protection levels that affect third-party apps:

1. Normal permissions
2. Signature permissions
3. Dangerous permissions

3.2.1 Normal Permissions

Normal permissions are used where an needs to access data or resources outside the app's sandbox but where there is very little risk to the user's privacy or the operation of other apps. According to Android 8.1, following are the list of normal permissions:

- `ACCESS_LOCATION_EXTRA_COMMANDS`
- `ACCESS_NETWORK_STATE`
- `ACCESS_NOTIFICATION_POLICY`
- `ACCESS_WIFI_STATE`
- `BLUETOOTH`
- `BLUETOOTH_ADMIN`
- `BROADCAST_STICKY`
- `CHANGE_NETWORK_STATE`
- `CHANGE_WIFI_MULTICAST_STATE`
- `CHANGE_WIFI_STATE`
- `DISABLE_KEYGUARD`
- `EXPAND_STATUS_BAR`
- `GET_PACKAGE_SIZE`
- `INSTALL_SHORTCUT`
- `INTERNET`

- KILL_BACKGROUND_PROCESSES
- MANAGE_OWN_CALLS
- MODIFY_AUDIO_SETTINGS
- NFC
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- RECEIVE_BOOT_COMPLETED
- REORDER_TASKS
- REQUEST_COMPANION_RUN_IN_BACKGROUND
- REQUEST_COMPANION_USE_DATA_IN_BACKGROUND
- REQUEST_DELETE_PACKAGES
- REQUEST_IGNORE_BATTERY_OPTIMIZATIONS
- SET_ALARM
- SET_WALLPAPER
- SET_WALLPAPER_HINTS
- TRANSMIT_IR
- USE_FINGERPRINT
- VIBRATE
- WAKE_LOCK
- WRITE_SYNC_SETTINGS

3.2.2 Signature Permissions

The system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission. According to Android 8.1, following are the list of signature permissions:

- BIND_ACCESSIBILITY_SERVICE
- BIND_AUTOFILL_SERVICE
- BIND_CARRIER_SERVICES
- BIND_CHOOSER_TARGET_SERVICE

- `BIND_CONDITION_PROVIDER_SERVICE`
- `BIND_DEVICE_ADMIN`
- `BIND_DREAM_SERVICE`
- `BIND_INCALL_SERVICE`
- `BIND_INPUT_METHOD`
- `BIND_MIDI_DEVICE_SERVICE`
- `BIND_NFC_SERVICE`
- `BIND_NOTIFICATION_LISTENER_SERVICE`
- `BIND_PRINT_SERVICE`
- `BIND_SCREENING_SERVICE`
- `BIND_TELECOM_CONNECTION_SERVICE`
- `BIND_TEXT_SERVICE`
- `BIND_TV_INPUT`
- `BIND_VISUAL_VOICEMAIL_SERVICE`
- `BIND_VOICE_INTERACTION`
- `BIND_VPN_SERVICE`
- `BIND_VR_LISTENER_SERVICE`
- `BIND_WALLPAPER`
- `CLEAR_APP_CACHE`
- `MANAGE_DOCUMENTS`
- `READ_VOICEMAIL`
- `REQUEST_INSTALL_PACKAGES`
- `SYSTEM_ALERT_WINDOW`
- `WRITE_SETTINGS`
- `WRITE_VOICEMAIL`

3.2.3 Dangerous Permissions

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, your app cannot provide functionality that depends on that permission. According to Android 8.1, following are the list of dangerous permissions:

- `READ_CALENDAR`
- `WRITE_CALENDAR`
- `CAMERA`
- `READ_CONTACTS`
- `WRITE_CONTACTS`
- `GET_ACCOUNTS`
- `ACCESS_FINE_LOCATION`
- `ACCESS_COARSE_LOCATION`
- `RECORD_AUDIO`
- `READ_PHONE_STATE`
- `READ_PHONE_NUMBERS`
- `CALL_PHONE`
- `ANSWER_PHONE_CALLS`
- `READ_CALL_LOG`
- `WRITE_CALL_LOG`
- `ADD_VOICEMAIL`
- `USE_SIP`
- `PROCESS_OUTGOING_CALLS`
- `BODY_SENSORS`
- `SEND_SMS`
- `RECEIVE_SMS`
- `READ_SMS`

- `RECEIVE_WAP_PUSH`
- `RECEIVE_MMS`
- `READ_EXTERNAL_STORAGE`
- `WRITE_EXTERNAL_STORAGE`

3.3 Application Program Interfaces (APIs)

The Android API refers to the collection of various software modules which make up the complete Android SDK. If we have to use any functionality of android, then we have to use API corresponding to that functionality.

Chapter 4

Method of App Analysis

In Google Play Store, there are lots of malicious applications. So, we have tried to find the behaviour of application and restrict its malicious intent if any. In section 4.1, we have discussed our approach. In section 4.2, we have discussed about analysis of some applications.

4.1 Design Approach

There are nearly 3.6 million applications on Google play store [14]. Not all the applications are benign on the Google play store. Some applications may have security issues. Android also supports installation of application from third and third party application can not be trusted. So, we have proposed a mechanism to determine the behaviour of application. If we find behaviour of application as malicious, then we have to restrict the malicious behaviour of applications. We are doing the whole process in following three phases:

1. Analysis of application.

2. Determining behaviour of application.

3. Restricting malicious intent of application.

We are doing static and dynamic analysis of applications. For analyzing the application we are using Droidmate [10], RiskIndroid [12] and AndroPyTool [6]. From the analysis we are extracting the permissions and API calls from the applications. After seeing the

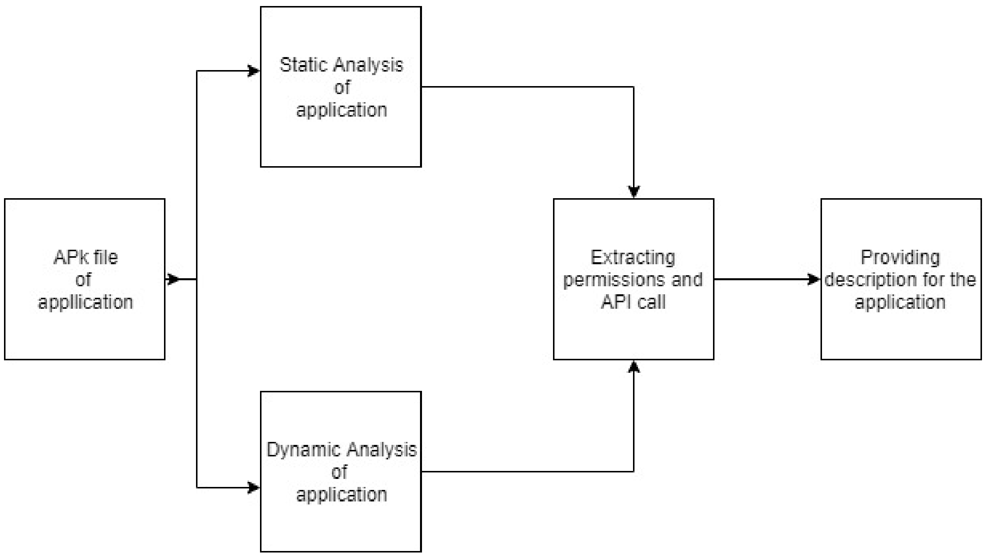

Static Analysis of application
APk file of appllication
Dynamic Analysis of application
Extracting permissions and API call
Providing description for the application

Figure 4.1: Workflow of analysis phase

permissions required by applications and API calls it is invoking, we are proposing the description of the application. Workflow for analysis phase is shown in Figure 4.1.

Using the analyzed data of applications whose class is known (it is known that whether application is benign or malicious), we are developing the classification model with the help of various machine learning techniques such as Logistic Regression, Support Vector Machine, Neural Network and Random Forest. Once the model is built, we are using that classification model to determine the class of applications that we have analyzed. Workflow for classification phase is shown in Figure 4.2

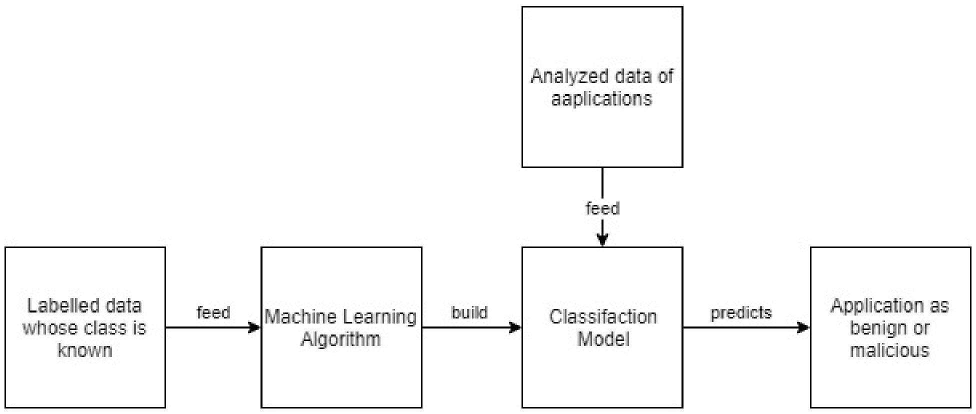

Analyzed data of aaplications
feed
Labelled data whose class is known
feed
Machine Learning Algorithm
build
Classifaction Model
predicts
Application as benign or malicious

Figure 4.2: Workflow of classification phase

Once we know that whether applications are malicious or benign, then we only focus of malicious applications. We try to restrict the malicious behaviour or intent of the applications by using a wrapper around the applications. Figure 4.3 shows the workflow of restriction phase.

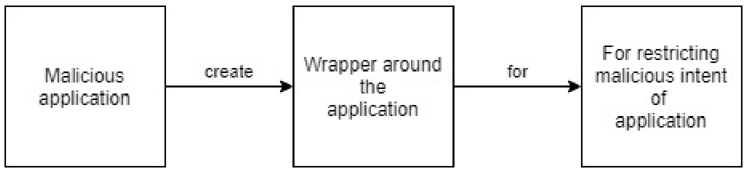

Malicious application
create
Wrapper around the application
for
For restricting malicious intent of application

Figure 4.3: Workflow of restirction phase

4.2 Analysis of Application

In dynamic analysis, we are extracting several information like Unique APIs found, actionable views seen, unique API+event pairs found, and actionable unique views clicked. We trace the API calls that have been invoked while triggering UI elements. In static analysis, we are extracting information about permissions declared and used in application.

For dynamic analysis, we have used DROIDMATE which is fully automatic test generator for Android application and for static analysis we have used RiskInDroid which is python tool used for reverse engineering, malware and goodware analysis of Android application. We have also used AndoPyTool for both Dynamic and static analysis of applications.

4.2.1 Budget Planner

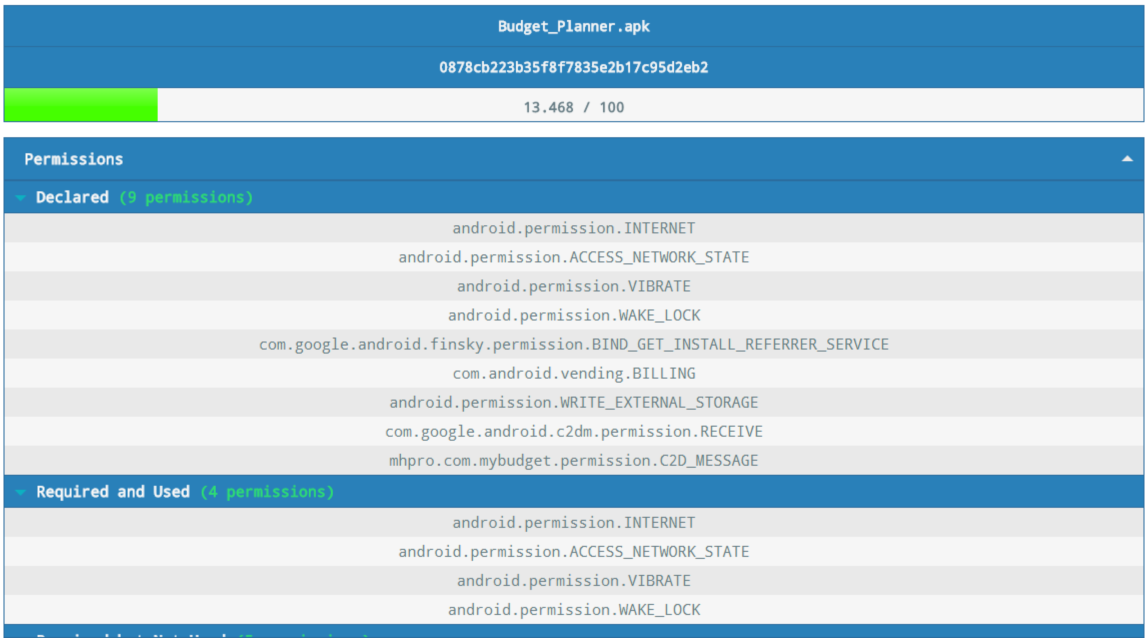
Budget Planner is an Android application which is used for managing your savings and expenses. Figure 4.4 shows the screenshot taken during static analysis and Figure 4.5 shows the screenshot taken during dynamic analysis of application.

Permissions

We have extracted the following 9 permissions during the static analysis of Budget Planner app:

Required and used (4 permissions)

- `android.permission.INTERNET` - Dangerous


Budget_Planner.apk

0878cb223b35f8f7835e2b17c95d2eb2

13.468 / 100

Permissions

Declared (9 permissions)

android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.VIBRATE
android.permission.WAKE_LOCK
com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE
com.android.vending.BILLING
android.permission.WRITE_EXTERNAL_STORAGE
com.google.android.c2dm.permission.RECEIVE
mhpro.com.mybudget.permission.C2D_MESSAGE

Required and Used (4 permissions)

android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.VIBRATE
android.permission.WAKE_LOCK

Figure 4.4: Screenshot captured during static analysis of Budget Planner app

- `android.permission.ACCESS_NETWORK_STATE` - Dangerous
- `android.permission.VIBRATE`
- `android.permission.WAKE_LOCK`

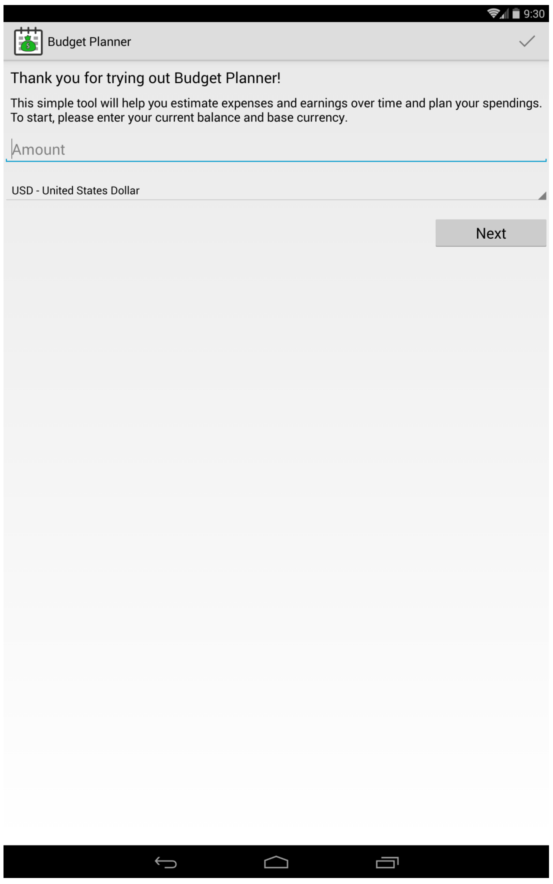
Required but not used (5 permissions)

- `com.google.android.finsky.permission.BIND_GET_INSTALL_REFERRER_SERVICE`
- `com.android.vending.BILLING`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `com.google.android.c2dm.permission.RECEIVE`
- `mhpro.com.mybudget.permission.C2D_MESSAGE`

Unique Android SDK API methods

Number of unique API calls count observed in the run is 4. Following is list of first calls to unique Android SDK API methods:

- `0m 12s - android.telephony.TelephonyManager: java.lang.String getDeviceId()`



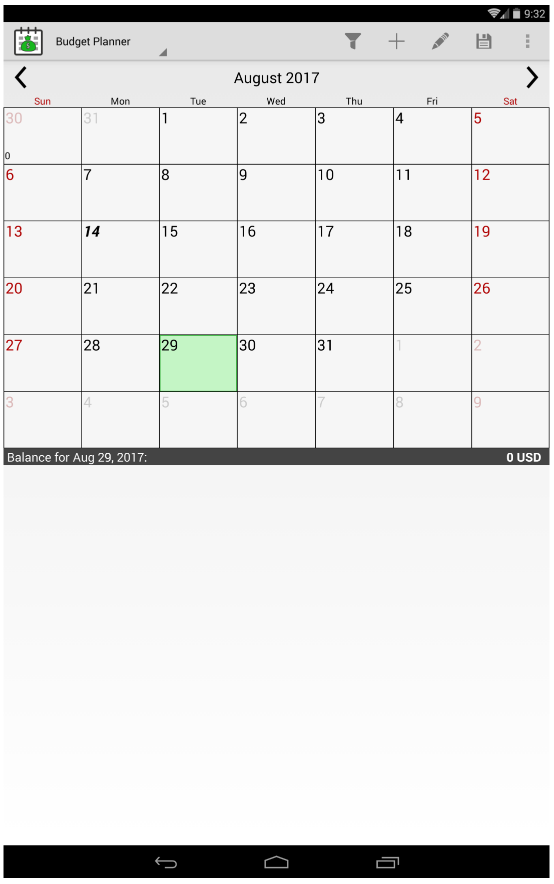
Thank you for trying out Budget Planner!

This simple tool will help you estimate expenses and earnings over time and plan your spendings.
To start, please enter your current balance and base currency.

Amount

USD - United States Dollar

(a) Budget Planner screenshot-1



Balance for Aug 29, 2017: 0 USD

(b) Budget Planner screenshot-2

Figure 4.5: Screenshot captured during analysis

- `0m 12s - org.apache.http.impl.client.AbstractHttpClient: org.apache.http.HttpResponse (org.apache.http.HttpHost, org.apache.http.protocol.HttpContext)`
- `0m 13s - java.net.Socket: void connect (java.net.SocketAddress,int)`
- `0m 13s - android.location.LocationManager: java.lang.String getBestProvider (android.location.Criteria,boolean)`

Unique [API call, event] Android SDK API methods

Number of unique [API call, event] pairs count observed in the run is 5. Following is the list of first calls to unique Android SDK API methods:

- `0m 12s - reset - android.telephony.TelephonyManager: java.lang.String getDeviceId()`

- `0m 12s - reset - org.apache.http.impl.client.AbstractHttpClient: org.apache. HttpResponse (org.apache.http.HttpHost, org.apache.http.protocol.HttpContext)`
- `0m 13s - reset - java.net.Socket: void connect (java.net.SocketAddress,int)`
- `0m 13s - reset - android.location.LocationManager: java.lang.String.get. BestProvider (android.location.Criteria,boolean)`
- `2m 17s - background - android.location.LocationManager:java.lang.String. getBestProvider (android.location.Criteria,boolean)`

Figure 4.6 shows the time at which different APIs and (API call, event) pairs are discovered. Total API discovered is 4 and total (API call,event) pair discoverd is 5.

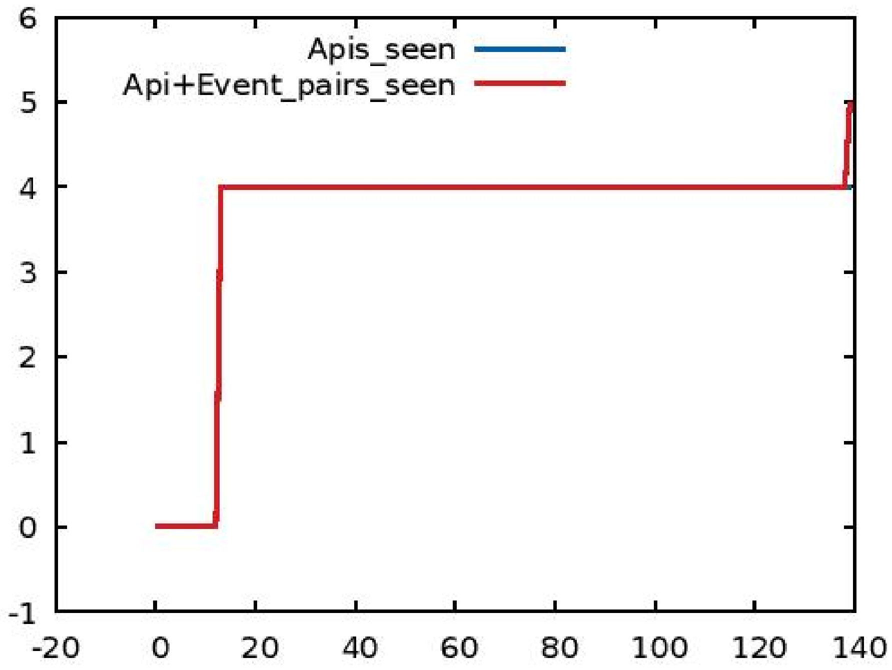


Figure 4.6: API count vs Exploration time

Actionable views

According Figure 4.7, nearly 180 unique actionable views is seen and 33 out of 180 views are clicked. The Table 4.1 shows 32 views are clicked once and one views are clicked thrice.

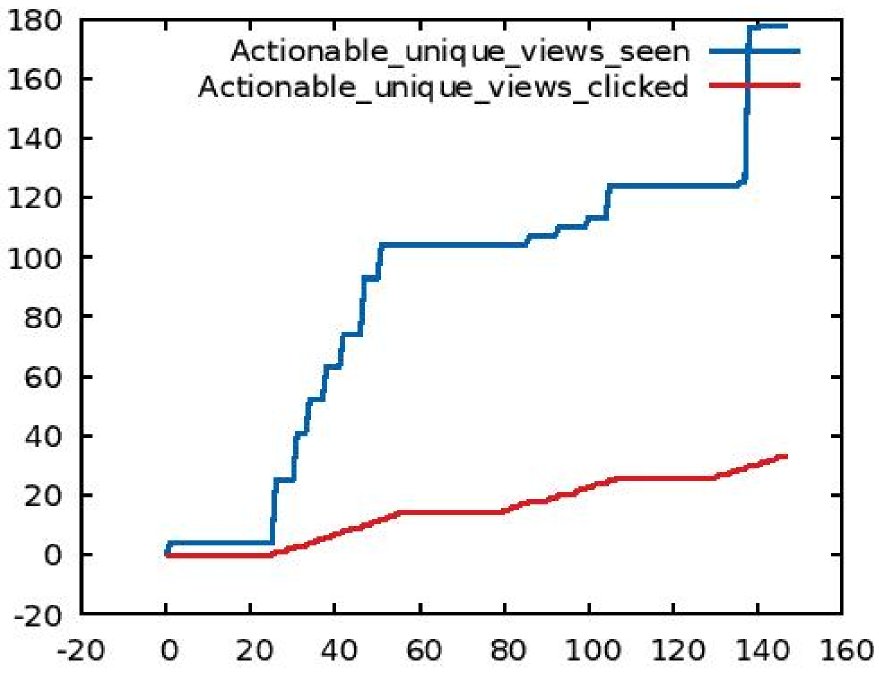


Figure 4.7: API count vs Exploration time

Number of Click	Views Count
0	0
1	32
2	0
3	1

Table 4.1: Click Frequency of views

Aggregate stats

Following is the summary of dynamic analysis of Budget planner application:

- File name - budget_planner-inlined.apk
- Package name - bdget_planner
- Exploration seconds - 153
- Actions - 39
- In this reset actions - 3
- Actionable unique views seen at least once - 178

- Actionable unique views clicked or long clicked at least once - 33
- Unique apis - 4
- Unique event api pairs - 5
- Exception - N/A (lack of DeviceException)

4.2.2 BHIM Making India Cashless

Bharat Interface for Money (BHIM) is an initiative to enable fast, secure, reliable cashless payments through your mobile phone. BHIM is inter operable with other Unified Payment Interface (UPI) applications, bank accounts for quick money transfers online. Figure 4.8 shows screenshot taken during the analysis of BHIM applications.

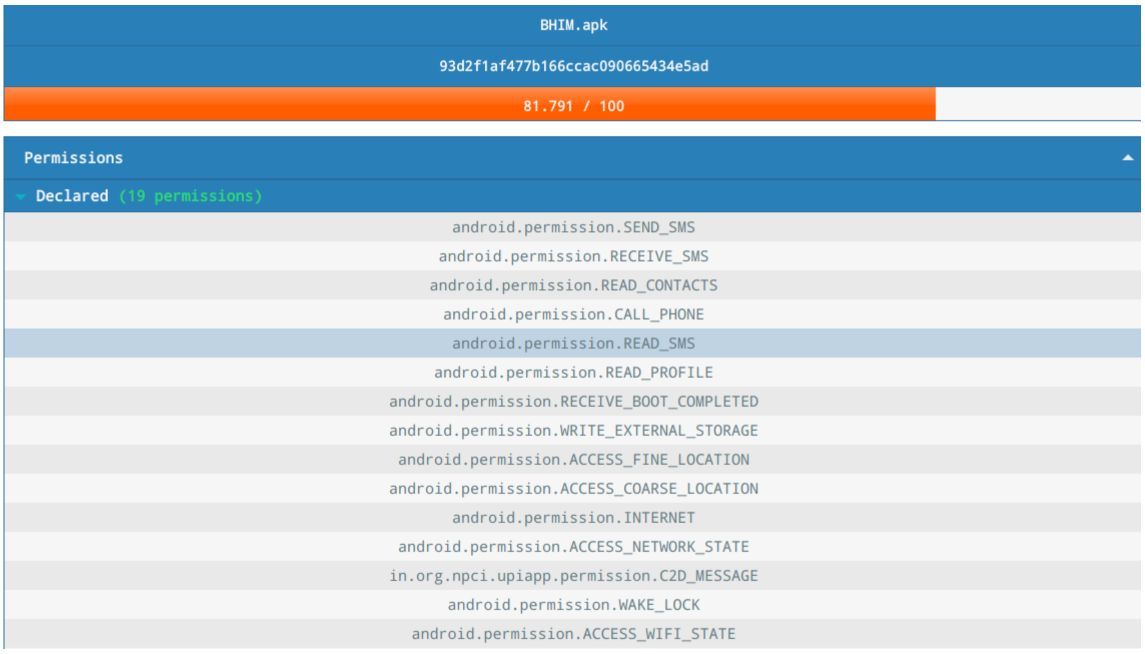

BHIM.apk
93d2f1af477b166ccac090665434e5ad
81.791 / 100
Permissions
Declared (19 permissions)
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.READ_CONTACTS
android.permission.CALL_PHONE
android.permission.READ_SMS
android.permission.READ_PROFILE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
in.org.npci.upiapp.permission.C2D_MESSAGE
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE

Figure 4.8: Screenshot captured during analysis of BHIM app

Permissions

We have extracted the following permissions during the static analysis of BHIM app:

Required and used (6 permissions)

- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous

- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.READ_PHONE_STATE` - Dangerous

Required but not used(13 permissions)

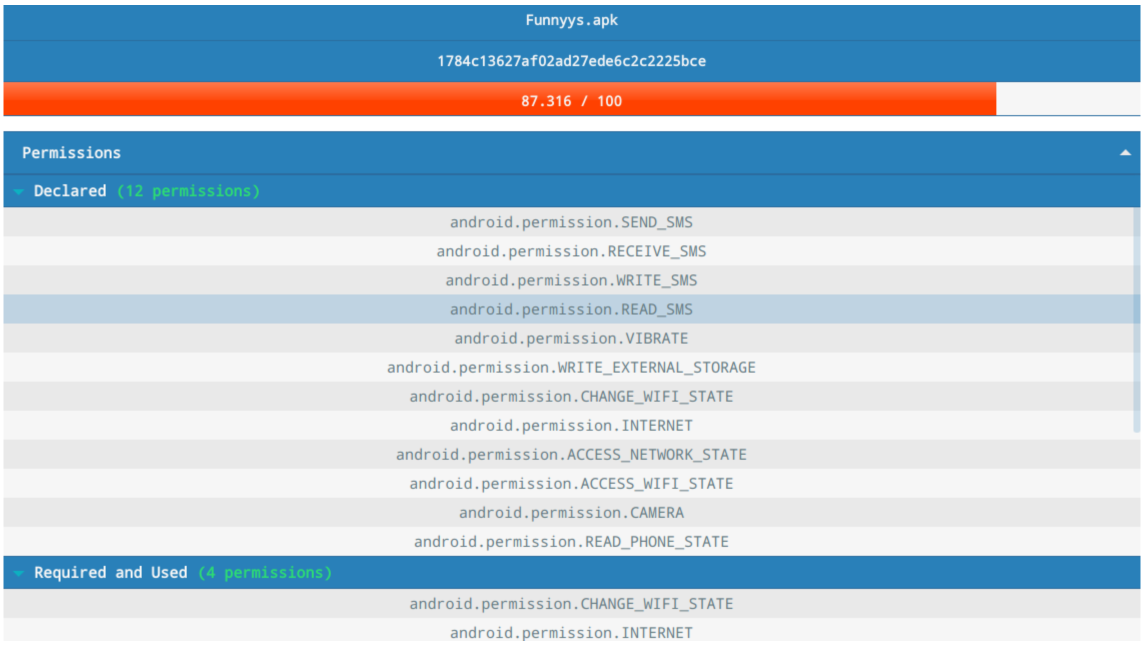
- `android.permission.SEND_SMS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.READ_CONTACTS` - Dangerous
- `android.permission.CALL_PHONE` - Dangerous
- `android.permission.READ_SMS` - Dangerous
- `android.permission.READ_PROFILE`
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `in.org.npci.upiapp.permission.C2D_MESSAGE`
- `android.permission.ACCESS_WIFI_STATE`
- `android.permission.CAMERA` - Dangerous
- `android.permission.READ_EXTERNAL_STORAGE` - Dangerous
- `com.google.android.c2dm.permission.RECEIVE`

Not required but used(1 permissions)

- `android.permission.VIBRATE`

4.2.3 Funnyys

Funnyys is an application which can add charges to your mobile bill by sending costly SMS messages without informing you first. Figure 4.9 shows screenshot taken during analysis Funnyys app.


Funnyys.apk

1784c13627af02ad27ede6c2c2225bce

87.316 / 100

Permissions

Declared (12 permissions)

android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.WRITE_SMS
android.permission.READ_SMS
android.permission.VIBRATE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.CHANGE_WIFI_STATE
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.CAMERA
android.permission.READ_PHONE_STATE

Required and Used (4 permissions)

android.permission.CHANGE_WIFI_STATE
android.permission.INTERNET

Figure 4.9: Screenshot captured during analysis of Funnyys app

Permissions

We have extracted the following permissions during the static analysis of Funnyys app:

Required and used (4 permissions)

- `android.permission.CHANGE_WIFI_STATE`
- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.ACCESS_WIFI_STATE`

Required but not used (8 permissions)

- `android.permission.SEND_SMS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.WRITE_SMS` - Dangerous
- `android.permission.READ_SMS` - Dangerous
- `android.permission.VIBRATE`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `android.permission.CAMERA` - Dangerous

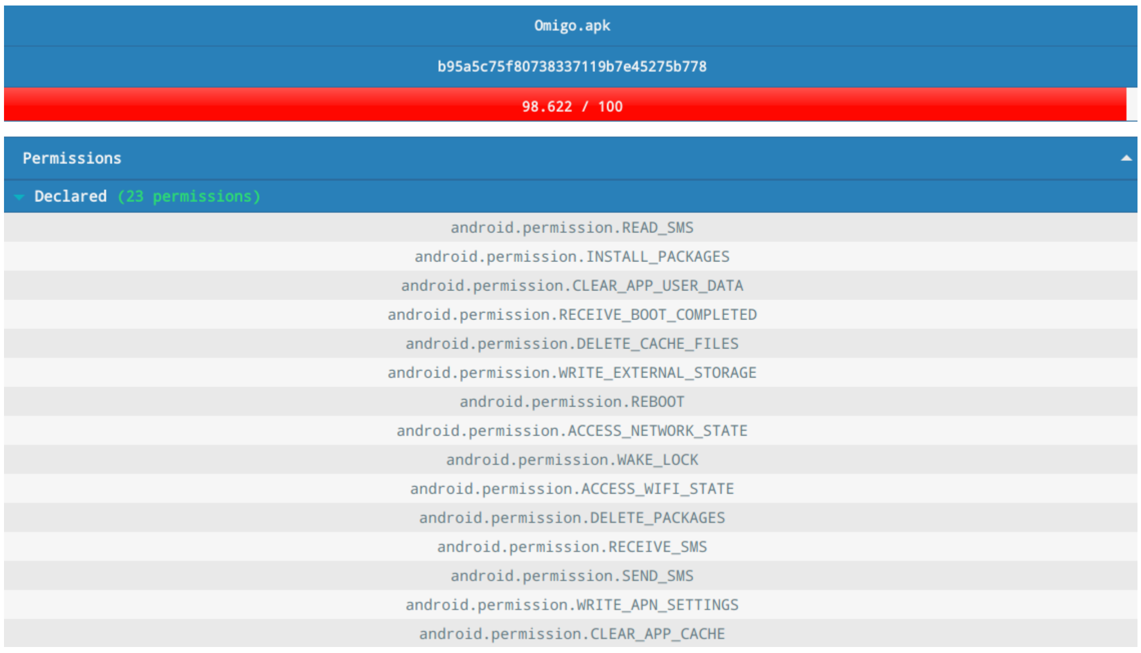
- `android.permission.READ_PHONE_STATE` - Dangerous

Not required but used(1 permission)

- `android.permission.WAKE_LOCK`

4.2.4 Omingo

This app lets hackers control your device, giving them unauthorized access to your data. Figure 4.10 shows screenshot taken during analysis of Omingo app.



Omigo.apk
b95a5c75f80738337119b7e45275b778
98.622 / 100
Permissions
Declared (23 permissions)

```
android.permission.READ_SMS
android.permission.INSTALL_PACKAGES
android.permission.CLEAR_APP_USER_DATA
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.DELETE_CACHE_FILES
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.REBOOT
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.DELETE_PACKAGES
android.permission.RECEIVE_SMS
android.permission.SEND_SMS
android.permission.WRITE_APN_SETTINGS
android.permission.CLEAR_APP_CACHE
```

Figure 4.10: Screenshot captured during analysis of Omingo app

Permissions

We have extracted the following permissions during the static analysis of Omingo app:

Required and used (4 permissions)

- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.ACCESS_WIFI_STATE`
- `android.permission.READ_PHONE_STATE` - Dangerous

Required but not used (19 permissions)

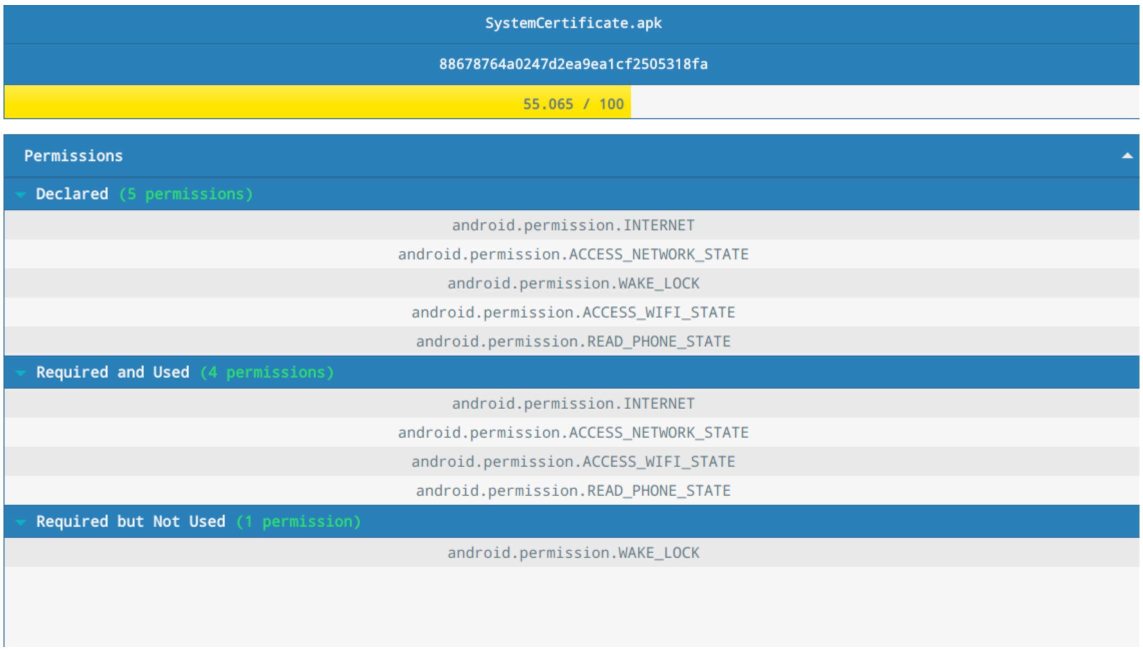
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.SEND_SMS` - Dangerous
- `android.permission.WRITE_APN_SETTINGS`
- `android.permission.CLEAR_APP_CACHE`
- `android.permission.READ_SMS` - Dangerous
- `android.permission.RECEIVE_WAP_PUSH`
- `android.permission.INSTALL_PACKAGES` - Dangerous
- `android.permission.CLEAR_APP_USER_DATA`
- `android.permission.MOUNT_UNMOUNT_FILESYSTEMS`
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.DELETE_CACHE_FILES`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `android.permission.REBOOT` - Dangerous
- `android.permission.RESTART_PACKAGES` - Dangerous
- `android.permission.CHANGE_WIFI_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.CHANGE_NETWORK_STATE`
- `android.permission.SET_WALLPAPER`
- `android.permission.DELETE_PACKAGES` - Dangerous

Not required but used(3 permission)

- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous
- `android.permission.VIBRATE`

4.2.5 System Certificate

System Certificate is a fake application which can damage your device and steal your data. Figure 4.11 shows the screenshot taken during static analysis of System Certificate app.


SystemCertificate.apk
88678764a0247d2ea9ea1cf2505318fa
55.065 / 100

Permissions
Declared (5 permissions)
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.READ_PHONE_STATE
Required and Used (4 permissions)
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.READ_PHONE_STATE
Required but Not Used (1 permission)
android.permission.WAKE_LOCK

Figure 4.11: Screenshot captured during analysis of System Certificate app

Permissions

We have extracted the following permissions during the static analysis of System Certificate app:

Required and used (4 permissions)

- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.ACCESS_WIFI_STATE`
- `android.permission.READ_PHONE_STATE` - Dangerous

Required but not used (1 permissions)

- `android.permission.WAKE_LOCK`

4.2.6 Tez – A new payments app by Google

Tez is an application send money to others. It is developed by Google. It is based on UPI. Figure 4.12 shows the screenshot taken during the analysis Tez app.

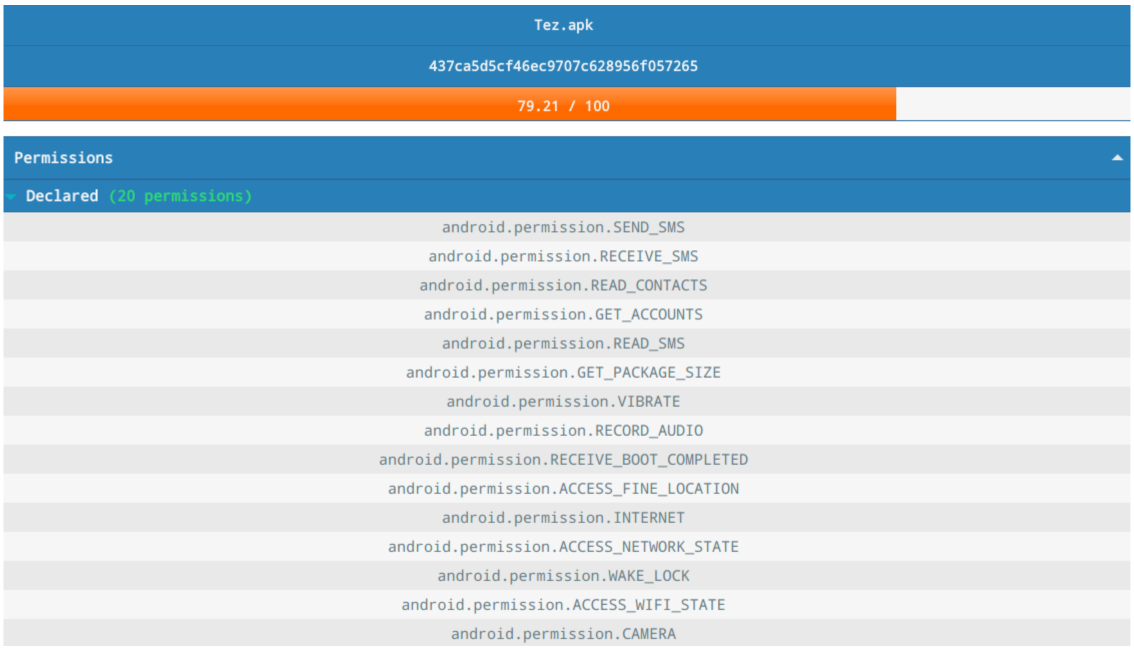

Tez.apk
437ca5d5cf46ec9707c628956f057265
79.21 / 100
Permissions
Declared (20 permissions)
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.READ_CONTACTS
android.permission.GET_ACCOUNTS
android.permission.READ_SMS
android.permission.GET_PACKAGE_SIZE
android.permission.VIBRATE
android.permission.RECORD_AUDIO
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.ACCESS_FINE_LOCATION
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.ACCESS_WIFI_STATE
android.permission.CAMERA

Figure 4.12: Screenshot captured during analysis of Tez app

Permissions

We have extracted the following permissions during the static analysis of Tez app:

Required and used (6 permissions)

- `android.permission.INTERNET`
- `android.permission.VIBRATE`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.BLUETOOTH`
- `android.permission.READ_PHONE_STATE` - Dangerous

Required but not used (14 permissions)

- `android.permission.SEND_SMS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.READ_CONTACTS` - Dangerous
- `android.permission.GET_ACCOUNTS` - Dangerous
- `android.permission.READ_SMS` - Dangerous
- `android.permission.GET_PACKAGE_SIZE`

- `android.permission.RECORD_AUDIO` - Dangerous
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_WIFI_STATE`
- `android.permission.CAMERA` - Dangerous
- `android.permission.MANAGE_ACCOUNTS` - Dangerous
- `com.google.android.c2dm.permission.RECEIVE`
- `com.google.android.providers.gsf.permission.READ_GSERVICES`

Not required but used (2 permission)

- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous
- `android.permission.MODIFY_AUDIO_SETTINGS`

4.2.7 MMS Beline

MMS Beline is an application which can call phone number and also send SMS without informing you, which will lead to increase in mobile bill. This application can steal your data and can run with administrative permissions. Figure 4.13 shows the screenshot taken during the static analysis of MMS Beline.

Permissions

We have extracted the following permissions during the static analysis of MMS beline app:

Required and used (5 permissions)

- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous
- `android.permission.INTERNET`
- `android.permission.GET_TASKS`
- `android.permission.READ_PHONE_STATE` - Dangerous

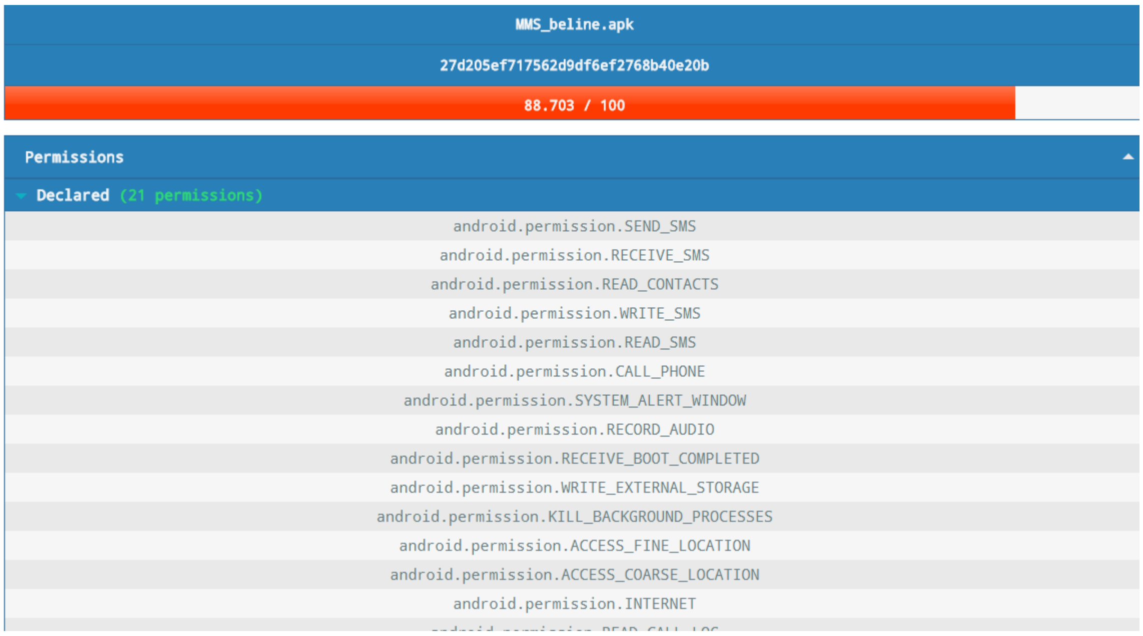

MMS_beline.apk
27d205ef717562d9df6ef2768b40e20b
88.703 / 100
Permissions
Declared (21 permissions)
android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.READ_CONTACTS
android.permission.WRITE_SMS
android.permission.READ_SMS
android.permission.CALL_PHONE
android.permission.SYSTEM_ALERT_WINDOW
android.permission.RECORD_AUDIO
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.KILL_BACKGROUND_PROCESSES
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.INTERNET

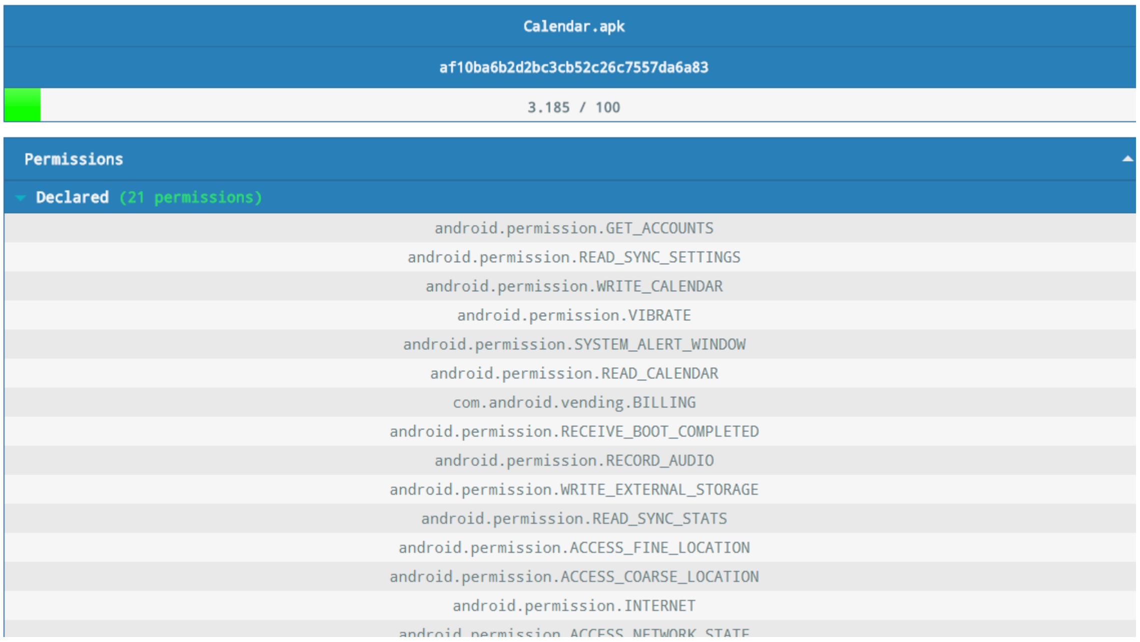
Figure 4.13: Screenshot captured during analysis of MMS Beline app

Required but not used (16 permissions)

- `android.permission.SEND_SMS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.READ_CONTACTS` - Dangerous
- `android.permission.WRITE_SMS` - Dangerous
- `android.permission.READ_SMS` - Dangerous
- `android.permission.CALL_PHONE` - Dangerous
- `android.permission.SYSTEM_ALERT_WINDOW`
- `android.permission.RECORD_AUDIO` - Dangerous
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `android.permission.KILL_BACKGROUND_PROCESSES` - Dangerous
- `android.permission.READ_CALL_LOG` - Dangerous
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.CAMERA` - Dangerous
- `android.permission.READ_EXTERNAL_STORAGE` - Dangerous

4.2.8 Google Calendar

Google Calendar is the free time management web application offered by Google.Google Calendar allows users to create and edit events. Reminders can be enabled for events, with options available for type and time. Event locations can also be added, and other users can be invited to events. Users can enable or disable the visibility of special calendars, including Birthdays, where the app retrieves dates of births from Google contacts and displays birthday cards on a yearly basis, and Holidays, a country-specific calendar that displays dates of special occasions. Figure 4.14 shows the screenshot taken during the static analysis of Calendar app.


Calendar.apk
af10ba6b2d2bc3cb52c26c7557da6a83
3.185 / 100

Permissions
Declared (21 permissions)

android.permission.GET_ACCOUNTS
android.permission.READ_SYNC_SETTINGS
android.permission.WRITE_CALENDAR
android.permission.VIBRATE
android.permission.SYSTEM_ALERT_WINDOW
android.permission.READ_CALENDAR
com.android.vending.BILLING
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.RECORD_AUDIO
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.READ_SYNC_STATS
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE

Figure 4.14: Screenshot captured during analysis

Permissions

We have extracted the following permissions during the static analysis of Calendar app:

Required and used (8 permissions)

- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous
- `android.permission.INTERNET`
- `android.permission.VIBRATE`

- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.RECORD_AUDIO` - Dangerous
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous

Required but not used (13 permissions)

- `android.permission.GET_ACCOUNTS` - Dangerous
- `android.permission.READ_SYNC_SETTINGS`
- `android.permission.WRITE_CALENDAR` - Dangerous
- `android.permission.SYSTEM_ALERT_WINDOW`
- `android.permission.READ_CALENDAR` - Dangerous
- `com.android.vending.BILLING`
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.READ_SYNC_STATS`
- `android.permission.GET_TASKS`
- `android.permission.READ_EXTERNAL_STORAGE` - Dangerous
- `android.permission.WRITE_SETTINGS`
- `android.permission.READ_PHONE_STATE`
- `com.google.android.providers.gsf.permission.READ_GSERVICES`

Unique Android SDK API methods

Number of unique API calls count observed in the run is 4. Following is list of first calls to unique Android SDK API methods:

- `0m 12s - android.app.ActivityThread: void installContentProviders (android.content.Context, java.util.List)`
- `0m 12s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String,android.os. CancellationSignal)`

- `0m 12s - android.content.ContentResolver: android.database. Cursor query (android.net.Uri, java.lang.String[], java.lang.String,java.lang.String[], java.lang.String, android.os. CancellationSignal)`
- `0m 12s - android.content.ContentResolver: android.database.Cursor query ( android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String,android.os. CancellationSignal)`
- `0m 12s - android.content.ContentResolver:void registerContentObserver( android.net.Uri, boolean ,android.database.ContentObserver, int)`
- `0m 12s - android.content.ContentResolver: void registerContentObserver (android.net.Uri, boolean, android.database.ContentObserver, int)`
- `0m 12s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, android.os. CancellationSignal)`
- `0m 12s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String,android.os. CancellationSignal)`
- `0m 24s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String,java.lang.String[], java.lang.String,android.os. CancellationSignal)`
- `0m 27s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String,java.lang.String[], java.lang.String,android.os. CancellationSignal)`
- `0m 31s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java. lang.String,java.lang.String[], j ava.lang.String,android.os. CancellationSignal)`
- `0m 33s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[],`

`java.lang.String, android.os. CancellationSignal)`

- `0m 38s - android.content.ContentResolver: android.database. Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String,android. os.CancellationSignal)`
- `0m 38s - android.content.ContentResolver: void registerContentObserver (android.net.Uri, boolean, android.database.ContentObserver, int)`
- `1m 21s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, android.os. CancellationSignal)`
- `1m 23s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String,android.os. CancellationSignal)`
- `1m 54s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, android.os. CancellationSignal)`
- `1m 56s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, android.os. CancellationSignal)`
- `2m 11s - android.content.ContentResolver: android.database.Cursor query (android.net.Uri, java.lang.String[], java.lang.String, java.lang.String[], java.lang.String, android.os. CancellationSignal)`

Figure 4.15 shows the time at which different APIs and (API call, event) pairs are discovered. Total API discovered is 4 and total (API call,event) pair discovered is 5.

Actionable views

According Figure 4.16, nearly 180 unique actionable views is seen and 33 out of 180 views are clicked. The Table 4.2 shows 18 views are clicked once and 6 views are clicked twice.

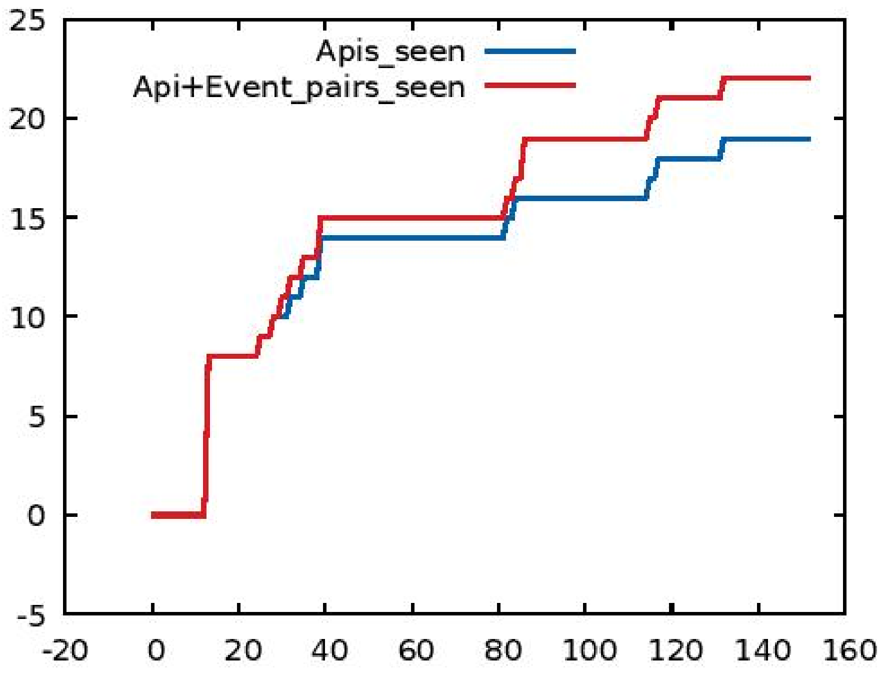


Figure 4.15: API count vs Exploration time

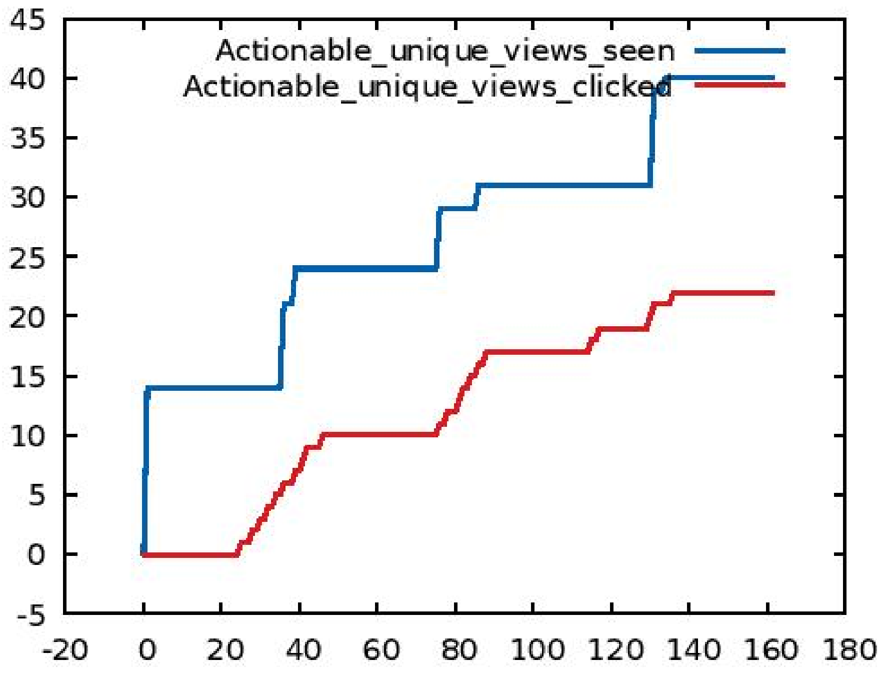


Figure 4.16: API count vs Exploration time

Aggregate stats

Following is the summary of dynamic analysis of Google Calendar application:

Number of Click	Views Count
0	0
1	18
2	6

Table 4.2: Click Frequency of views

- File name - Calendar-inlined.apk
- Package name - com.google.android.calendar
- Exploration seconds - 167
- Actions - 35
- In this reset actions - 4
- Actionable unique views seen at least once - 40
- Actionable unique views clicked or long clicked at least once - 22
- Unique apis - 19
- Unique event api pairs - 22
- Exception - N/A (lack of DeviceException)

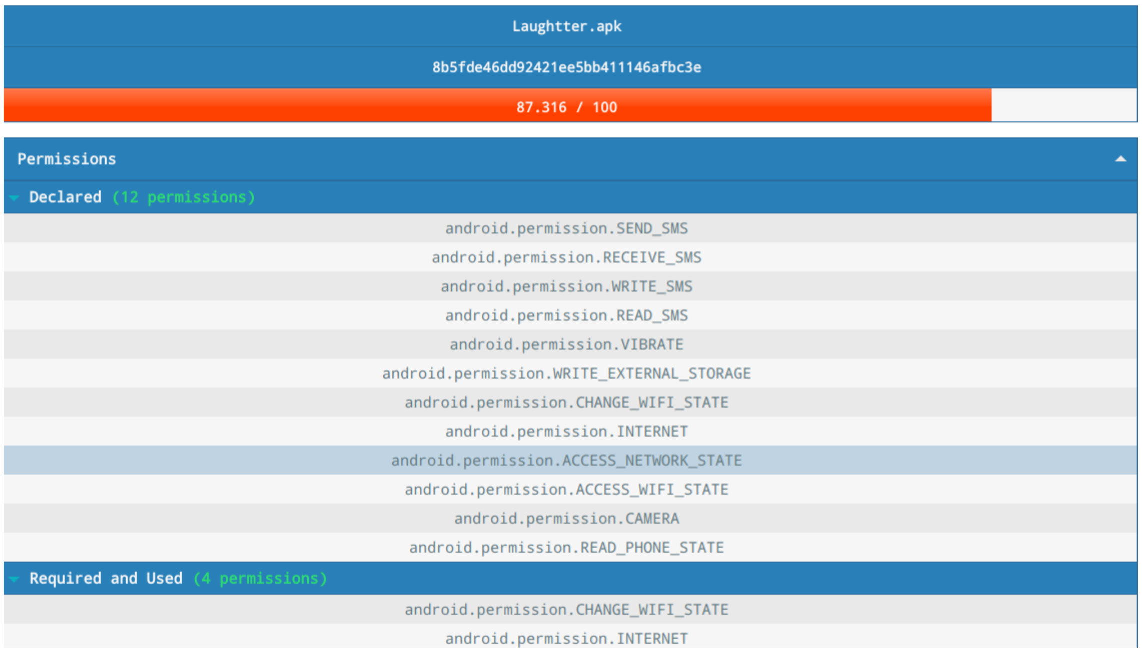
4.2.9 Laughtter

Laughtter is an application which can add charges to your mobile bill by sending costly SMS message without informing you first. Figure 4.17 shows the screenshot taken during analysis of Laughtter app.

Permissions

We have extracted the following permissions during the static analysis of Laughtter app:
Required and used (4 permissions)

- `android.permission.CHANGE_WIFI_STATE`
- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`


Laughtter.apk

8b5fde46dd92421ee5bb411146afbc3e

87.316 / 100

Permissions

Declared (12 permissions)

android.permission.SEND_SMS
android.permission.RECEIVE_SMS
android.permission.WRITE_SMS
android.permission.READ_SMS
android.permission.VIBRATE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.CHANGE_WIFI_STATE
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.CAMERA
android.permission.READ_PHONE_STATE

Required and Used (4 permissions)

android.permission.CHANGE_WIFI_STATE
android.permission.INTERNET

Figure 4.17: Screenshot captured during analysis of Laughtter app

- `android.permission.ACCESS_WIFI_STATE`

Required but not used (8 permissions)

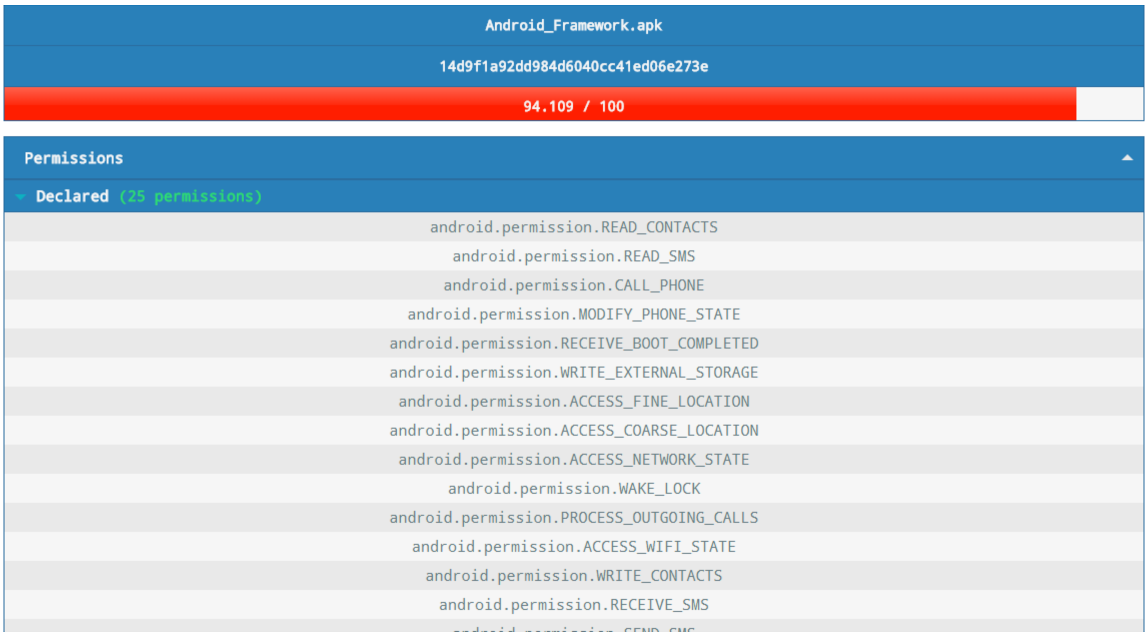
- `android.permission.SEND_SMS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.WRITE_SMS` - Dangerous
- `android.permission.READ_SMS` - Dangerous
- `android.permission.VIBRATE`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `android.permission.CAMERA` - Dangerous
- `android.permission.READ_PHONE_STATE` - Dangerous

Not required but used (1 permissions)

- `android.permission.WAKE_LOCK`

4.2.10 Android Framework

Android Framework lets control your device, giving them unauthorized access to your data. Figure 4.18 shows the screenshot taken during static analysis of the application.



Android_Framework.apk

14d9f1a92dd984d6040cc41ed06e273e

94.109 / 100

Permissions

Declared (25 permissions)

android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.CALL_PHONE
android.permission.MODIFY_PHONE_STATE
android.permission.RECEIVE_BOOT_COMPLETED
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.ACCESS_FINE_LOCATION
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_NETWORK_STATE
android.permission.WAKE_LOCK
android.permission.PROCESS_OUTGOING_CALLS
android.permission.ACCESS_WIFI_STATE
android.permission.WRITE_CONTACTS
android.permission.RECEIVE_SMS

Figure 4.18: Screenshot captured during analysis of Android Framework app

Permissions

We have extracted following permissions during the analysis of Android Framework application:

Requested and used (9 permissions)

- `android.permission.ACCESS_FINE_LOCATION` - Dangerous
- `android.permission.ACCESS_COARSE_LOCATION` - Dangerous
- `android.permission.CHANGE_WIFI_STATE`
- `android.permission.INTERNET`
- `android.permission.ACCESS_NETWORK_STATE`
- `android.permission.WAKE_LOCK`
- `android.permission.ACCESS_WIFI_STATE`
- `android.permission.RECORD_AUDIO` - Dangerous
- `android.permission.READ_PHONE_STATE` - Dangerous

Requested but not used (16 permissions)

- `android.permission.READ_CONTACTS` - Dangerous
- `android.permission.READ_SMS` - Dangerous

- `android.permission.CALL_PHONE` - Dangerous
- `android.permission.MODIFY_PHONE_STATE` - Dangerous
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.WRITE_EXTERNAL_STORAGE` - Dangerous
- `android.permission.PROCESS_OUTGOING_CALLS` - Dangerous
- `android.permission.WRITE_CONTACTS` - Dangerous
- `android.permission.RECEIVE_SMS` - Dangerous
- `android.permission.SEND_SMS` - Dangerous
- `android.permission.WRITE_APN_SETTINGS`
- `android.permission.WRITE_SMS` - Dangerous
- `android.permission.BROADCAST_PACKAGE_REMOVED`
- `android.permission.CHANGE_NETWORK_STATE`
- `android.permission.MODIFY_AUDIO_SETTINGS`
- `android.permission.READ_EXTERNAL_STORAGE` - Dangerous

Chapter 5

App Classification Models

Classification is the technique to classify new observation on the basis of a set of data containing observations whose class is known [22]. Classification is considered as an instance of supervised learning i.e. learning where training set of correctly identified observation is available. To build a classification model we need labelled data. We train classification model with using machine learning algorithm. Figure 5.1 shows the workflow of building the model. Once the model is trained, we feed new observation into classification model to classify them. Figure 5.2 shows the architecture of classifying the new observation using trained Model.

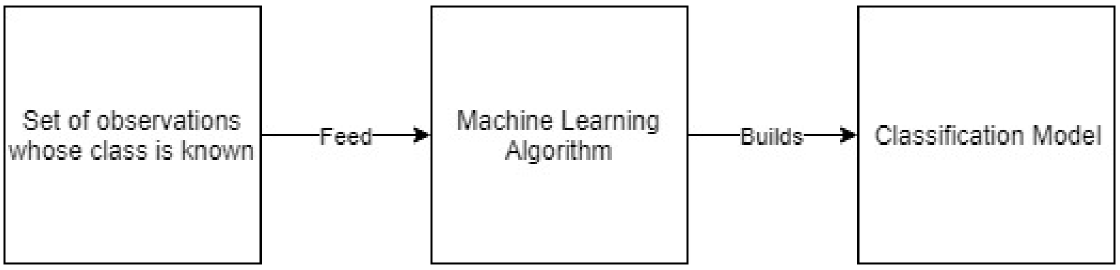

Set of observations whose class is known
Feed
Machine Learning Algorithm
Builds
Classification Model

Figure 5.1: Workflow of building a classification model.

To check the performance of model we have used confusion matrix [17]. A confusion matrix is a summary of prediction results on a classification problem. The number of correct and incorrect predictions are summarized with count values and broken down by each class. The confusion matrix shows the ways in which your classification model is confused when it makes predictions. It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made. A

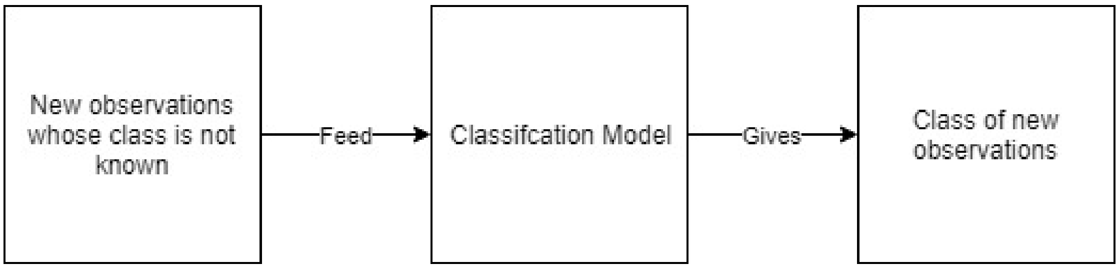

New observations whose class is not known
Feed
Classifcation Model
Gives
Class of new observations

Figure 5.2: Workflow of classifying the new observations.

classification model is good if there is high value along the diagonal of confusion matrix. Table 5.1 shows the general confusion matrix.

Actual	Predicted	
	Benign	Malicious
Benign	TB	FM
Malicious	FB	TM

Table 5.1: Confusion Matrix

Definition of the Terms:

- Benign (B): Observation is benign.
- Malicious (M): Observation is malicious.
- True Benign (TB): Observation is benign, and is predicted to be benign.
- False Malicious (FM): Observation is benign, but is predicted malicious.
- True Malicious (TM): Observation is malicious, and is predicted to be malicious.
- False Benign (FM): Observation is malicious, but is predicted benign.

We have collected the labelled data of 26000+ applications [13]. Data set contains observations/traces of 25593 benign applications and 741 malicious applications. API calls has been used as the feature vector for application. Each application has 854 features. We have developed various classification models using machine learning techniques to classify new applications as benign or malicious. We have used following machine learning techniques:

1. Logistic Regression
2. Support Vector Machine
3. Neural Network
4. Random Forest

5.1 Logistic Regression

Logistic Regression is a classification technique which is is usually taken to apply to a binary dependent variable [20]. We have divided the data set into training set which is 80% of the total data set and testing set which 20% of the total data set. We have trained our model on the training set and have checked the performance of the model on test data. Model classifies 93.8% of benign correctly and 83.4% of malicious app correctly. Table 5.2 shows the confusion matrix obtained from Logistic regression model. We have

Actual	Predicted	
	benign	malicious
benign	93.8	6.2
malicious	16.6	83.4

Table 5.2: Confusion matrix for Logistic regression

used this model to predict the class of applications which we have analyzed. Following application is classified as benign:

- Budget Planner
- BHIM Making India Cashless
- System Certificate
- Tez: A new payment app by Google
- Google Calendar

Following application is classified as malicious:

- Funnyys

- Omingo
- MMS Beline
- Laughtter
- Android Framework

5.2 Support Vector Machine

In machine learning, Support Vector Machine (SVM) is considered as supervised learning models associated with learning algorithms [23]. Here, we have used this model for classification. We have divided the data set into training set which is 80% of the total data set and testing set which 20% of the total data set. We have trained our model on the training set and have checked the performance of the model on test data. Model classifies 92.6% of benign correctly and 78.7% of malicious app correctly. Table 5.3 shows the confusion matrix obtained from Support Vector Machine model. We have used this

Actual	Predicted	
	benign	malicious
benign	92.6	7.4
malicious	21.3	78.7

Table 5.3: Confusion matrix for Support Vector Machine

model to predict the class of applications which we have analyzed. Following application is classified as benign:

- Budget Planner
- Funnyys
- BHIM Making India Cashless
- System Certificate
- Tez: A new payment app by Google
- Google Calendar

Following application is classified as malicious:

- Omingo
- MMS Beline
- Laughtter
- Android Framework

5.3 Neural Network

A Artificial Neural Network (ANN) or Neural Network (NN) is a machine learning technique which is inspired from the biological nervous system [16]. An ANN is based on a collection of connected units or nodes called artificial neurons which loosely model the neurons in a biological brain. We have divided the data set into training set which is 80% of the total data set and testing set which 20% of the total data set. We have trained our model on the training set and have checked the performance of the model on test data. Model classifies 96.3% of benign correctly and 86.6% of malicious app correctly. Table 5.4 shows the confusion matrix obtained from Neural Network model. We have used this

Actual	Predicted	
	benign	malicious
benign	96.3	3.7
malicious	13.4	86.6

Table 5.4: Confusion matrix for Neural Network

model to predict the class of applications which we have analyzed. Following application is classified as benign:

- Budget Planner
- BHIM Making India Cashless
- Tez: A new payment app by Google
- Google Calendar

Following application is classified as malicious:

- Funnyys
- Omingo
- System Certificate
- MMS Beline
- Laughtter
- Android Framework

5.4 Random Forest

Random Forest is a machine learning technique for classification, regression and other tasks [21]. It is made up of various decision trees [19]. We have used random forest for classification. We have divided the data set into training set which is 80% of the total data set and testing set which 20% of the total data set. We have trained our model on the training set and have checked the performance of the model on test data. Model classifies 97.2% of benign correctly and 87.1% of malicious app correctly. Table 5.5 shows the confusion matrix obtained from Random Forest model. We have used this model to

Actual	Predicted	
	benign	malicious
benign	97.2	2.8
malicious	12.9	87.1

Table 5.5: Confusion matrix for Random Forest

predict the class of applications which we have analyzed. Following application is classified as benign:

- **Benigin apps:** Budget Planner, BHIM Making India Cashless, Tez and Google Calendar.
- **Malicious apps:** Funnyys, Omingo, System Certificate, MMS Beline, Laughtter and Android Framework.

5.5 Comparison

We have used four classification algorithms namely Logistic Regression, Support Vector Machine (SVM), Neural Network, and Random Forest. Table 5.6 shows the comparison of accuracy of different algorithms.

Algorithm	**Accuracy of benign apps**	**Accuracy of malicious app**
Logistic Regression	93.8%	83.4%
Support Vector Machine	92.6%	78.7%
Neural Network	96.3%	86.6%
Random Forest	97.2%	87.1%

Table 5.6: Comparison of accuracy of different algorithms

Table 5.7 gives the information about whether app is identified as benign or malicious by various machine learning algorithms.

Algorithm	**Logistic Regression**	**SVM**	**Neural Network**	**Random Forest**
Budget Planner	benign	benign	benign	benign
BHIM	benign	benign	benign	benign
Funnyys	malicious	benign	malicious	malicious
Omingo	malicious	malicious	malicious	malicious
System Certificate	benign	benign	malicious	malicious
Tez	benign	benign	benign	benign
MMS Beline	malicious	malicious	malicious	malicious
Google Calendar	benign	benign	benign	benign
Laughtter	malicious	malicious	malicious	malicious
Android Framework	malicious	malicious	malicious	malicious

Table 5.7: Categorization of apps into benign and malicious by various algorithm

Chapter 6

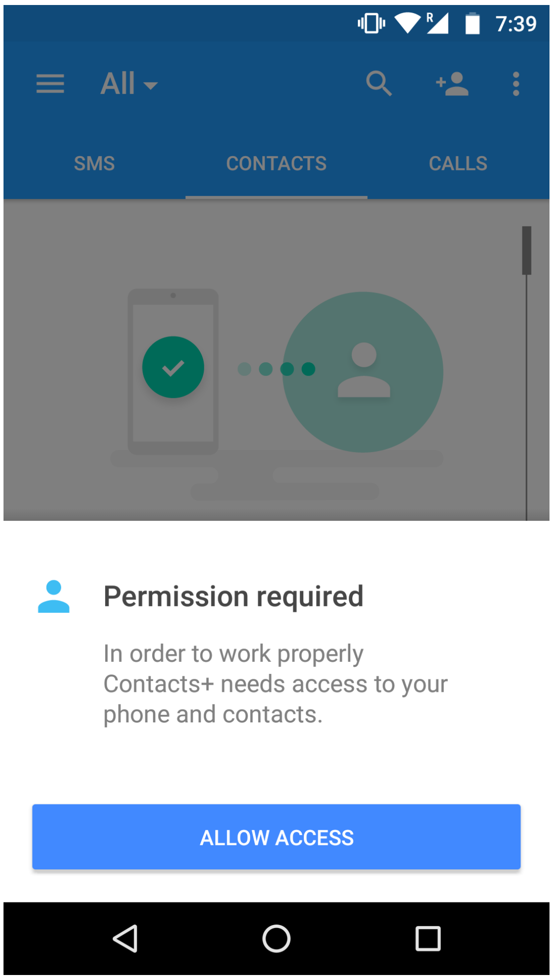
Restricting Malicious Intent of Application

Android current security system is based on the permissions. If an application needs to access any sensitive data or sensors which is considered as dangerous, then they need permissions to do that. In current security scenario, we can restrict the particular permission of an application. But major problem with current scenario is if we restrict the particular permission of an application, then application will not work properly as shown in Figure 6.1.

Apart from the permissions, there are more things that can be used to exploit the private data of user and also can be used to degrade the performance of the device itself. Some of those are clipboard, view, etc. Clipboard is buffer which is used to hold the data copied from any application. Since clipboard is accessible to all applications, any application can read the data from clipboard and can send it to third party server. Currently, we can to stop applications from accessing clipboard data. Some applications opens the browser via link. We can stop link from opening in the browser.

Currently if an application asks permission to access message, then application will get permissions for both send message and read message. If we want to restrict an application such that it can read message but can not send. In other words, we want to control the permission of application at API level. But currently this feature is not provided by Android operating system.

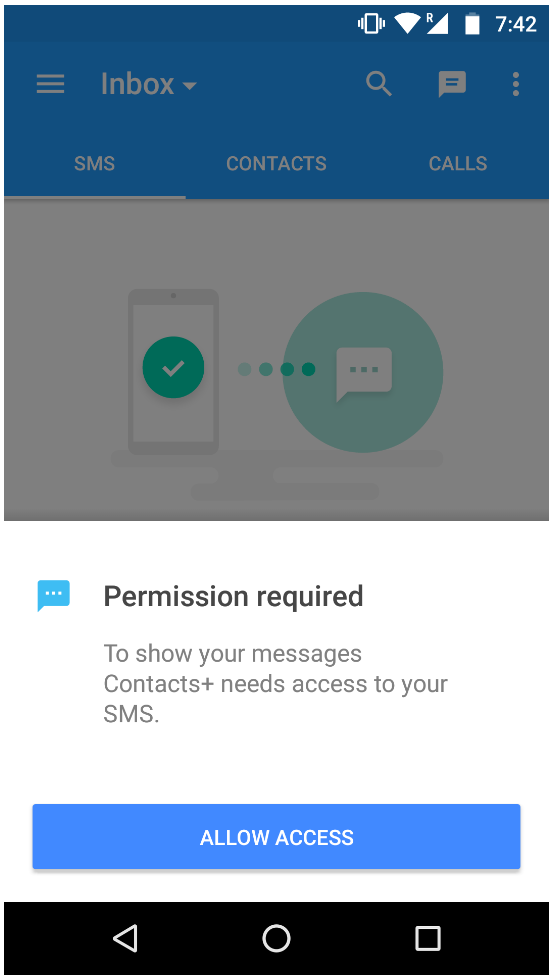
We have to restrict the malicious behaviour of applications. Since, currently every



Permission required

In order to work properly Contacts+ needs access to your phone and contacts.

(a) When permission to access phone is denied.



Permission required

To show your messages Contacts+ needs access to your SMS.

(b) When permission to access message is denied.

Figure 6.1: Problems in Android current security scheme

application runs in its own process, with its own instance of the Dalvik virtual machine (DVK)/ Android runtime (ART). So, we can not restrict applications by developing another at fifth layer of the Android software stack. One way to restrict the malicious behaviour of applications is by modifying the Android Package (APK) file. For modifying the APK file, we have to decompile it and it convert it into smali and then we can change code of application. Smali is more of assembly based language so making change in smali is tedious task. Another way to restrict applications is without modifying the APK file. We can restrict applications at the system level by modifying the Random access memory (ROM). We have used Xposed framework [3] to achieve this.

6.1 Wrapping Applicaitons

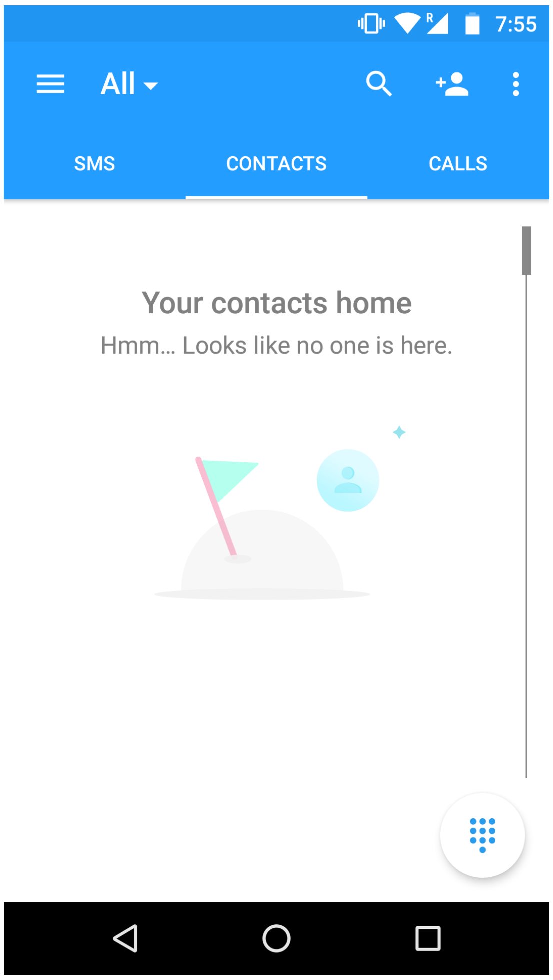
Xposed is framework for module which can restrict or change the behaviour of the system and applications without touching any APKs. Xposed works at system level so your device should be rooted. Instead of modifying corresponding DEX or ART representation of the application, it is deeply integrated with the Android system where it replaces some system components after backing up the originals. There are multiple advantages to this approach. Some of them are:

- you can modify parts of the system you otherwise could not.
- you can apply multiple modifications to the same app in a combination best fitting your intentions.
- changes are easy to undo: As all changes are done in the memory, you just need to deactivate the module and reboot to get your original system back.

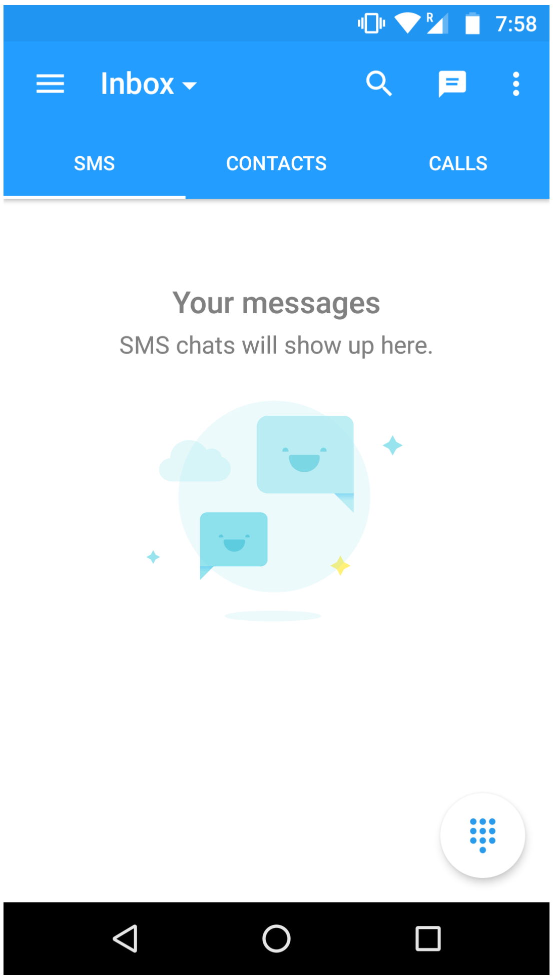
Zygote is the main process of the Android runtime. Every application is started as a copy ("fork") of it. So, this process is also called as the head of Android runtime. This process is started by an */int.rc* script when your phone is booted. The process is done with */system/bin/app_process*, which loads the needed classes and invokes the initialization methods.

Xposed framework creates an extended version of existing app_process. When you install the framework, a modified and extended app_process is copied to */sytem/bin.* This extended startup process adds an additional jar to the classpath and calls methods from there at certain places. For instance, just after the VM has been created, even before the main method of Zygote has been called. And inside that method, we are part of Zygote and can act in its context. So, now we can control the method calls of other application from there.

To provide the security, we have to restrict the method calls from the application. When an application calls any method, we can do three things with that method. Either



(a) When permission to access phone is denied.



(b) When permission to access message is denied.

Figure 6.2: App working properly after restrictions

we can change the parameter of method or we can change body of the method. We can also leave the method unmodified. We have modified the existing modules of Xposed framework to control the methods. Now, if an application wants to access the resource, then we can feed the application with no data of fake data. We have restricted the applications from accessing contacts and messages. Since Xposed module does not revoke or block permissions from an application, so most application will continue to work as before and won't force close or crash as shown in Figure 6.2. Now, we can also control the applications at API level. We can also restrict the application from pasting the data from clipboard either manually or automatically.

6.2 Implementation

Android provides different layers of software stack to handle different things. Currently, if an application invokes any methods, it is handled by Binder IPC module. Binder IPC module binds the method with native libraries. Interaction between apps, application framework, and linux kernel on Android is shown in Figure 6.3.

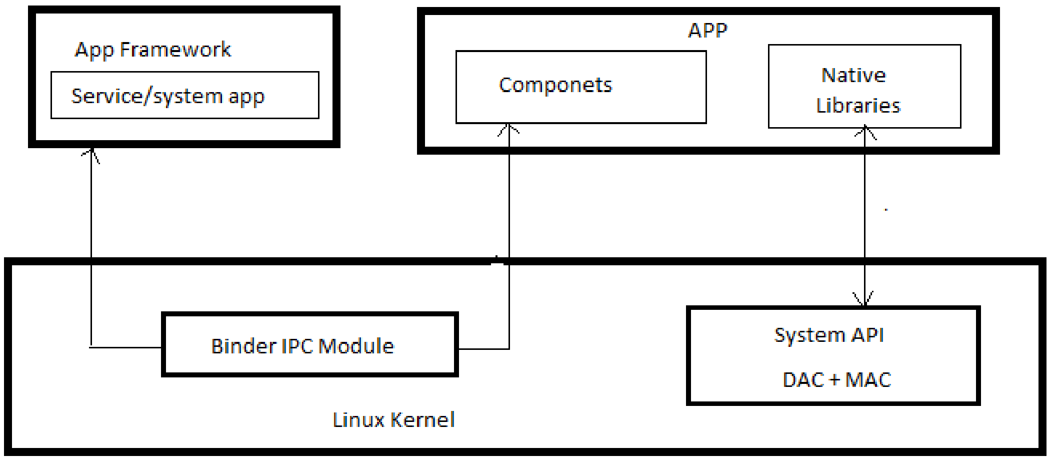


Figure 6.3: Interaction between apps, application framework, and Linux kernel on Android

Xposed framework modifies the *app_process* file of */sytem/bin* and adds an extra jar file to Android to intercept the methods. Since, it modifies the *app_process* and every other process is child process of *app_process*, so Xposed becomes the integral part of the every methods. Workflow of Xposed framework is shown in Figure 6.4.

Xposed framework provides mechanism to intercept method calls of any application. Xposed provides `handleHookedMethod` to handle the methods call of applications. Using this method we can modify the parameters of method, or body of methods. It also provides two more methods namely `beforeHookedMethod` and `afterHookedMethod`. We can modify this method to achieve the desired results. We have modified the existing modules of exposed framework to restrict different permissions. We have modified it such that it gives the fake data or no data to the applications, so applications work properly.

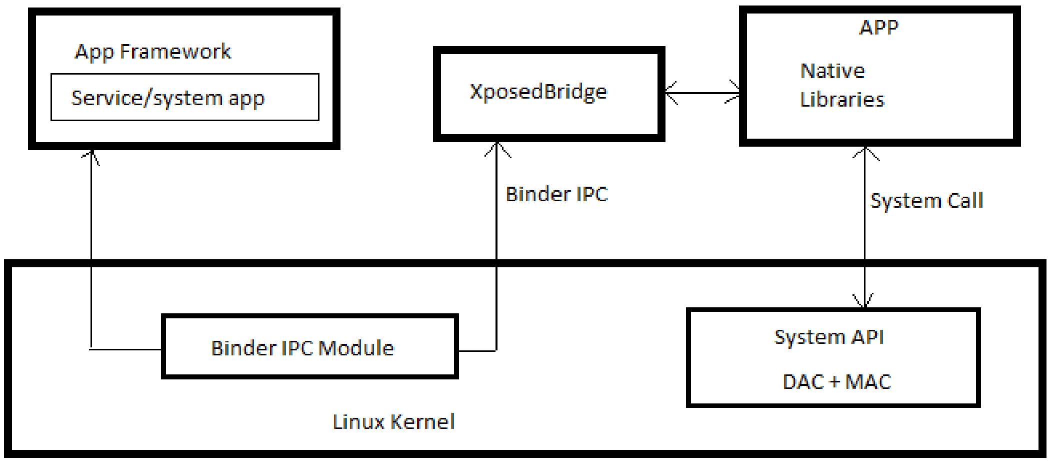


Figure 6.4: Interaction between apps, application framework, and Linux kernel on Android

6.3 Case Study

We have analyzed 10 applications. After analyzing, we have categorized the applications into classes namely benign and malicious. Following applications have been labelled as the benign:

- Budget Planner
- BHIM Making India Cashless
- Tez: A new payment app by Google
- Google Calendar

Since, these applications are benign we do not have to restrict any permissions of the applications. Following applications have been labelled as the malicious:

- Funnyys
- Omingo
- System Certificate
- MMS Beline
- Laughtter
- Android Framework

Since, these applications are malicious we have restricted its malicious behaviour by wrapping it. Permissions which are causing malicious behaviour is denied.

Funnyys

Funnys is an online game applications, so it does not require permissions to access message, phone and camera. So we have restricted the following permissions:

- `android.permission.SEND_SMS`
- `android.permission.RECEIVE_SMS`
- `android.permission.WRITE_SMS`
- `android.permission.READ_SMS`
- `android.permission.CAMERA`

Omingo

This app lets hackers control your device, giving them unauthorized access to your data. To prevent the malicious behaviour of this application, we have restricted following permissions:

- `android.permission.RECEIVE_SMS`
- `android.permission.SEND_SMS`
- `android.permission.WRITE_APN_SETTINGS`
- `android.permission.CLEAR_APP_CACHE`
- `android.permission.READ_SMS`
- `android.permission.RECEIVE_WAP_PUSH`
- `android.permission.INSTALL_PACKAGES`
- `android.permission.CLEAR_APP_USER_DATA`
- `android.permission.MOUNT_UNMOUNT_FILESYSTEMS`
- `android.permission.RECEIVE_BOOT_COMPLETED`
- `android.permission.DELETE_CACHE_FILES`
- `android.permission.WRITE_EXTERNAL_STORAGE`
- `android.permission.REBOOT`
- `android.permission.RESTART_PACKAGES`
- `android.permission.DELETE_PACKAGES`

System Certificate

System Certificate is a fake application which can damage your device and steal your data. To stop its malicious intent, we have restricted the following permissions:

- `android.permission.INTERNET`

MMS Beline

MMS Beline is a third party application which can increase your mobile bill by sending message and by calling to the premium number. This application is installed by another malicious applications. It runs in background. We have to uninstall this application because this application is of no use.

Lughtter

It is another application which can add charges to your mobile bill by sending costly SMS message without informing you first. So we have restricted the following permissions:

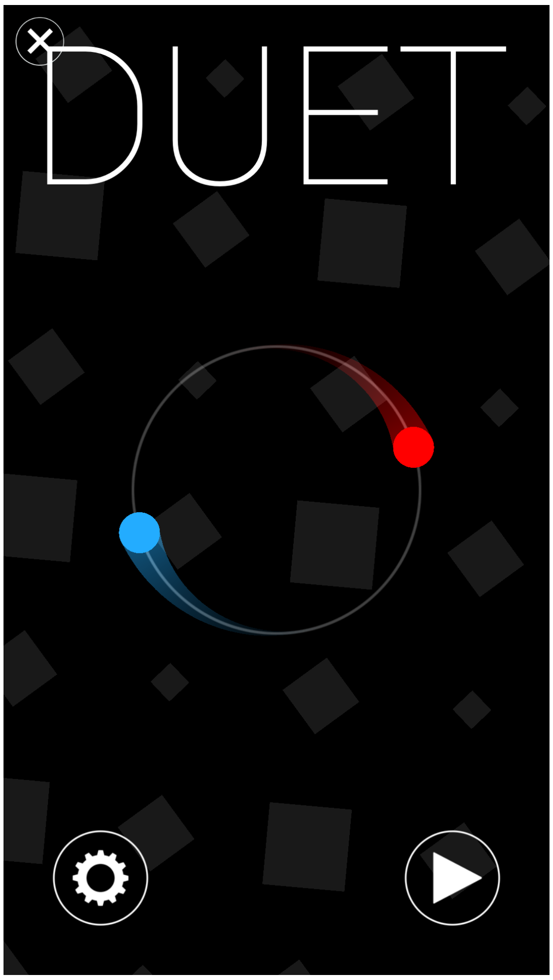
- `android.permission.SEND_SMS`
- `android.permission.RECEIVE_SMS`
- `android.permission.WRITE_SMS`
- `android.permission.READ_SMS`
- `android.permission.CAMERA`
- `android.permission.READ_PHONE_STATE`

Android Framework

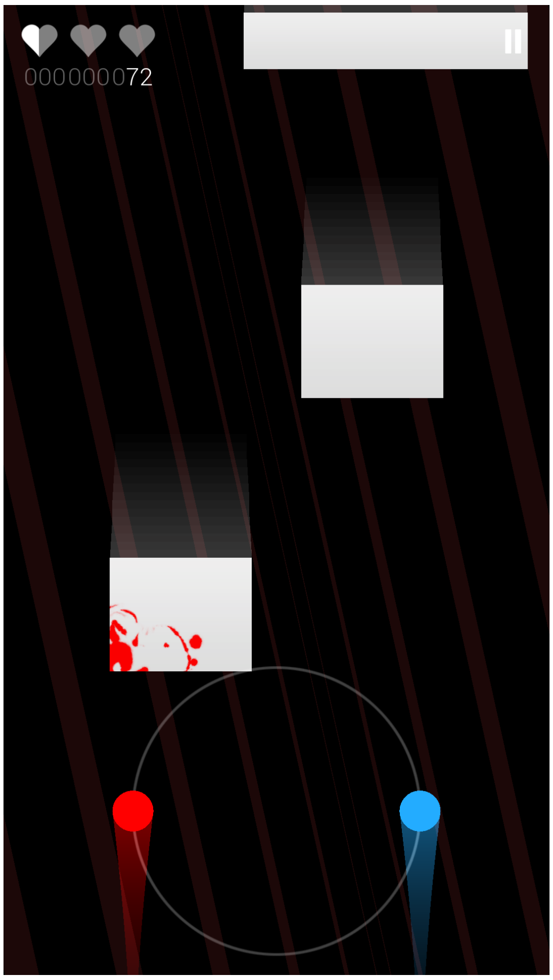
Android Framework is a third party application which can increase your mobile bill by sending message and by calling to the premium number. It can install other malicious applications to your device. So we have to uninstall this application.

Duet

Duet is a game. We have restricted access to identification, internet, location, phone, and view. Though, we have restricted some access for application still it is working properly. Figure 6.5 shows the screenshot of application working properly after restrictions.



(a) Screenshot- 1



(b) Screenshot- 2

Figure 6.5: App working properly after restrictions

Chapter 7

Conclusion and Future Work

In this work we have presented 4 different machine learning models that are trained on a well-known permission classification dataset. We could improve the identification and classification on Android apps with higher accuracy than the available current methods. We have presented our approach with running case studies on a few Android apps and presented our tool's accuracy in identification of malicious apps. Our approach involved a mix of static and dynamic analysis such that even the apps that have unused APIs present in the code that can be traversed post classification of the app as benign by the application store. We have restricted the malicious behaviour of application without interrupting its execution.

As an extension of this work, one may automate the process of permission classification of installed apps. Currently, we are manually deciding which permissions are causing malicious behaviour in a particular application. We can integrate our app classification model with a tool which can give the list permissions which are causing the malicious behaviour.

Bibliography

[1] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49(6):259–269.

[2] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*. ACM, pages 15–26.

[3] XDA contributors. 2014. Xposed framework. `https://forum.xda-developers.com/xposed/xposed-installer-versions-changelog-t2714053`.

[4] Thomas Debize. 2012. Androwarn. `https://github.com/maaaaz/androwarn/`.

[5] Android devlopers. 2018. Android runtime. `https://source.android.com/devices/tech/dalvik/`.

[6] Alejandro Martín García. 2018. Andropytool for dynamic analysis and static analysis. `https://github.com/alexMyG/AndroPyTool`.

[7] Google Developers. 2018. Android permissons. `https://developer.android.com/guide/topics/permissions/overview`.

[8] Google Developers. 2018. Android platform architecture. `https://developer.android.com/guide/platform/`.

[9] Takamasa Isohara, Keisuke Takemori, and Ayumu Kubota. 2011. Kernel-based behav-

ior analysis for android malware detection. In *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*. IEEE, pages 1011–1015.

[10] Konrad Jamrozik and Andreas Zeller. 2016. Droidmate: a robust and extensible test generator for android. In *Mobile Software Engineering and Systems (MOBILESoft), 2016 IEEE/ACM International Conference on*. IEEE, pages 293–294.

[11] Ryan Johnson, Zhaohui Wang, Corey Gagnon, and Angelos Stavrou. 2012. Analysis of android applications' permissions. In *Software Security and Reliability Companion (SERE-C), 2012 IEEE Sixth International Conference on*. IEEE, pages 45–46.

[12] Alessio Merlo and Gabriel Claudiu Georgiu. 2017. Riskindroid: Machine learning-based risk analysis on android. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, pages 538–552.

[13] Prof. Zeller. 2014. dataset. `https://www.st.cs.uni-saarland.de/appmining/chabada/`.

[14] Statista. 2018. Available application in google play store. `https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/`.

[15] Statista. 2018. Global market share of smartphone operating system. `https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/`.

[16] Wikipedia contributors. 2018. Artificial neural network — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=846600702`.

[17] Wikipedia contributors. 2018. Confusion matrix — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=844134813`.

[18] Wikipedia contributors. 2018. Dalvik (software) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Dalvik_(software)&oldid=840000078.

[19] Wikipedia contributors. 2018. Decision tree learning — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=845025616.

[20] Wikipedia contributors. 2018. Logistic regression — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Logistic_regression&oldid=844391326.

[21] Wikipedia contributors. 2018. Random forest — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Random_forest&oldid=846743962.

[22] Wikipedia contributors. 2018. Statistical classification — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Statistical_classification&oldid=845256559.

[23] Wikipedia contributors. 2018. Support vector machine — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=842541213.

Acknowledgements

I am thankful to my supervisor **Prof. R.K. Shyamasundar** for his enormous support and insightful suggestions throughout my project. I am grateful to him for providing me his code base support which was very useful in carrying out my experiments. His insightful suggestions to the various problems that I faced during my project. were not only useful but also helped me in broadening my basic understanding of project area. I am also thankful to **Dr. Vishwas Patil** for his enormous support and suggestion throughout my project.

Signature: Dharmendra K

Dharmendra Kumar

163050032

Date: 30 June 2018