University of Tartu

Institute of Computer Science

Malicious Android application for security testing

Author: Jamil Gurbanzade

Master's Thesis (30 ECTS)

Supervisor: Alo Peets

Tartu 2021

Contents

Foreword

As the world moves into the digital age, generating a vast amount of data and born-digital content, there will be a greater need to educate people about digital literacy. It is my personal passion not only to learn but also to teach other people to cope with difficulties around them concerning technological advancements. Undoubtedly, I could not achieve my current level of success without a strong group of individuals.

First of all, my parents supported me with a huge passion and understanding. I have to mention one person specifically, my close relative Ulviya Teymurkhanova, a great mathematician. It would be unfair if I did not mention her moral support during my study abroad years. And secondly, my supervisor has provided patient advice and guidance throughout the research process.

Finally, I would like to deliver my sincere gratitude to the University of Tartu's administration and dear lecturers of the Institute of Computer Science. Thank you all for your unwavering support.

Malicious Android app for security testing

Abstract

The growing popularity of mobile device artifacts brings data security issues with themselves. Sensitive personal data on mobile phones have already become an open target for malicious third-party developers. The lack of digital literacy of the users enables not only the spiteful developers but also companies to acquire personal data through application permissions. This study investigates the implementation and designing of a prototype Android application to discover what kind of user data is possible to get by application permissions. The prototype application obtains private user data such as contact names, contact numbers, SMS messages, last location coordinates, email account ID, last picture and file (optional), calendar events, and lastly, call logs. Users will see the collected data on a webpage. This thesis's outcome can be used as a good sample of demonstrate permission issues and improve consumers' digital security awareness.

Keywords:

Android application, android permissions, personal data, data collection, digital literacy awareness

CERCS: P170

Pahatahtlik androidi rakendus turvalisuse testimiseks

Nutiseadmete kasvav populaarsus toob endaga kaasa täiendavad andmeturbe probleemid. Mobiiltelefonide tundlikest isikuandmetest on hetkel saanud pahatahtlike kolmandate osapoolte arendajate avatud sihtmärk. Kasutajate digitaalse kirjaoskuse puudumine võimaldab lisaks pahatahtlikele arendajatele ka korporatsioonidel omandada tundlikke isikuandmeid rakenduse

lubade kaudu. Lõputöös käigus arendatakse illustreeriva Androidi rakenduse prototüüp, et teada saada, millistele kasutaja andmetele on rakenduse õigustega katsetades ligipääs võimalik saada. Valminud rakendus hangib isiklikke kasutajaandmeid, nagu kontaktinimed, kontaktnumbrid, SMS-sõnumid, viimase asukoha koordinaadid, e-posti konto ID, viimane pilt ja fail (valikuline), kalendrisündmused ja viimasena kõnelogid. Tarkvara laeb andmed ka internetti ja programmi kasutajad näevad kogutud andmeid veebilehel. Selle lõputöö tulemusi saab kasutada illustreerimaks Android rakenduste turvalubadega seotud probleeme ja parandada tarbijate teadlikkust digitaalsest turvalisusest nutiseadmetes.
Võtmesõnad: Androidi rakendus, androidi õigused, isikuandmed, andmete kogumine, digitaalne kirjaoskus
CERCS: P170

1 Introduction

Mobile phones are considered essential gadgets. People of various ages have steadily used mobile devices for different needs over the past years. Due to its variety of devices and rich features, Android is very popular among users [1]. Concerns regarding the reliability of the Android security system and the protection and privacy of users are growing. The Android security system uses a permission-based security mode that protects access to user data such as call logs, pictures and audio files, location coordinates and any private data on Android devices [2]. This ensures that each app must ask permission to access sensitive data.

Developers should define which permissions they need to apply in their application. The Android smartphone user will have the ability to see what the applications request to access and have the right to provide the app permission to collect personal data and all other functionality of the device. While the user provides permission, Android allows or refuses the use of particular tools according to the user's permission [3]. Android permission model is designed to enhance the security framework by Android users. This can be achieved by reminding to the user about the security ramifications of access permissions.

The first chapter of the thesis describes the history and evolution of the Android OS. Then significant components of Android OS is explained. The first chapter finalizes with the description of segments of Android applications. The second chapter of the thesis is about personal data and GDPR statements related to user data privacy. The next chapter is dedicated to the Android permissions, evolutions process, and categories of the permissions. Besides that, the latest updates in the permission system are pointed out as well.

The fifth chapter belongs to the perception of app permissions from the users and developers. The chapter suggests and explains app permission usage, and configuration is presented for both parties to avoid mistakes and misleads.

The sixth chapter is devoted to the whole thesis project architecture. The chapter defines what and how the project is made of. Moreover, the technical explanation of components of web application and Android application is mentioned as well. Furthermore, the Android app workflow, which means the steps of the app working process is stated. The last chapter is about the implementation of the Android application. The methods which collect user data are clarified and illustrated.

1.1 Problem Statement

Despite current technological advancement, the digital literacy of users with regard to the security of their digital identity is still very limited. Even though years have been passed, the leakage of general knowledge is still observed among smartphone users. Before downloading a mobile app from the Google Play Store, the application notifies the user about the terms and other further requested permissions. Predominantly, smartphone users grant all the permissions that the application demands. However, most of those permissions are undoubtedly not related to the primary usage of the application. For instance, the user wants to crop pictures, make a collage, and add some effects to it. Nonetheless, the downloaded photo crop tool asks permission to get access to record audio or obtain call logs. Obviously, this is a very suspicious case and that may lead to the stealing of user's confidential data.

Even though Android permission model is considered robust,however, this model rather attractive for malicious developers and leverages the limitations of the model [4]. Regardless of this, countless user data have been collected and accessed by a vast range of mobile applications.

In the Figure 1, the results of statistics of conducted mobile threat by Kaspersky was depicted. According to the graphics, 1,245,894 malicious installers detected, an increase of 93,232 over the previous quarter[5].

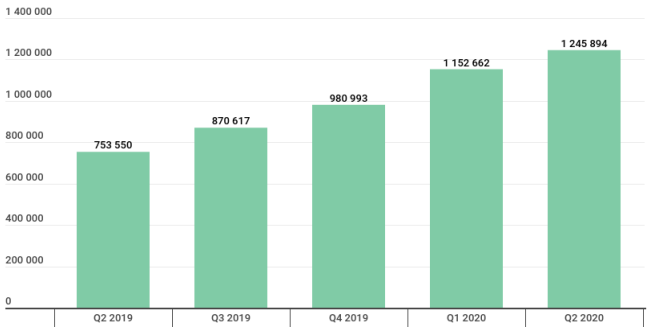

Figure 1. Mobile threat statistics by Kaspersky for 2019-2020 [5]

1.2 Related works

The configuration period and the runtime are used in the implementation of the permission system.While installing an application, the user of an Android device grants necessary permissions and then system utilize the monitor to detect if the application possesses a certain API (Application Program Interface) at the same time as program execution [6]. If user has desire to utilize features and functionalities of the third-party Android software, permissions must be accepted by the user before the application is used. Android application package installer will terminate the installation process whether the user refuses to obtain mentioned permission on the dialog box [7].Users require a more versatile permisson system to ensure that diverse services are accessed and secured by users.

Many analysis were done in order to investigate the user comprehension over Android permissions. Some applications were developed by Greenwood et.all [8] to inspect the understanding of permissions while application is installed. It is used to assess if the Android software's permission list is accurate when the application is installed by the end user. Consequences of the research illustrated that permissions are not supportive enough to assist users to make security choises. The user will not be able to obtain clear insights from the poorly described the explanation of permission.

Ryan et.all developed a framework for malware detection that notifies hazardous level of combination of permissions [9]. These sort of permissions are known as runtime permissions, also known as dangerous permissions [10]. If application consists set of dangerous level permissions such as READ_CONTACTS, RECORD_AUDIO, READ_PHONE_NUMBERS it can be almost a potential malicous software that intends to monitor user's sensitive data. The application could be malicous if attemps to get location coordinates such as ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION as well.

Dynamic analysis platform called VetDroid [11] implemented by Jingzheng et.all to observe permission usage analysis on Android appications. Through

the app it is able to monitor how applications reach system resource via granted permissions after installation. Such reports have shown that permissions can lead to the potential security issues. Other similiar research have been conducted by Bilal Abdullah [11-1] mentions that performing speech recognition analysis on gathered by granting microphone access. Another work by Jekaterina Gorohhova [11-2] is dedicated to the private user data collection by granting dangerous level permissions via malicious app and forwarding those data to the email address of the user.

1.3 The goal

The main goal of this master thesis is to study what kind of data can be accessed on the Android application via granted permissions. For the abovementioned purposes a sample application is developed and designed which depicts what sort of data can be collected if user grants some specific permissions. Therefore, it will inform user about the consequences of how malicious software can reach the sensitive user data and explain techniques how to ensure protection of data. Collected user data will be demonstrated on the webpage.

2 Background of Android

2.1 History of Android OS versions

The Android Open Source Project (AOSP) was developed and led by Google after the acquisition of Android, Inc. The Android OS update history started with the introduction of Android 1.0, released in September 2008 [12]. Operating systems from Google are often named after a cookie [13]. 2019 release of Android platform versions with cumulative distribution in percentages and API levels is illlustrated in the Figure 2.

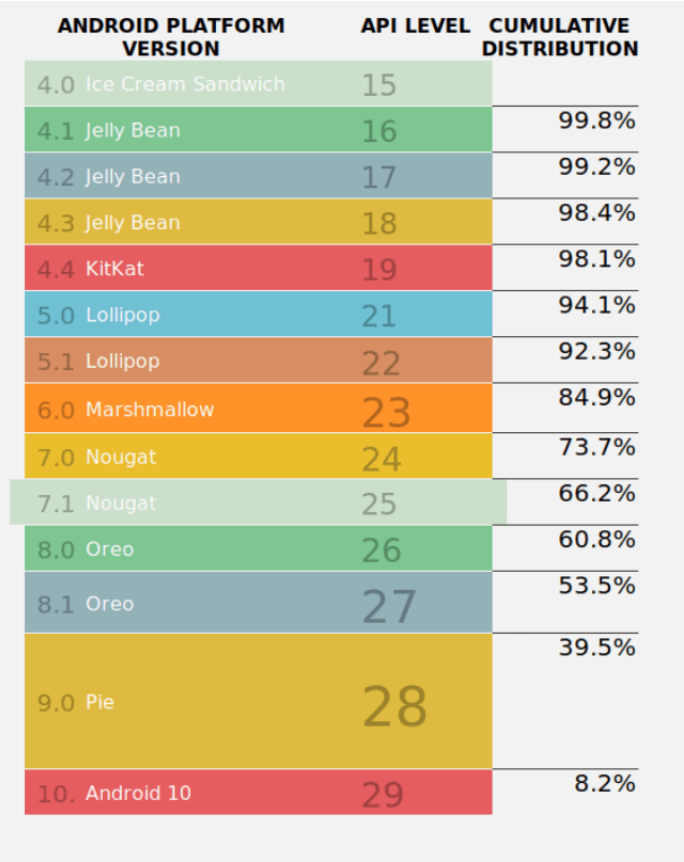

Figure 2. Cumulative distribution of Android versions [13]

The very first version of Android OS released in November 2007 and it did not have an official name, however, some researchers call it Astro. Because that version came up with a earlier smartphone brand called HTC Dream Astro [14]. The first Android tablet version was launched on the Motorola Xoom, named as 'Honeycomb', in February 2011. Honeycomb particularly was developed for tablets to enjoy wider and larger screens. Most of Android versions are already deprecated thereby Google announced that all versions till "Nougat" 7.x.x will not be supported[15].

2.2 Architecture of Android

Android is a Linux operating system-based open source mobile application platform [16]. The architecture of Android system represents software stack which includes four major layers (from top to bottom):

- Applications layer
- Application framework
- Libraries
- Linux kernel

In the Figure 3 the four layers of Android architecture with sample components is visualised [17]:

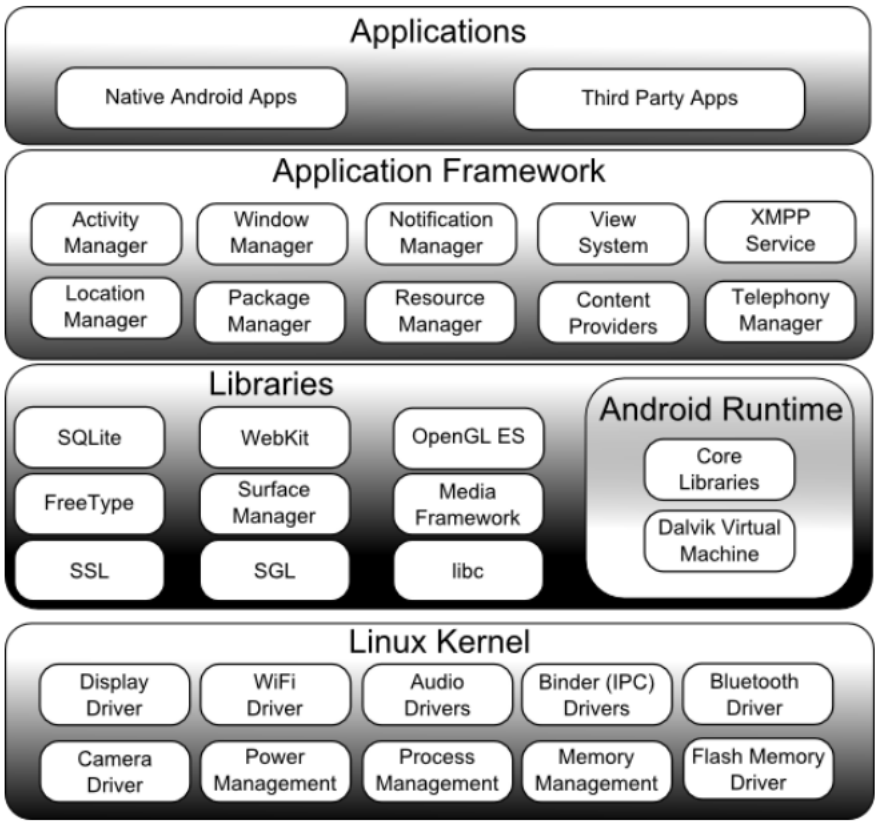

Figure 3. Architecture of Android system [17]

Linux kernel and runtime - Linux kernel is situated at the very bottom layer which provides abstraction between hardware and higher levels. Linux allows several processes to run simultaneously in the multitasking environment. Back in the earlier years of Android, apps used to run on seperate instances of DVM (Dalvik Virtual Machine) without interfering each other. Each application is sandboxed which means those can not interrrupt each other and have access to the hardware resources. Android used the idea of user ID that allows kernel to store applications, limit their access to the hardware or other services. Thus, by design, the data and resources of each app located in location that can only be utilized by the app and core (Figure 4)[18].

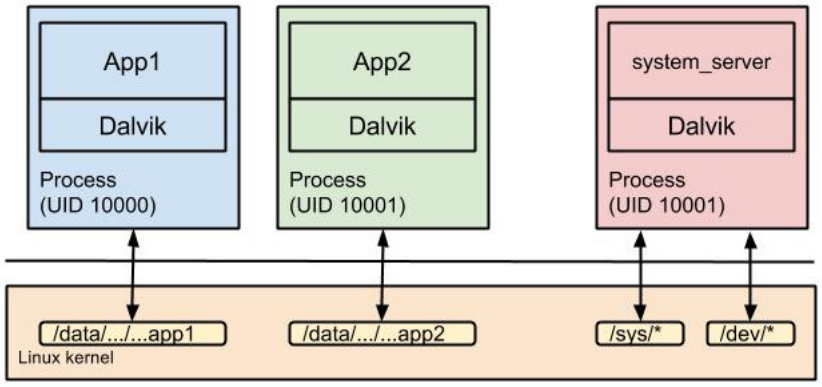

Figure 4. Application processing in Sandbox [18]

After Android Lollipop (5.x.x), Android Runtime (ART) take over the place of Dalvik Virtual Machine. In comparison to Dalvik method, ART uses AOT approach (ahead-of-time) to compile whole program in native machine code druing booting process or installation. In the Figure 5 Android Runtime flow is depicted.

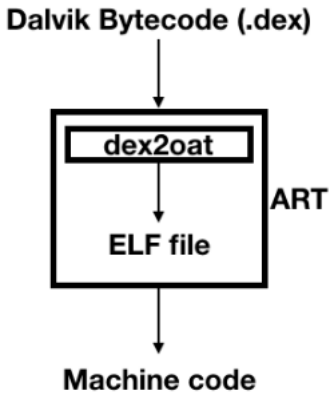

Figure 5. The flow of Android Runtime [19]

.class files inside APK file (Application Package) are converted into Dalvik bytecode (.dex file) through Dex compiler. Nevertheless, Dalvik compiled bytecode during execution. Because of AOT, the dex files are being compiled beforehand[19].

Absolutely, runtime approach has a lot of advantages in terms of:

- Robust batttery performance - Unlike DVM, runtime does not compile code everytime and it significantly improves battery life.
- App launch time - Because of AOT, runtime has the native code ready to launch immediately[19].

Application framework - is the box of services which launches the ecosystem that Android apps run and regulated. These are collaborative services. The essential concept of the application framework is to develop application from reusable and interchangeable segments. There are a lot of main services or components in the application framework such as Window Manager, Activity Manager, Content Providers and etc (Figure 2).

- Activity Manager - manages set of myriad application lifecycle states such as *startingg,running,paused,stopped and destroyed*
- Window Manager **-** adjusts the ratio of the screen for the application
- Notification Manager - is responsible to notify happening events to the user. It has various types such as icon in the status bar, flashing LED's, playing a sound or vibrating [20]

Native libraries layer - Native libraries layer assists to manage various types of data. Native Development Kit (NDK) allows developers use the code that written in C and C++ together with Android. Within NDK tools, developers may gain access to control the physical components of the phone such as working with device sensors [21]. It is necessary to mention some open-source libraries:

- **SQlite** - it is a relational internal database storage to store data in the device. Database is available for all the applications on the phone.
- **Webkit** - A browser engine that allows to show the web content inside an app.
- **OpenGL ES** - The library that enables to render 2D and 3D graphics.
- **Surface Manager** - A component that sets up a windows for the screen.

Applications are placed on the top of the Android stack. This level consists of 2 application categories:

- Third party applications which can be downloaded from Google Play or other web sources. The author of those apps are third-party developers. Malicious applications that have felonious goals are also included in third-party app category.
- Built-in or pre-installed apps that comes with the device. These applications may be developed and configured exclusively by system type, depending on the needs of specific manufacturers.

2.3 The components of Android applications

Components are used to develop an Android software. This is a segment that possesses special functionality and with the proper permissions could be initialized by other application.There are 4 major types of components available [22]:

- **Activity**
- **Services**
- **Content providers**

- **Broadcast receiver**

Activity component originally is the interface of the application. The inital screen that observed by user when the application is run is the Main Activity. In order to initalize a new specific activity, ***<activity> tag*** has to be declared in **Android manifest file** in the application package (manifest.xml). So that,if other apps want to start the same activty, need to declare corresponding ***<uses-permission> tag*** in their manifest file. Two normal permissions (INTERNET and ACCESS_NETWORK_STATE) were declared at the top of the manifest file [23] in the Figure 6.

```
<uses-permission android:title="android.permission.INTERNET" />
<uses-permission android:title="android.permission.ACCESS_NETWORK_STATE" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.Holo.Light">
    <activity
        android:label="@string/app_name"
        android:title=".MainActivity">
        <intent-filter>
            <action android:title="android.intent.action.MAIN" />

            <category android:title="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity
        android:label="@string/title_activity_settings"
        android:title=".SettingsActivity"></activity>
</application>

</manifest>
```

Figure 6. Declared normal permissions on the manifest file [24]

Service component performs operation that has long execution time in the background without depicting on the user interface [25]. Multiple operations could be performed within services such as network transactions, playing audio via music player, communication with content providers etc. There exist 3 types of services:

- **Foreground**

- **Background**
- **Bound**

Foreground - It is a sort of service that performs task which can be noticable by user. Notification icon can be seen and it is a must to display on the bar while foreground service executes. Audio player application is a good example for this case.
Background - A task that the user does not explicitly observe is done by the background service. An application for compacting storage is considered a background service [26].
Bound - it is implemented in mostly client-server interface which allows perform IPC[1]. It is possible to bind multiple application components or activities to the service [26].

Content provider - content providers maintain data sharing among activities and applications. A content provider introduces needed data for external applications which looks identical data structure in the standart database tables. Data encapsulation process and data protection measures is ensured by content providers as well [27]. **android.content** package has classes that provides data accessing and broadcasting. The accessed data to external apps is displayed in a form of a file or a table by a content provider. The data can be stored in a SQLite data database or in other permanent storage [28]. In the figure 6, content provider data accession schema is illustrated [27]

[1]interprocess communication

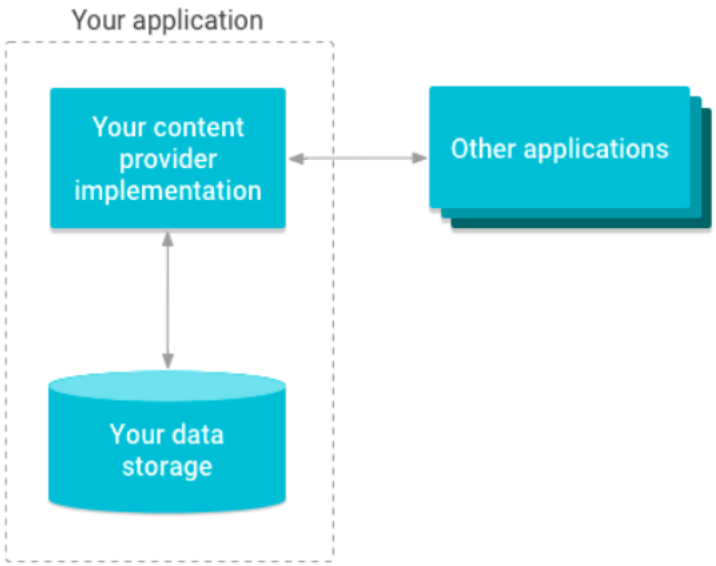

Figure 7. Content provider data accession [28]

Mostly content providers are considered the reasonable asbtraction to share data to other applications. One of the major advantageous of content provider is, it does not have an impact the data storage implementation. One detailed explanation about data storage migration to different sources was presented in official documentation of Android Developers Guide manual [29]. It can be seen in the below illustration that other applications retrieve data through developed content provider implementation from alternative storage:

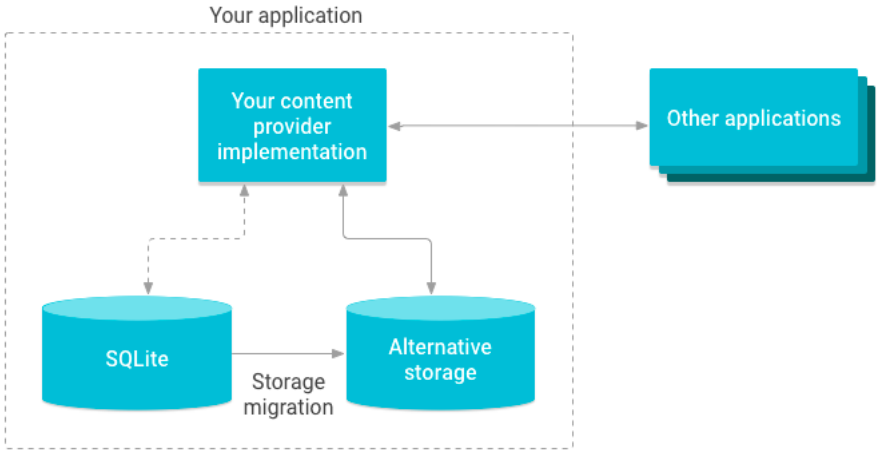

Figure 8. Content provider migration depiction [29]

In order to access data in content provider **ContentResolver object** has to be used in application. This object as client communicates with content provider and accepts data requests from clients. After that it performs the action demanded and returns the data. The flow for accessing data storage via content provider is presented blow picture (Figure 9):

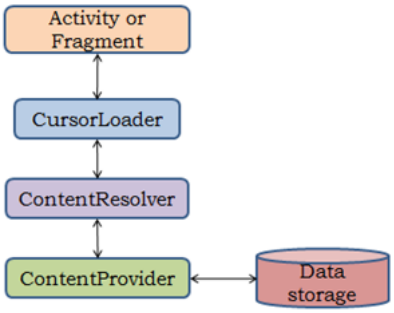

Figure 9. Content provider data retrieval from data storage [30]

As it can be seen from the picture, CursorLoader is called from user activity and then ContentResolver query is run. ContentResolver get requests from users and channel them to the content provider. User dictionary

data structure is by default data structure in the Android system. In order to retrieve data from provider user permission has to be defined with corresponding <***uses-permission***> tag in the manifest file. CRUD (create, read, update, delete) methods contains in the content provider class corresponds to the abstract methods such as insert, query, update, and delete. A sample of ContentResolver query is depicted below figure [31] (Figure 10):

```
cursor = contentResolver.query(
        UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
        projection,                         // The columns to return for each row
        selectionClause,                    // Selection criteria
        selectionArgs.toTypedArray(),       // Selection criteria
        sortOrder                           // The sort order for the returned rows
)
```

Figure 10. A sample of ContentResolver query [31]

The arguments of query matches an SQL SELECT parametres. The ContentResolver query has similarities with ordinary SQL queries. Below described the comparison of SQL query and ContentResolver query keywords:

- *Uri* - Content URI is the path of data table that provides address for the source. It is similar to ***FROM*** keyword in the SQL statement.
- *projection* - This parameter is a set of columns to be used in each retrieved row. It is similar to name of columns in the SQL statement.
- *selection* - it defines specific criteria for chosen rows. It is similar to ***WHERE*** parameter in the SQL statement.
- *selectionClause* - there is no equivalent of this keyword.
- *sortOrder* - it can be noticed from the paramter name that returns data in a particular order. It is similar to ***ORDER BY*** parameter in the SQL statement.

Broadcast receiver - sometimes called just as receiver, is a component that allows developers to register for the app events. It can be referred a messaging system accross the application. Applications can send or receive messages which known as broadcast messages from the system itself or another Android application. Broadcasts messages are similar to publish-subscribe design pattern. They are sent when an event of interest occurs. Applications can send custom broadcasts to notify other apps of something that they might be interested in (for example, some data has been downloaded)[32].
Broadcast messages are wrapped in an **Intent** object. The **action string** identifies the event that ocurred such as android.intent.action.AIRPLANE_MODE, android.intent.action.BATTERY_LOW etc.
There are two types of broadcast receivers:

- Manifest declared
- Context declared

Manifest declared broadcast recievers sometimes called static receivers which are declared in the manifest file and it operates even though the application closed. After API level 26, additional constraints on manifested declared receivers are imposed on the system thus, many of the broadcast messages can only be Second type of receivers sometimes referred as dynamic receivers that works only if application is active or minimized.

3 Personal data and regulations

3.1 Data sensitivity

Personal data is any information about a person that is associated with him, allows him to be fully identified, to obtain other information about him, to commit any encroachments on the secret of his personal life or property [33]. Android applications requires permissions to access system features to operate properly. Obviously, not all the users are sophisticated enough to understand the privacy of their personal data. As a result, data theft happens because of the negligency of the end users and developers through applications acquire personal data.
Generally, target groups who are at risk of loosing personal data can be classifed who :

- owns bank cards
- receiving medical services
- has pension savings
- registered on a government services
- has mortgages

Thus it is so important to have a personal data security policy.

3.2 GDPR and data processing

The GDPR or General Data Protection Regulation is an Europan Union regulation on data protection and data privacy that applies to all data processing done by organizations and institutions operating in the EU and outside of the EU if they are processing personal information of the citizens or residents of the EU [33]. Estonian Personal Data Protection Act[34] regards

personal data as information relating to an identified or identifiable natural person. Data processing stated as several operations on user data such as collection, retrieval, recording, transmission, destruction etc. regardless which procedure is carried out [34].

4 Android permissions

4.1 Permission evolution

Android applications are designed to carry out a series of operations, some of which require user permissions. In comparison to the iOS platform, Android came little late in using app permissions. The earlier Android versions asked for specific user permissions while downloading the application from the Google Play Store. Hence, the developers who built an application used to submit the needed user permissions to the Google Play Store. Like every other mechanism, this mechanism had upside and downside. The advantageous part was the user would observe all the required permissions at once and beforehand. The user did not see the required permission during the runtime of the app. The disadvantageous side is if there were certain permission which the user did not want to grant, the user would not install that app. At the same time, companies and developers initiated to gather data by adding suspicious permissions to the app, and the user community did not welcome that. Therefore, this procedure was common back in days before releasing Android M (Marshmallow) [35]. The below figure illustrates the earlier permission asking (Figure 11).

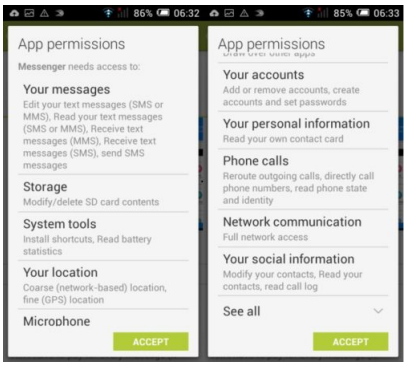

Figure 11. Old Android permission asking mechanism [36]

With the introduction of Android Marshmallow, Google has launched runtime permissions for Android, improving the permissions environment for

the better. In this case, developers do not obtain all the permissions during the downloading process. The permission will not be asked until the feature of the app is revoked. The user can decline runtime permission and app will explain the necessity of asked permission. In case of denial, permission will be pemanently denied and could only be activated from the settings. User flow of runtime permission process is graphically described at the below chart[37].

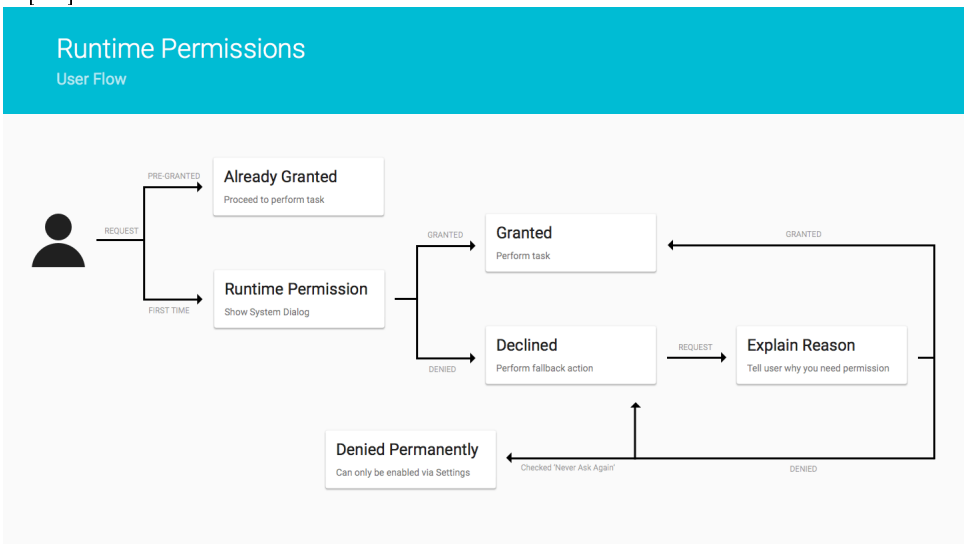

Figure 12. User-flow of runtime permissions [37]

4.2 Permission types

Android classifies permissions of various forms, including install-time permissions, runtime permissions, and special permissions. Two essential entities such as restricted data and restricted actions is protected by user permissions. Each type of permission identifies the level of the restricted data that app can access, and the scope of the restricted actions that app can execute while the device gives the permission to the app [38].

- *Install-time permissions* - provide limited access to private user data to execute restricted actions and it has minimal effect to the app and

the system. When app developer declares install-time permissions in the application, the system will grant it automatically without asking the user.

- *Normal permissions* - are categorized as a subtype of install-time permissions. Normal permissons provide access to the data which is outside of sandbox with less risk.

- *Dangerous permissions* - sometimes referred as runtime permissions, give access to restricted user data and performs restricted actions which might significantly affect to system and other applications in the device. When application requests dangerous permission, promt dialog box (Figure 13) appears to ask user consent. Senstive user data such as calendar events, location coordinates is accessed through runtime permissions.

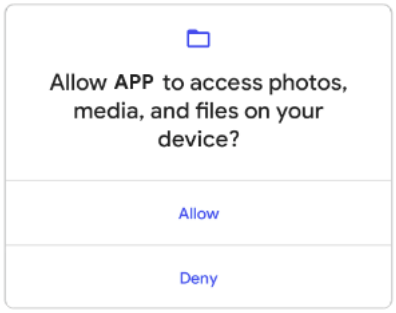

Figure 13. Dangerous permission promt [38]

- *Signature permissions* - known as subtype of install-time permissions. System grants signature permissions during installation period. The app that demands permission must be signed with the same signature as the app that specifies the necessary permission [38].

In the latest version of Android (API level 30) whenever application requests location or some related dangerous permissions such as access to camera or

microphone, new sort of option dialog box appears. This option is named as ***Only this time option*** which means application is only able to the access data while app in use. When application is not used, the system will not allow app to access data until the next usage [39].

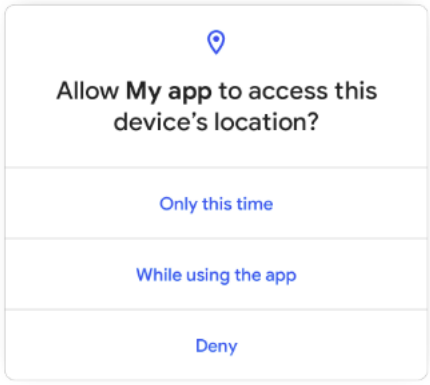

Figure 14. One-time permission dialog [39]

The research paper [40] about the analysis of the Android permissions system depicts considerable growth in the number permissions.

Table 1. Total number of permissions per each API release [40]

Release Code Name	Android Platform Version	API Level	Dangerous	Normal	Signature	Signature orSystem	Total Permissions
Android 11 Beta	11	30	30	47	48	42	167
Q	10	29	30	45	42	41	158
PI	9	28	27	42	38	41	148
Oreo	8.0–8.1	26 - 27	26	39	38	41	144
Nougat	7.0 – 7.1	24 - 25	24	35	35	41	135
Marshmallow	6	23	24	35	32	40	131
Lollipop	5.0 – 5.1	21 - 22	24	32	24	40	120
KitKat Watch	4.4W	20	24	32	18	40	113
KitKat	4.4	19	23	32	18	40	112
Jelly Bean	4.1 – 4.3.1	16 - 18	23	29	16	36	104
Ice Cream Sandwich	4.0.1- 4.0.4	14-15	20	29	14	35	98
Honeycomb	3.0.x – 3.2	11-13	19	29	12	35	95
Gingerbread	2.3- 2.3.5	9-10	19	29	11	35	94
Froyo	2.2.x	8	18	27	11	35	87
Eclair	2.0 -2.1	5-7	18	26	9	33	86
Donut	1.6	4	18	26	9	32	85
Cupcake	1.5	3	17	25	8	31	81
Base	1 -1.1	1-2	17	24	7	27	73

As it can be seen from the Table 1, since the first release of Android the number of different permission categories significantly increased. The latest version of Android already has total 167 permissions. The number of dangerous and normal permissions almost doubled and particularly signature permissions grew 6 times since the base version of Android.

5 App permissions from user and developer perspective

5.1 User perspective

Android is the most common OS for mobile devices,therefore at some point cybercriminals will try to take advantage of this. Although Android permissions can be risky, it is possible to prevent troubles by paying attention to certain aspects:

- It is recommended to download applications from the verified stores such as Google Play Store. Because malicious applications is the straightforward way into the device for malware. By injecting vicious code snippet to the app, developer can get a backdoor and data theft is inevitable.
- Checking application description to look which permissions will be asked before downloading app from the store is important. These descriptions don't usually explain why a permission is needed, but they will help user to know what to expect.
- It is also possible to send an email to the developer to get detailed description of application permissions.
- Doing research at Play Store reviews, check-forums and blogs is considered helpful in order to discover more about the security of the applications.

5.2 Developer perspective

Android Developers Guide suggests [41] to follow the blueprint of best practices of app permission declaration while developing an app:

- While developing an app, programmers should avoid unnecessary permissions. Only add permissions that are necessary for the application workflow.

- Mobile programmers have to be careful with the libraries which they work. Some libraries demands permissions in the applications. For instance, if developers eager to conduct an experiment via ads and analytics libraries, system will ask LOCATION permission. However, from user's perspective it looks like this permission comes from the application.

- It is highly recommended to state why the user needs permissions. Being transparent and honest make users feel comfortable. The user's willingness to grant permission depends on a clear and detailed explanation.

- Making system accesses clear is rather needed. It helps users comprehend exactly when application wants to gain access to the private data or execute restricted actions. For example, notifying the user when app begins to use camera or microphone.

6 Technical overview of system and data workflow

6.1 Project architecture

Technical stack of whole thesis project consists of several technologies. Instead of traditional Java, Android application for thesis developed in Kotlin. Kotlin is an open-source programming language, created by JetBrains and utterly interoperable with standart Java libraries [42]. It is a supported language by Google and Google officialy upholds Kotlin for Android development [43]. In the below figure, the architecture of the project is illustrated.

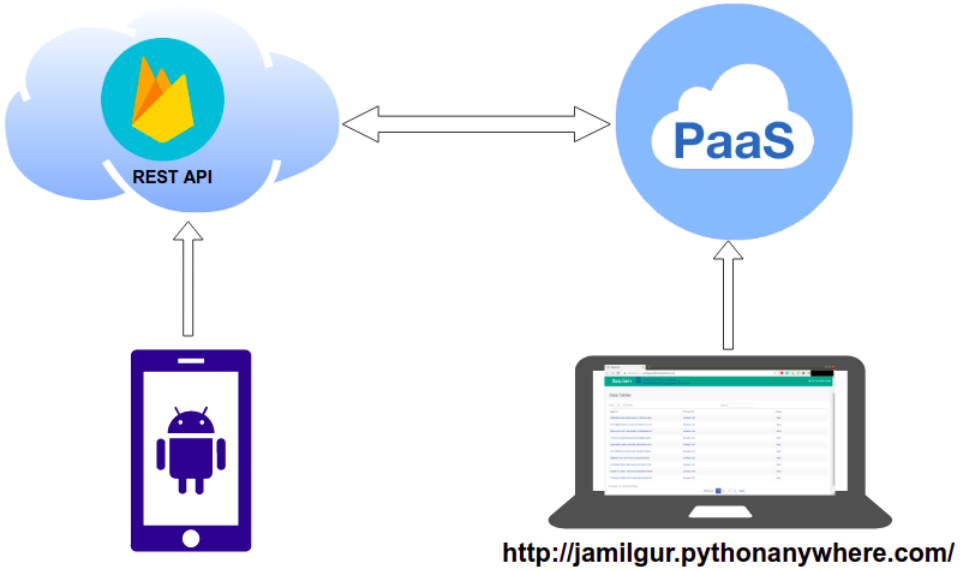

Figure 15. Blueprint of the project

Webpage for demonstration of collected data is created through essential front-end web tools such as Javascript, Jquery and Bootstrap. Bootstrap library makest the webpage responsive for miscellaneous screen sizes. The data table and its components on the webpage is developed by JS DataTable

library. DataTables library enables to build advanced, customized and fully-interactive web tables[44]. As a database Google Firebase Realtime Database is chosen. Google Firebase Realtime Database [45] is a NoSQL cloud-hosted database where all private user data is stored and synchronized. Firebase is a Backend as a Service that presents massive variety of cloud services for mobile and web developers. User data from the database is retrieved while the page is loaded. Back-end part of code is running in Django application. Django is an open-source back-end framework of which follows the model-template-views architectural pattern [46]. Django application for the private user data demonstration is hosted in the "PythonAnywhere" platform. "PythonAnywhere" is one of the PaaS[2] platforms that presents hosting service for Python based web applications [47]. When user executes application on the phone, "userData" reference is created in the database. While pushing buttons, each time user adds data to the JSON tree and it becomes a node with associated key. That associated key is randomly generated unique UID for each session. Collected data that resides in the database is represented as a structured JSON tree shape:

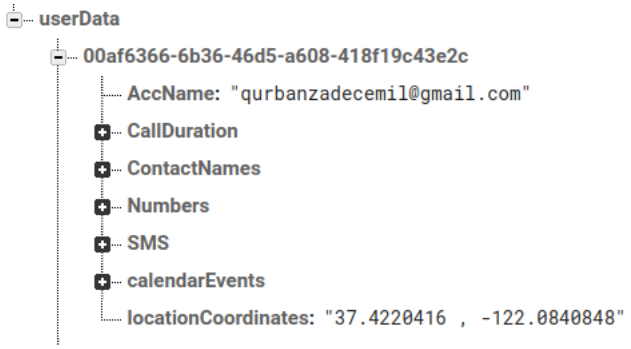

Figure 16. Data structure in the database

[2]Platform as a Service

6.2 Application workflow

The workflow of the prototype application is linear. The workflow is described step by step below:

1. The prototype application is installed.
2. The applications asks for declared permissions in the beginning
3. User presses the buttons on the screen to collect data
4. User clicks textbox to see own UID
5. After pushing buttons, new activity with URL link appears
6. User clicks URL and redirects to the web browser
7. The prototype application finishes its work.

7 Application implementation and performance evaluation

7.1 Application interfaces

The prototype application is named "**SpyJam**". Application will ask to grant following permissions to operate properly:

- READ_SMS
- GET_ACCOUNTS
- READ_PHONE_NUMBERS
- READ_CONTACTS
- READ_CALL_LOG
- ACCESS_FINE_LOCATION
- READ_CALENDAR
- READ_EXTERNAL_STORAGE

After launching application, it is going to demand to grant all asked permissions. User must grant permissions to see the functionalities of the app. Before collecting data, an info table appears (Figure 17) to inform user about the app.

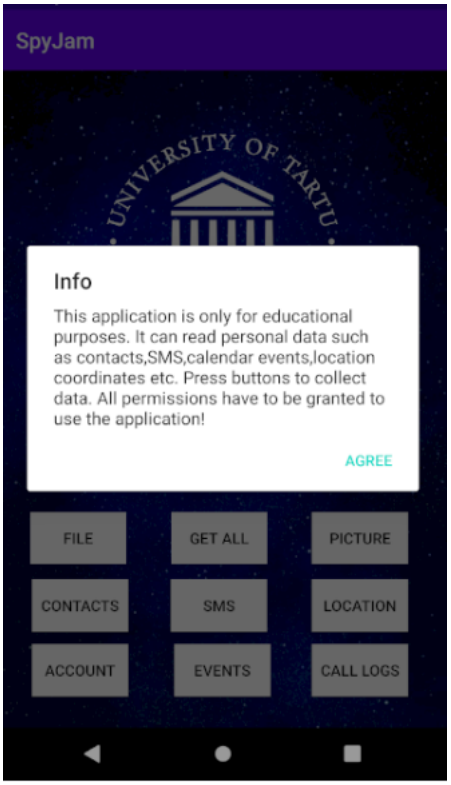

Figure 17. Info screen of the application

After pushing "Agree" button main screen of app appears. The main screen contains 9 buttons and one text box. Each of these buttons has functionality that gathers various data such as contact names and phone numbers, SMS messages, location coordinates, Gmail account names, calendar events, call logs,last downloaded file and last picture respectively.

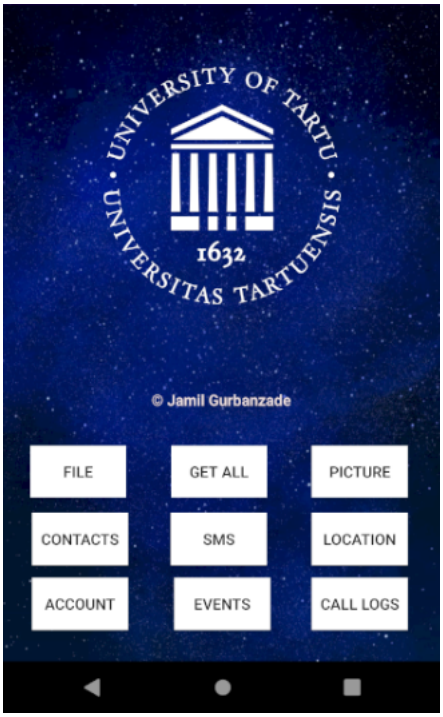

Figure 18. Main screen of the application

The app functionality is located in the **MainActivty.kt** file. Thus, right after pressing buttons, another activity appears (**LinkDemo.kt**) where the user will spot the website's link and the user ID. By clicking to the link, it redirects to the webpage on a browser. Afterwards, user types the UID in the search box, finds its UID then clicks to UID and collected data appears on the right window. User scrolls down to look at the gathered private data. In the below subsections, the methods which gather personal data is explained.

7.2 Obtaining contact details data

SpyJam application asks READ_PHONE_NUMBERS, READ_PHONE_STATE and READ_CONTACTS permissions to get access to the contact list. getContactDetails() method has two cursor object that is created to send query

to the phone database. Collected contact names and contact numbers are appended to the nameList and numberList arraylist, accordingly.

```
val nameCursor : Cursor? = contentResolver.query(
    ContactsContract.Contacts.CONTENT_URI,
     projection: null,
     queryArgs: null,
     cancellationSignal: null
)
while (nameCursor!!.moveToNext() && nameStep!=5){
    val name : String = nameCursor.getString(nameCursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)).toString()
    nameStep++
    nameList.add(name)
}
```

Figure 19. Reaching the user's contact name data

```
while (NumberCursor!!.moveToNext() && numberStep!=5) {
    val numberID : String! = NumberCursor.getString(NumberCursor.getColumnIndex(ContactsContract.Contacts._ID))
    val numberCursor : Cursor? = contentResolver.query(
        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
         projection: null,
         selection: ContactsContract.CommonDataKinds.Phone.CONTACT_ID + " = ?",
        arrayOf<String>(numberID),
         sortOrder: null
    )
    numberStep++
    while (numberCursor!!.moveToNext()){
        val rowID : String! = NumberCursor.getString(NumberCursor.getColumnIndex(ContactsContract.Contacts._ID))
        number = numberCursor.getString(numberCursor.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER)).toString()
        numberList.add(number)
    }
```

Figure 20. Reaching the user's contact numbers data

7.3 Obtaining SMS data

Application asks *READ_SMS* permission to get access to the phone SMS. *content://sms/inbox* is the directory where SMS messages are located and it was given as a selection argument to the query. Through while loop body of SMS message is taken and via getString method it is converted to the string and added to the arraylist.

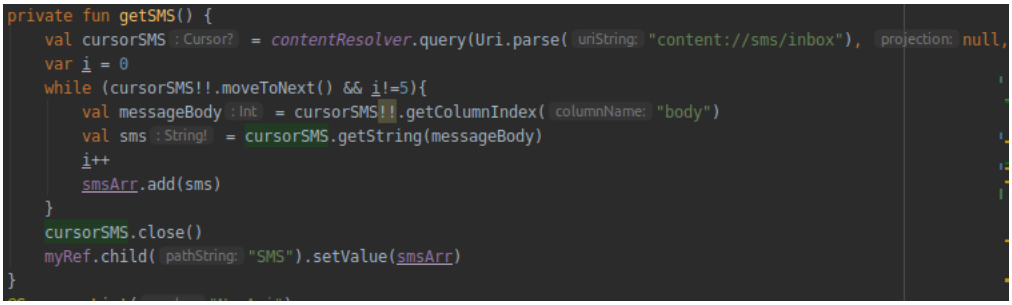

Figure 21. Reaching the user's sms data

7.4 Obtaining user last location data

SpyJam application asks ACCESS_FINE_LOCATION permission to get the last known location of the user. First of all, Fused Location Provider Client instance has to be declared and inside OnCreate function Location Services called. A task is returned by **lastlocation** method and it retrieves the coordinates of a geographic location data such as longitude and latitude.

Figure 22. Reaching the user's location data

7.5 Obtaining user's Google account ID

Application asks GET_ACCOUNTS permission to get the user's account ID. *Account Manager* returns array of accounts and code snippet below retireves the Gmail account ID of the user from the array.

```
fun getAccName(){
    val  manager : AccountManager  = getSystemService(ACCOUNT_SERVICE) as AccountManager
    val list : Array<(out) Account!> = manager.accounts
    if (list.size>0){
        var gmail = "empty"
        for (account : Account! in list){
            if (account.type.equals( other: "com.google", ignoreCase = true)){
                gmail = account.name
                break
            }
        }
        val text = "Uploaded!"
        val duration : Int = Toast.LENGTH_SHORT
        val toast : Unit = Toast.makeText(applicationContext, text, duration).show()
        myRef.child( pathString: "AccName").setValue(gmail)
    }
    else {
        val text = "Uploaded!"
        val duration : Int = Toast.LENGTH_SHORT
        val toast : Unit = Toast.makeText(applicationContext, text, duration).show()
        myRef.child( pathString: "AccName").setValue("empty")
    }
```

Figure 23. Reaching the user's location data

7.6 Obtaining calendar events data

Application asks READ_CALENDAR permission to get the calendar evets of the user. **CalendarContract** class contains **Events** class and *TITLE* is the property of the last one. Besides title, start and ending time, location of the events is also possible to acquire. Collected calendar events added to the arraylist.

```
fun getCalendarEvents(){
    val eventArr = ArrayList<String>()
    val uri : Uri! = CalendarContract.Events.CONTENT_URI;
    val selection : String = CalendarContract.Events.EVENT_LOCATION + " = ? "
    val myProjection : Array<String> = arrayOf(
        "_id",
        CalendarContract.Events.TITLE,
        CalendarContract.Events.DTSTART
    )
    val cursor : Cursor? = contentResolver.query(uri, myProjection, queryArgs: null, cancellationSignal: null)
    var eventSteps = 0
    while (cursor!!.moveToNext() && eventSteps!=10) {
        var title : String! = cursor.getString(cursor.getColumnIndex(CalendarContract.Events.TITLE))
        eventArr.add(title)
        eventSteps++
    }
    cursor.close()
    myRef.child( pathString: "calendarEvents").setValue(eventArr)

}
```

Figure 24. Reaching the user's calendar events

7.7 Obtaining user call log data

Application asks READ_CALL_LOG permission to get the call logs of the user. **CallLog** class contains **Calls** class and *NUMBER* and *DURATION* are the properties of that last one. Phone numbers and duration of that call is acquired through the cursor and via getString method is converted to the string. Collected data is added to the hashmap.

```
fun getLogs(){
    val callMap = HashMap<String, String>()
    val cursor : Cursor? = contentResolver.query(CallLog.Calls.CONTENT_URI,
        projection: null,
        queryArgs: null,
        cancellationSignal: null)
    while (cursor!!.moveToNext()){
        val numberIndex : Int = cursor!!.getColumnIndex(CallLog.Calls.NUMBER)
        val durationIndex : Int = cursor!!.getColumnIndex(CallLog.Calls.DURATION)
        val number : String! = cursor.getString(numberIndex)
        val callDuration : String! = cursor.getString(durationIndex)
        callMap.put(number, callDuration)
    }
    cursor.close()
    myRef.child( pathString: "CallDuration").updateChildren(callMap as Map<String, Any>)
}
```

Figure 25. Reaching the user's call logs

7.8 Performance evaluation

The application was tested both in virtual and real devices. The Android Virtual Device manager of Android Studio provides miscellanious phone types to test application in different versions of API. Total 12 tests were carried out on mobile phones and users shared their feedback about the correctness and performance of the app. App essentially is tested on two widespread versions on Android 9 and Android 10.

While testing application on a bit earlier versions some problems were experienced. For instance, one of the users had a bit earlier versions of Android (Nougat) and after installation user pressed buttons to collect data. However,

app stopped by a critical error immidiately. General feedback were relatively normal and app performed properly on other devices. Below pictures portray the user impressions who tested SpyJam app on their own devices.

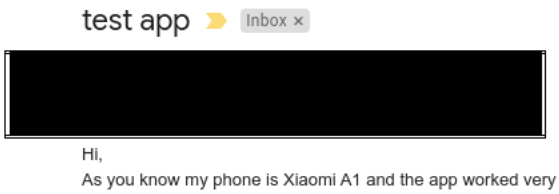

Figure 26. Feedback #1

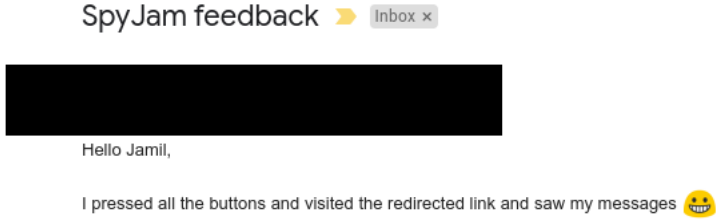

Figure 27. Feedback #2

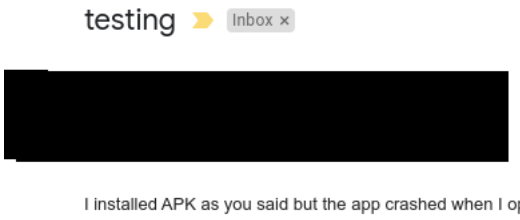

Figure 28. Feedback #3

Figure 29. Feedback #4

Figure 30. Feedback #5

7.9 Future work

This application is currently developed only for the Android platform. However, developers can implement the same concept app for the iOS platform to explore Apple devices' security testing in the future. The project will be on the Git repository, and contributions are appreciated. Other app features could be implemented, such as voice recording, capturing a picture, etc. In addition, it is eligible to include multiple language options other than English to enhance the area of usage.

8 Summary

Permissions are considered essential part of the security system of Android. Developers define user permissions in the manifest file in order to gain access to system recources. The goal of the study was to determine what kind of data could be accessed through granted permissions. Mobile phones already became a gadgets that even most individuals carry, share and store classified documents within them. Through dangerous permissions which have been implemented in the prototype app to demonstrate that obtaining data is not so much hard process. Thats why end users have to be careful and vigilant especially about the third-party apps. Apps from only reliable central sources such as Google Play Store, Amazon Appstore etc. are recommended to download and use.

Studies and investigations have been reached an ultimate result that private user data stealing occurs due to the negligency and digital illiteracy of user. Several user perspectives were described in the thesis to avoid mistakes and enhance digital secuirity awareness of users. Besides that permission tips for developers has been mentioned to accurately implement and configure as well.

The following graduation thesis, "Malicious Android application for security testing" is about to test how much private user data can an Android application obtain via user permissions. As a result of this thesis, a prototype Android application was built, and collected user data is depicted on a webpage. The thesis structure contains the following parts: Background of Android, Personal Data and Regulations, Android Permissions, Permission implementation and understanding for the user and developer, Technical overview of system and data workflow, and lastly, Application implementation.

In the background part, the history and evolution of Android is stated. In addition, the architecture of Android OS and Android application parts were clearly explained as well. After granting dangerous permissions, which are

categorized in the Android permissions chapter, through two essential objects of Android applications - content providers and content resolvers, data transfer occurs. For the users and the developers to comprehend the permission concept properly and to configure it accurately, tips and suggestions were given in the thesis as well. By taking into account the strictness of data privacy, particular GDPR statements, and data processing terms mentioned in the thesis. This Android application can be used as a good demonstrative tool to present to smartphone users how important it is to be responsible and cautious to prevent and avoid data stealing.

To conclude, in this thesis Kotlin programming language used to develop Android application and essential web tools such as Javascript, Django, Bootstrap, Firebase realtime DB is used to create web application. Web app is hosted on PaaS plaform.

References

[1] Developers, Android Open Source Project. https://source.android.com/, (01.11.2020)

[2] Gian Luca Scoccia, Ivano Malavolta, Marco Autili, Amleto Di Salle, and Paola Inverardi. User-centric android flexible permissions. In Proceedings of the 39th International Conference on Software Engineering Companion, pages 365–367. IEEE Press, 2017. (07.11.2020)

[3] Paul Gerber, Melanie Volkamer, and Karen Renaud. The simpler, the better? presenting the coping android permission-granting interface for better privacyrelated decisions. Journal of Information Security and Applications, 34:8–26, 2017.(19.11.2020)

[4] Yeonjoon Lee, Tongxin Li, Nan Zhang, Soteris Demetriou, Mingming Zha, XiaoFeng Wang, Kai Chen, Xiaoyong Zhou, Xinhui Han, and Michael Grace. Ghost installer in the shadow: Security analysis of app installation on android. In Dependable Systems and Networks (DSN), 2017 47th Annual IEEE/IFIP International Conference on, pages 403–414. IEEE, 2017.(20.11.2020)

[5] Kaspersky investigation. https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/ (01.11.2020)

[6] Mauro Conti, Vu Thien Nga Nguyen, and Bruno Crispo. Crepe: Context-related policy enforcement for android. In ISC, volume 10, pages 331–345. Springer, 2010. (04.11.2020)

[7] Alexandre Bartel, Jacques Klein, Yves Le Traon, and Martin Monperrus. Automatically securing permission-based software by reducing the attack surface: An application to android. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pages 274–277. ACM, 2012. (22.11.2020)

[8] Adrienne Porter Felt, Kate Greenwood, and David Wagner. The effectiveness of install-time permission systems for third-party applications. University of California at Berkely, Electrical Engineering and Computer Sciences, Technical report, 2010. (23.11.2020)

[9] Ryan Stevens, Jonathan Ganz, Vladimir Filkov, Premkumar Devanbu, and Hao Chen. Asking for (and about) permissions used by android apps. In Proceedings of the 10th Working Conference on Mining Software Repositories, pages 31–40. IEEE Press, 2013. (25.12.2020)

[10] Permission overview. https://developer.android.com/guide/topics/permissions/overview (08.11.2020)

[11] Jingzheng Wu, Mutian Yang, and Tianyue Luo. Pacs: Pemission abuse checking system for android applictions based on review mining. In Dependable and Secure Computing, 2017 IEEE Conference on, pages 251–258. IEEE, 2017. (25.12.2020)

[11-1] Abdullah.B. Analysis of Software Applications Computing Resources Usage on the Edge: A Case Study of Speech Recognition. University of Tartu. Institute of Computer Science. 2019. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=67480&year=2019 (10.01.2021)

[11-2] Gorohhova.J. Malicious Android App for Security Testing. University of Tartu. Institute of Computer Science. 2020. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=70525&year=2020 (10.01.2021)

[12] Android Developers. Android history review. http://developer.android.com/about/index.html (30.10.2020)

[13] Android OS names. https://inshorts.com/en/news/why-google-names-its-operating-system-after-sweets-1478511182621 (04.10.2020)

[14] Krajci I., Cummings D. (2013) History and Evolution of the Android OS. In: Android on x86. Apress, Berkeley, CA. (05.01.2021)

[15] Android versions. https://www.lifewire.com/android-versions-4173277 (20.10.2020)

[16] C. Nimodia, and H. Deshmukh. "ANDROID OPERATING SYSTEM" Software Engineering, 3(1), 10. 2012

[17] W. Hu, D. Han , A. Hindle, and K. Wong. "The build dependency perspective of android's concrete architecture" The 9th Working Conference on Mining Software Repositories, page to appear, 2012. (6.01.2021)

[18] App sandboxing. http://hiqes.com/android-security-part-1/ (22.10.2020)

[19] Android Runtime. https://www.journaldev.com/23464/android-runtime-dvm-vs-art-aot-vs-jit (02.11.2020)

[20] Android Developers. Android Notification Manager. https://developer.android.com/reference/android/app/NotificationManager (05.11.2020)

[21] Android Developers. Android Native development kit. https://developer.android.com/ndk/guides (12.11.2020)

[22] Android Developers. Application Fundamentals. http://developer.android.com/guide/components/fundamentals.html. (21.10.2020)

[23] Manifest declaration. https://developer.android.com/reference/android/app/Activity#Perm (21.10.2020)

[24] Risky app permissions. https://laptrinhx.com/a-developers-guide-to-risky-android-app-permissions-3804176284/ (25.10.2020)

[25] Android Developers. Application Fundamentals. http://developer.android.com/guide/components/fundamentals.html. (05.12.2020)

[26] Android Developers. Android services. https://developer.android.com/guide/components/services (06.12.2020)

[27] Android Developers. Content providers. https://developer.android.com/guide/topics/providers/content-providers (07.12.2020)

[28] Content Providers & Content Resolvers. https://www.androiddesignpatterns.com/2012/06/content-resolvers-and-content-providers.html (15.12.2020)

[29] Android Developers https://developer.android.com/guide/topics/providers/content-providers#Advantages (08.12.2020)

[30] https://www.c-sharpcorner.com/article/understand-content-provider-in-xamarin-with-visual-studio/ (10.12.2020)

[31] https://developer.android.com/guide/topics/providers/content-provider-basics#kotlin (12.12.2020)

[32] https://developer.android.com/guide/components/broadcasts (15.12.2020)

[33] GDPR acts. https://gdpr-info.eu/ (13.12.2020)

[34] General Data Protection Regulations. https://gdpr-info.eu/art-4-gdpr (13.12.2020)

[35] Delac, Goran & Silic, Marin & Krolo, Jakov. (2011). Emerging security threats for mobile platforms. 1468-1473. (13.12.2020)

[36] Previois permission mechanism. https://www.inlovewithandroid.com/app-permissions-in-android-m.html (17.12.2020)

[37] Runtime permission user flow. https://blog.iamsuleiman.com/runtime-permissions-android-marshmallow/ (20.12.2020)

[38] Permission definition. (https://developer.android.com/guide/topics/permissions/overview) (17.12.2020)

[39] Android developers. One-time permission. https://android-developers.googleblog.com/2020/02/Android-11-developer-preview.html) (01.01.2021)

[40] Almomani, Iman & Khayer, Aala. (2020). A Comprehensive Analysis of the Android Permissions System. IEEE Access. 8. 216671 - 216688. 10.1109/ACCESS.2020.3041432. (10.01.2021)

[41] Android Developers. Best practise tips.https://developer.android.com/training/permissions/notes (02.01.2021)

[42] Kotlin language. https://kotlinlang.org/ (03.01.2021)

[43] Google official recommendation. https://www.zdnet.com/article/google-were-using-kotlin-programming-language-to-squash-the-bugs-that-cause-most-crashes/ (05.01.2021)

[44] Javascript DataTables. https://datatables.net/ (05.01.2021)

[45] Google Firebase Database. https://firebase.google.com/docs/database (05.01.2021)

[46] Django project. https://www.djangoproject.com/ (06.01.2021)

[47] PythonAnywhere Paas Platform. https://www.pythonanywhere.com/ (07.01.2021)

Appendix

I. Links for repositories

Github repo link for Android application - https://github.com/jamil2595/SpyJam
Github repo link for Web application - https://github.com/jamil2595/django-web
Android APK - https://drive.google.com/drive/folders/1MdWYeevEYOUrsnrBZvEm085JaqRPDwRm?usp=sharing

II. Example of webpage displaying data send by SpyJam application

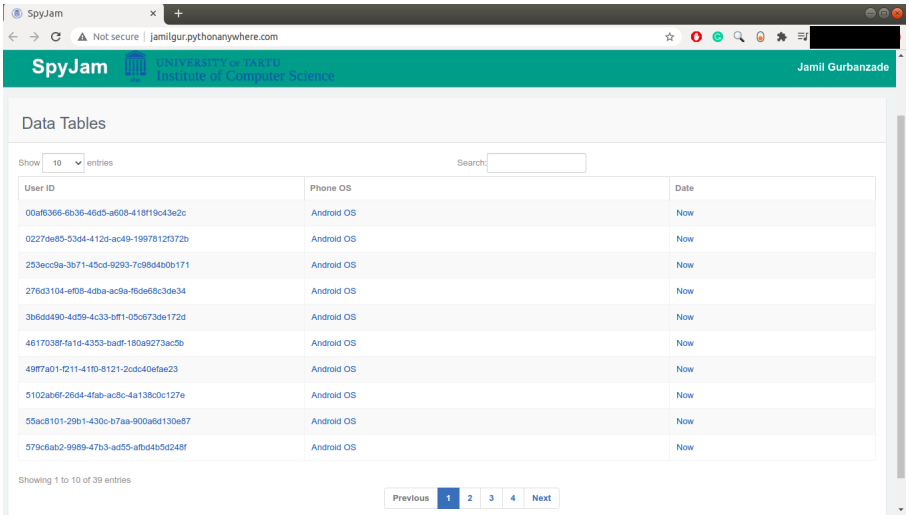

Figure 31. Webpage view

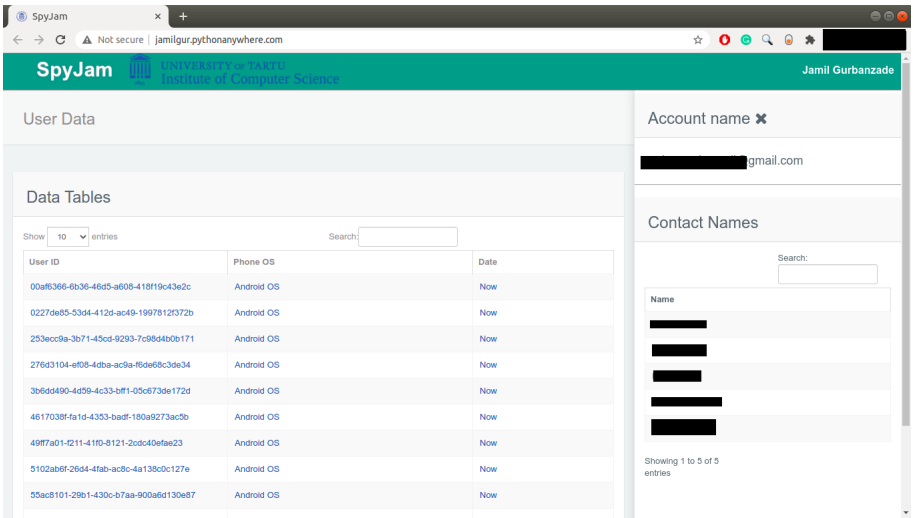

Figure 32. Collected data on the webpage

III. Example of webpage displaying data send by SpyJam application in the mobile screen

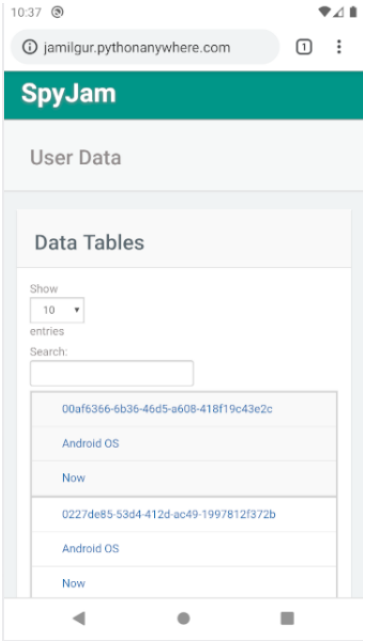

Figure 33. Mobile view

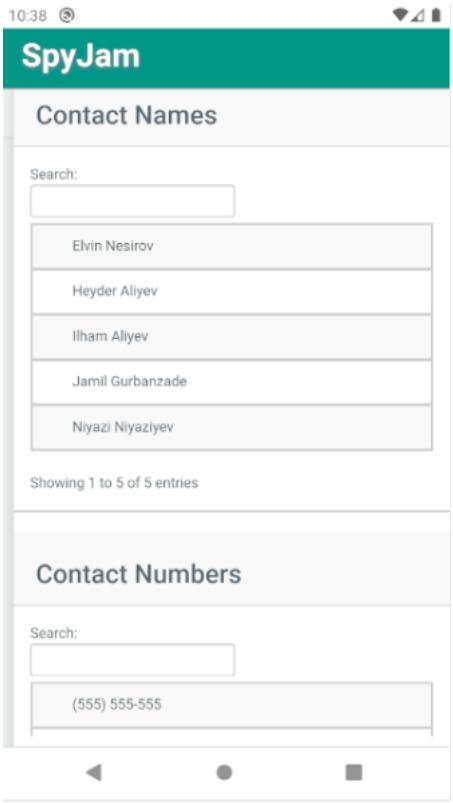

Figure 34. Collected data on the mobile view

IV. Screen of mobile application

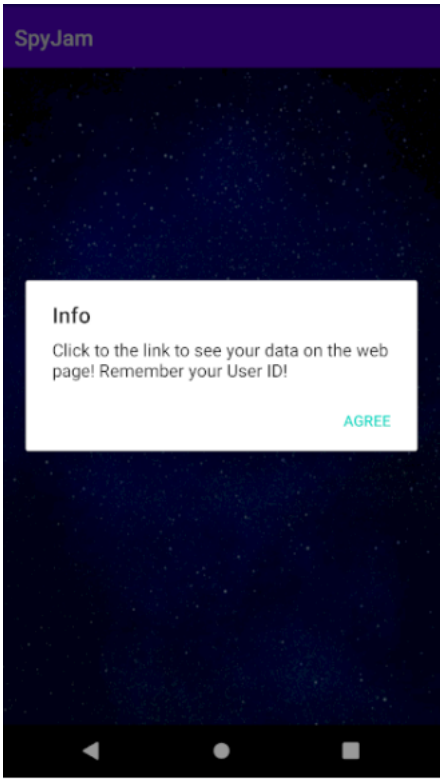

Figure 35. Second info screen

Figure 36. Second info screen

License

Non-exclusive license to reproduce thesis and make thesis public

I,_____*Jamil Gurbanzade*_____,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive license) to:

1.2 make available to the public via the web environment of the University of Tartu, includ-ing via the DSpace digital archives, under the Creative Commons license CC BY NCND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work from 14/01/2021 until the expiry of the term of copyright,

_____
_____
_____*Malicious Android application for security testing*
(title of thesis)

supervised by *Alo Peets*
(supervisor's name)

2. I am aware of the fact that the author retains the rights specified in p. 1.

3. I certify that granting the non-exclusive license does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Jamil Gurbanzade

14/01/2021