UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Jekaterina Gorohhova

Malicious Android app for security testing

Bachelor's Thesis (9 ECTS)

Supervisor: Alo Peets

Tartu 2020

Pahatahtlik androidi rakendus turvalisuse testimiseks

Lühikokkuvõte:

Selles bakalaureusetöös kirjeldatakse Androidi rakenduse prototüüpi, mida saab kasutada illustreeriva näitena, kuidas antud operatsioonsüsteemi rakendused saavad kasutada õigusi kasutaja tundlikele andmetele juurdepääsuks, loomist ja kasutamist. Rakendus kasutab ainult sisseehitatud Androidi API-t ja saab ligipääsu ning võib edastada erinevat tüüpi delikaatseid kasutaja andmeid, sealhulgas fotod, allalaaditud failid, asukoht, SMS-id ja kõnelogid. Selle lõputöö tulemusi saab kasutada kasutajate digitaalse kirjaoskuse parandamiseks.

Võtmesõnad:

Androidi rakendus, andmete kogumine, õigused

CERCS: P175, Informaatika, süsteemiteooria

Malicious Android app for security testing

Abstract:

This bachelor’s thesis details the design, implementation, and deployment of a prototype of an Android application that is used for an illustrative example of how applications in a given operating system can use permissions to access user's sensitive data. Application uses only built-in Android API and it can access and transmit different kinds of user’s sensitive information including photos, downloaded files, location, SMS messages, and call logs. The output of this thesis can be used to enhance the digital literacy of users.

Keywords:

Android app, data collection, permissions

CERCS: P175, Informatics, systems theory

Table of Contents

1 Introduction

Over the past decade, people of all ages have increasingly used mobile devices for various needs. In this regard, a large number of mobile applications gain access to countless user data, from physical indicators to emotional state. The most used operating system for them at the moment is Android, which occupies about 72% of all used mobile devices, according to the state on March 2020 [1]. Moreover, Statista researchers claim that users of the Android official marker, Google play, worldwide downloaded 84.3 billion mobile apps during year 2019 [2]. On the other hand, the widespread adoption of Android, as well as the fact that the platform is open source, made it a prime target for malware developers. According to Mobile Threat Report by Pulse Secure Mobile Threat Center, Android is considered the number one smartphone OS in terms of the amount of malware [3].

1.1 The Problem

Despite the general technological progress, the digital literacy of users regarding the security of their digital identity is still quite poor. At the first launch, the application informs user about the terms of use, as well as requests additional rights, for example, access to a microphone or camera. Most users give the application all the requested rights without even looking, even if they are obviously redundant - for example, full access to their Google account, only to play some game [4][5]. This means that developer of that game can possibly access any information linked to user’s Google account – read messages, check events in Calendar or user’s phone current location.

1.2 The Goal

This work is a study about what data smartphones can access in the Android operating system through applications, using the set of given rights. In the course of this bachelor's thesis, a prototype of application is created for the above purpose, which shows the user what data this application can use with the given rights, explains how malware can use this data and gives advice on how to protect their data by restricting application rights.

2 Background

2.1 Definition of permissions

According to the Android Developers Web Handbook [6], the role of rights in the operating system is to protect user privacy. In the Android operating system, applications must request permission to use sensitive user data (e.g., contacts, SMS, media files, photos) and some system features (camera, Internet access, microphone). Depending on the function, the operating system may grant permissions independently or ask the user for confirmation. By default, due to general Android's security architecture, applications can't do anything that could harm other applications, the user, or the operating system itself. It also includes reading or writing user data or other application data, accessing the Internet, etc. Each Android application also runs in its own sandbox with limited access. If an application needs to use any resources outside of this sandbox, that application must request it. Each application must publish the necessary permissions by adding the appropriate tags to its manifest.

Permissions are divided into several levels of protection, as shown in Figure 1. The level of protection affects whether you need to ask the user for permission or whether the operating system can handle the permission automatically. There are three levels of protection that affect third-party applications: *normal*, *signature*, and *dangerous*.

This is about normal level of protection when an application needs access to resources outside its sandbox, but there is very little risk of compromising user privacy or influence on other application performance. For example, "ACCESS_NETWORK_STATE", which allows applications to access information about networks is a permission with a normal level of protection. If the application indicates that it needs a normal protection level permission, the system will automatically grant that permission during installation. In this case, the user's consent is not requested, but users also cannot revoke those permissions.

For signature permissions, the system grants the required permissions to applications during installation, as with regular permissions, but only if the application that is trying to use that right is signed with the same certificate as the application that defines the permissions.

It is about dangerous level of protection when an application wants to use data or resources that could potentially affect the user's own data, influence on other application performance or the user's privacy. For example, “READ_CONTACTS” permission, which gives application an ability to read user’s contacts data is a permission with dangerous level of protection. If an application reports that it needs that kind of permission, the system must ask the user for their consent. Until the user has given their confirmation, the application will not be able to offer features that depend on this permission.

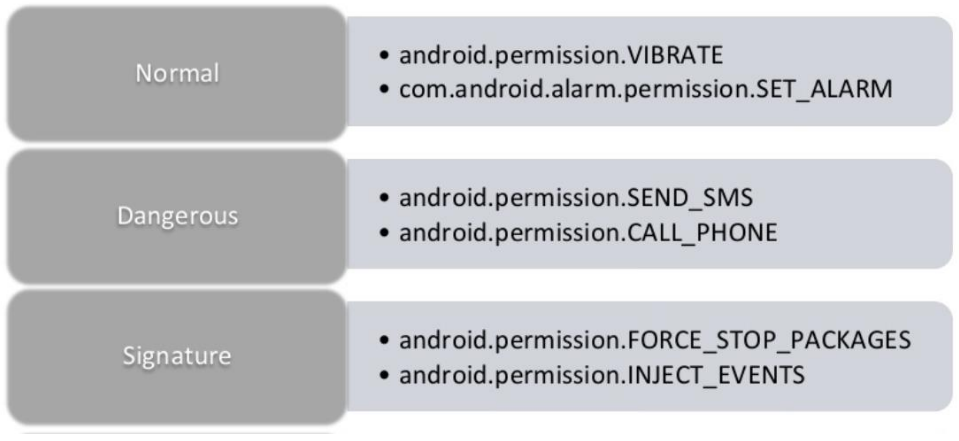

Normal
• android.permission.VIBRATE
• com.android.alarm.permission.SET_ALARM
Dangerous
• android.permission.SEND_SMS
• android.permission.CALL_PHONE
Signature
• android.permission.FORCE_STOP_PACKAGES
• android.permission.INJECT_EVENTS

Figure 1: Three levels of protection [7]

Permissions are divided into different groups according to the capabilities or functions of the device. Each request for permission is then processed at group level, and one group of permissions corresponds to several declarations of rights in the application manifest. This means that, for example, a group of SMS messages contains both a READ_SMS declaration and a RECEIVE_SMS declaration. Dividing permissions in this way allows the user to make more informed choices without having to deal with complex technical permission requests. Any permission can belong to a group of permissions; it does not depend on the level of protection. All permissions currently defined in the system can be checked using the system application *Settings*.

2.2 Definition of personal data and data processing

According to Estonian Personal Data Protection Act [8] and General Data Protection Regulation (hereinafter GDPR) [9], personal data are all data concerning a natural person who is unambiguously identified, as well as directly or indirectly unambiguously identifiable, which express the physical, mental, psychological, economic, cultural or social characteristics, relationships and affiliation of that person.

Also, personal data are either sensitive or non-sensitive. Normally, sensitive personal data includes the following, as stated by Estonian Personal Data Protection Act [8]:

1. data describing political views, religious or philosophical beliefs;
2. data describing ethnic origin or racial affiliation;
3. information on health status or sexual life;
4. information concerning the criminal offenses committed and court punishments;
5. information concerning criminal proceedings.

Processing of personal data means the collection, storage, organization, modification, extraction, use, transfer, aggregation, deletion or destruction of data, or any of the operations applicable to data, regardless of the manner in which the operations are carried out or the means used [8] [9].

2.3 Developer perception

Requests for permissions protect sensitive data on the device and should only be used if access to the data is necessary for the workflow of the application [10].

The developers of the Android Developers Web Handbook [11] [12] recommend following these basic rules when working with Android rights:

1. Avoid applying for unnecessary permissions. Use only those permissions that are necessary for the work of the application. Depending on how and what permissions you use, there may be some other way to perform the required functionality that does not require access to the user's sensitive data;
2. Pay attention to the permissions of the libraries you use. When using libraries, its legal requirements also apply. You need to know which libraries you use, what rights they need and what they are used for. For example, some ad and analytics libraries may require access to some rights groups that are not directly used by the application, but from the user's point of view, the rights request comes from the application and not from the library.

3. Be transparent and honest. When applying for permission, clearly state what and what for the access is needed so that users can make informed decisions. Make this information available immediately through authorized dialog boxes at the time of installation, startup, or upgrade.
4. Make system access clear and visible. Be constantly alert when an application is using sensitive data (such as a camera, microphone, viewing user files on a storage medium, or contact list) - this allows the user to understand what the application is doing and avoids a situation where the user may suspect that their data may be secretly collected.

Developer faces the difficult task of creating an application that allows the user to perform necessary actions, and at the same time prevent the leakage of any personal information.

However, many developers neglect the rule on the use of redundant rights. This happens for various reasons, but in the end it can often lead to security problems and data leaks. Most of the software at the project or prototype level often has deliberately overestimated requirements for the granted rights and underestimated requirements for the security of the internal infrastructure. Also, regularly the list of required permissions in applications is laid down with an eye on the future - to enable additional functionality, bypass possible compatibility problems, or the ability to ignore the limitations of older OS versions.

Despite the fact that large development companies identify these shortcomings during release (or in the "day one patch"), many applications created by small companies or individuals leave such vulnerabilities in themselves for a long time, sometimes even until the closure of the project.

Also, it is impossible to exclude the human factor and the deliberately malicious inclusion of redundant functionality for further collection of information about the user, including confidential information, for further use.

For example, some developers transmit redundant personal data of users, which may even include the exact location, to advertising agencies [13]. For this to happen legally, the user must be informed and give his consent. Though, many users do not consider it necessary to read long informational agreements, and often do not pay attention to what rights and why the application requires, which in the future can be used by an unscrupulous developer.

2.4 User perception

Nowadays, more and more people are using smartphones in everyday life, and there are thousands of applications to meet various needs - social networking applications, games, etc. Millions of applications are available for download at any time, and each of them requires any permissions to work correctly. On such a scale, it is natural that issuing permissions for users has become something common. Many people, especially those who are not technically savvy, are negligent with regard to their own personal information, not paying attention to exactly what data developers receive through applications.

Many people are at risk of losing important information during the processing of personal data, among them [14]:

- people using bank cards;
- people receiving medical services;
- owners of pension savings;
- bank depositors;
- property owners.

This is just a short list; many others can suffer from leakage of personal data. Therefore, a personal data protection system is so necessary.

2.4.1 Consequences of sensitive data leakages

The consequences of leaks can be serious for data owners and developers. For the first group, there are numerous risks of becoming a victim of intruders. They may suffer from:

- the disclosure of any information related to the person;
- blackmail, social engineering;
- illegal debit of funds from a bank card;
- interference in personal life;
- threats to their loved ones.

The minimum consequence will be the unlawful transfer of information to advertisement companies. But even this makes it possible to file a lawsuit against the developer if the source of the leakage can be reliably established [8][9].

3 System design

3.1 System requirements

The following is a list of functional requirements that must be performed.

1. User should be able to view a summary of the categories of collected data after they have been collected.
2. User should be able to see additional info regarding categories of collected data.
3. User should be able to enter an email address to which the data should be sent.
4. The app should be able to collect the data from user’s phone and put it in a zip-folder.
5. The app should be able to send a zip-folder with collected data to email address, entered by user.

3.2 System architecture

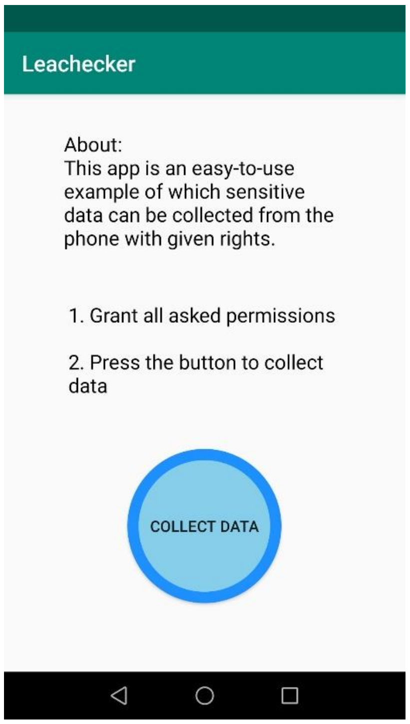
Application work is linear and is executed in the following order, step by step:

1. The application is installed;
2. The application asks for the necessary permissions at startup;
3. The application is waiting for the "Test" button to be pressed;
4. The application checks the availability of photos and downloaded files in the storage, the accessibility of geolocation data, the presence of SMS and calls, as well as authenticated Google accounts;
5. The application shows a screen with additional information about the data that can be retrieved from the phone, and the permissions that are responsible for this;
6. The application is waiting for user to press the "Send" button;
7. The application calls a mail client preinstalled on the phone with a draft letter, where a zip archive with the extracted data is attached;
8. The application ends its work.

4 Prototype implementation and evaluation of the performance

4.1 Prototype implementation

The application starts its work after asking and receiving all necessary permissions. User must press the “TEST” button shown in Figure 3. Next, the application starts collecting data from the user's smartphone, while at the same time showing him information about what exactly will be collected, as in Figure 4. Info and sending email views can be found in Appendix at Figures from 21 to 27.


Leachecker

About:
This app is an easy-to-use example of which sensitive data can be collected from the phone with given rights.

1. Grant all asked permissions

2. Press the button to collect data

Figure 3. Starting screen

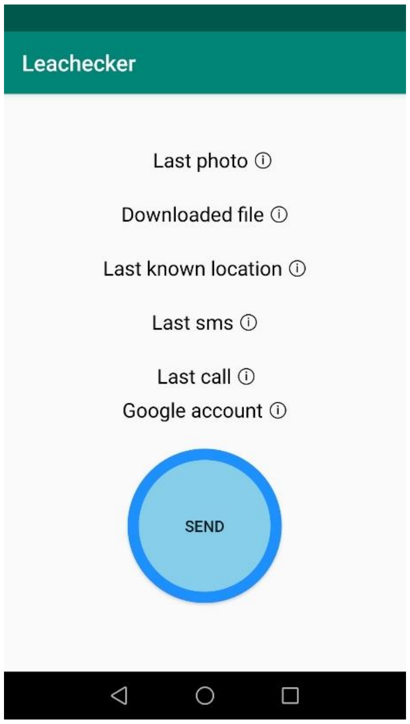
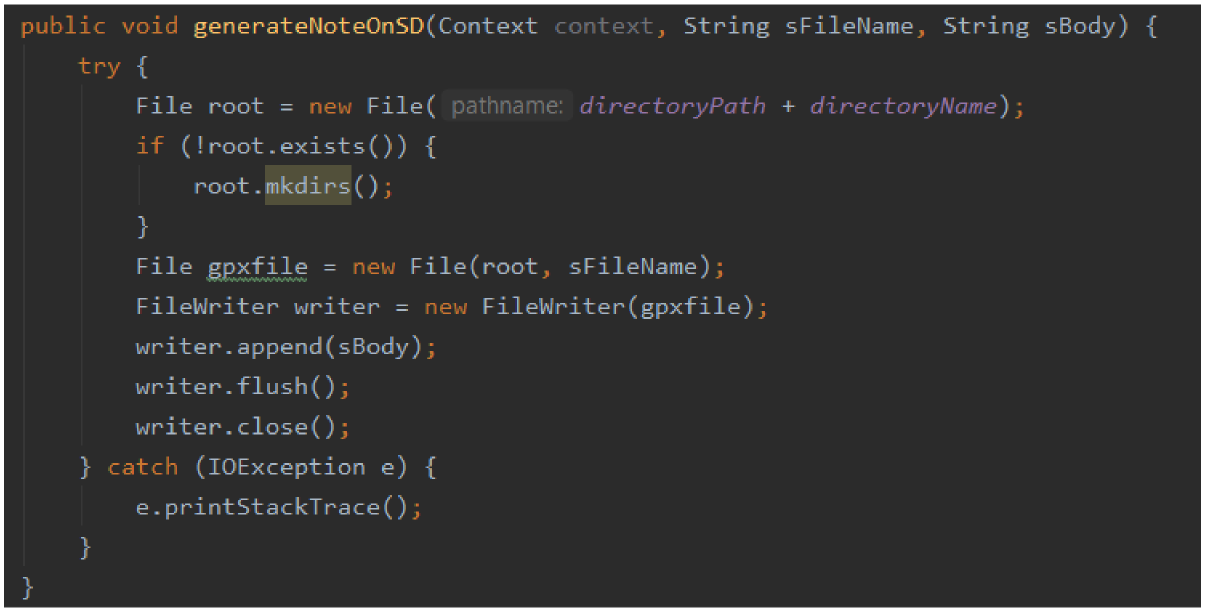


Figure 4. General info screen

The main functionality of the application is situated in class “DataCollectingActivity.java”. At first, the folder “collectedData” is created in standard directory named “Downloads” for further use. All the data gathered from the phone is stored in this folder, and it is packed into zip archive at the final stages of the application work.

The application collects the data that is described in more detail in paragraphs 4.2.1 - 4.2.6. Text information is placed separately in text files using the generateNoteOnSD method, shown in Figure 5.

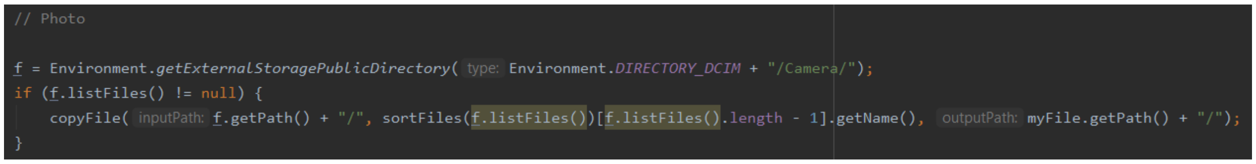


```java
public void generateNoteOnSD(Context context, String sFileName, String sBody) {
    try {
        File root = new File( pathname: directoryPath + directoryName);
        if (!root.exists()) {
            root.mkdirs();
        }
        File gpxfile = new File(root, sFileName);
        FileWriter writer = new FileWriter(gpxfile);
        writer.append(sBody);
        writer.flush();
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 5. generateNoteOnSD method

4.1.1 Photo

Application uses permissions “READ_EXTERNAL_STORAGE” and “WRITE_EXTERNAL_STORAGE” to get access to photos in the storage and copy them. It enters the directory „Camera“, sorts all the photos there, selects the last one and copies it into the folder “collectedData”, as shown on Figure 6.

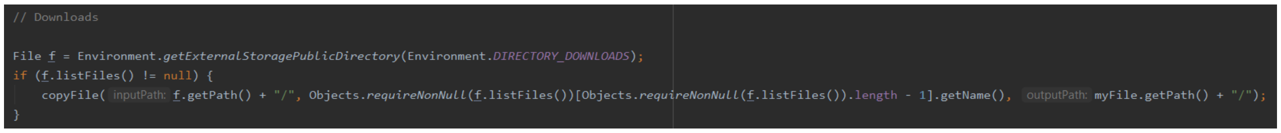


```java
// Photo

f = Environment.getExternalStoragePublicDirectory( type: Environment.DIRECTORY_DCIM + "/Camera/");
if (f.listFiles() != null) {
    copyFile( inputPath: f.getPath() + "/", sortFiles(f.listFiles())[f.listFiles().length - 1].getName(),  outputPath: myFile.getPath() + "/");
}
```

Figure 6. Accessing photo from storage

4.1.2 Downloaded file

Application uses permissions “READ_EXTERNAL_STORAGE” and “WRITE_EXTERNAL_STORAGE” to get access to downloaded files in the storage and copy them. It enters the directory „Downloads“, selects the last one file and copies it into the folder „collectedData“, as shown on Figure 7.

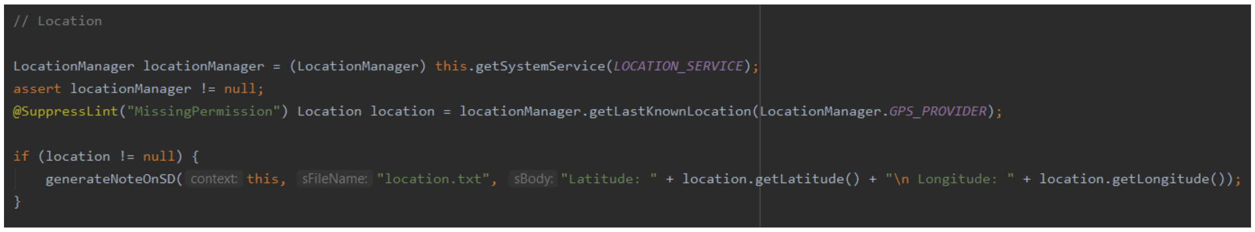


```java
// Downloads

File f = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);
if (f.listFiles() != null) {
    copyFile( inputPath: f.getPath() + "/", Objects.requireNonNull(f.listFiles())[Objects.requireNonNull(f.listFiles()).length - 1].getName(),  outputPath: myFile.getPath() + "/");
}
```

Figure 7. Accessing downloaded file from storage

4.1.3 Last known location

Application uses “ACCESS_FINE_LOCATION” permission to be able to gain access to user’s location. It gets last known coordinates via “GPS_PROVIDER”, writes them to the file and puts this file to the folder “collectedData”, as shown on Figure 8.

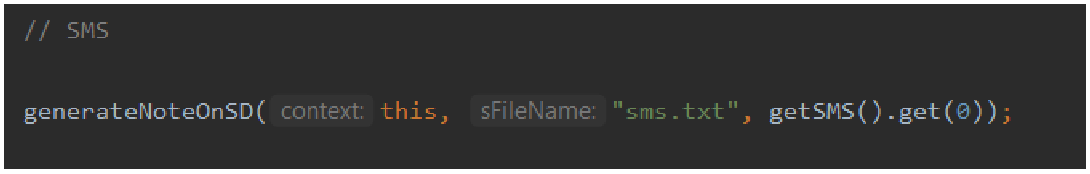


```java
// Location

LocationManager locationManager = (LocationManager) this.getSystemService(LOCATION_SERVICE);
assert locationManager != null;
@SuppressLint("MissingPermission") Location location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

if (location != null) {
    generateNoteOnSD( context: this,  sFileName: "location.txt",  sBody: "Latitude: " + location.getLatitude() + "\n Longitude: " + location.getLongitude());
}
```

Figure 8. Accessing info about last location

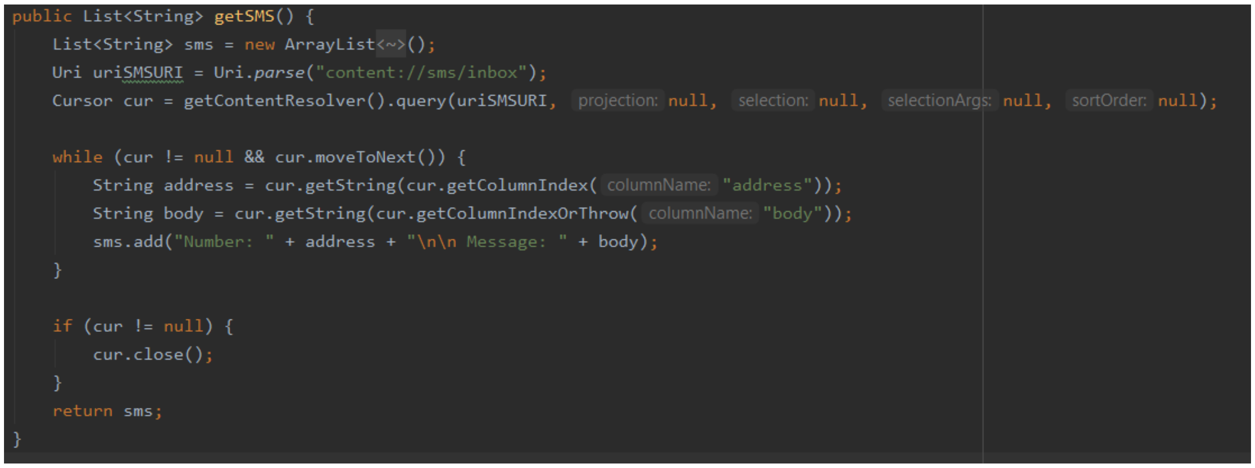
4.1.4 Last SMS

Application uses “READ_SMS” permission to be able to read user’s SMS. It enters the “//sms/inbox” directory, selects the last sms and parses the content to pick out the sender’s number and actual text of the SMS message, then puts it to the text file and adds the file to the folder “collectedData”, as shown on Figures 9 and 10.



```java
// SMS

generateNoteOnSD( context: this,  sFileName: "sms.txt", getSMS().get(0));
```

Figure 9. Accessing info about last SMS

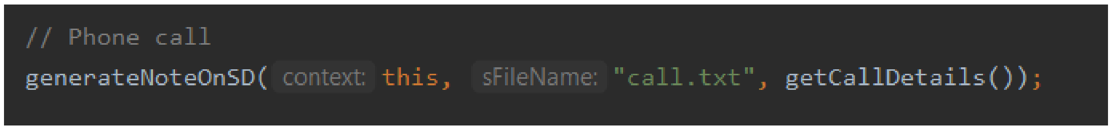


```java
public List<String> getSMS() {
    List<String> sms = new ArrayList<~>();
    Uri uriSMSURI = Uri.parse("content://sms/inbox");
    Cursor cur = getContentResolver().query(uriSMSURI,  projection: null,  selection: null,  selectionArgs: null,  sortOrder: null);

    while (cur != null && cur.moveToNext()) {
        String address = cur.getString(cur.getColumnIndex( columnName: "address"));
        String body = cur.getString(cur.getColumnIndexOrThrow( columnName: "body"));
        sms.add("Number: " + address + "\n\n Message: " + body);
    }

    if (cur != null) {
        cur.close();
    }
    return sms;
}
```

Figure 10. getSMS method

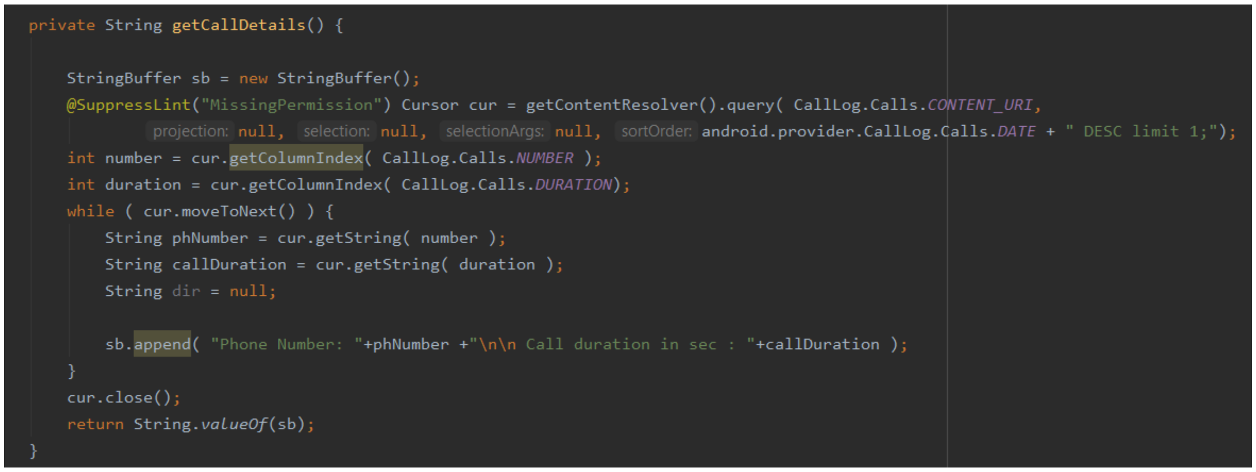
4.1.5 Last call

Application uses "READ_CALL_LOG" permission to get access to user's list of calls. The application then parses the calls log to select the last call and pick out the caller's number, call duration and put them to the text file. Afterwards, this text file is added to the folder „collectedData", as shown on Figures 11 and 12.



```
// Phone call
generateNoteOnSD( context: this,  sFileName: "call.txt", getCallDetails());
```

Figure 11. Accessing last call info

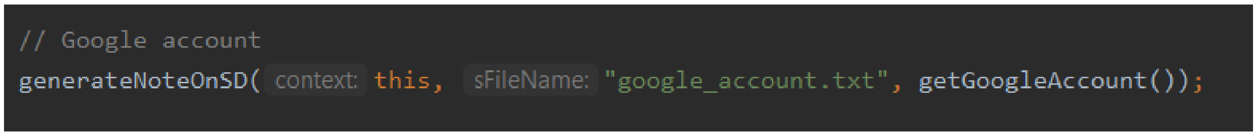


```
private String getCallDetails() {

    StringBuffer sb = new StringBuffer();
    @SuppressLint("MissingPermission") Cursor cur = getContentResolver().query( CallLog.Calls.CONTENT_URI,
            projection: null,  selection: null,  selectionArgs: null,  sortOrder: android.provider.CallLog.Calls.DATE + " DESC limit 1;");
    int number = cur.getColumnIndex( CallLog.Calls.NUMBER );
    int duration = cur.getColumnIndex( CallLog.Calls.DURATION);
    while ( cur.moveToNext() ) {
        String phNumber = cur.getString( number );
        String callDuration = cur.getString( duration );
        String dir = null;

        sb.append( "Phone Number: "+phNumber +"\n\n Call duration in sec : "+callDuration );
    }
    cur.close();
    return String.valueOf(sb);
}
```

Figure 12. getCallDetails method

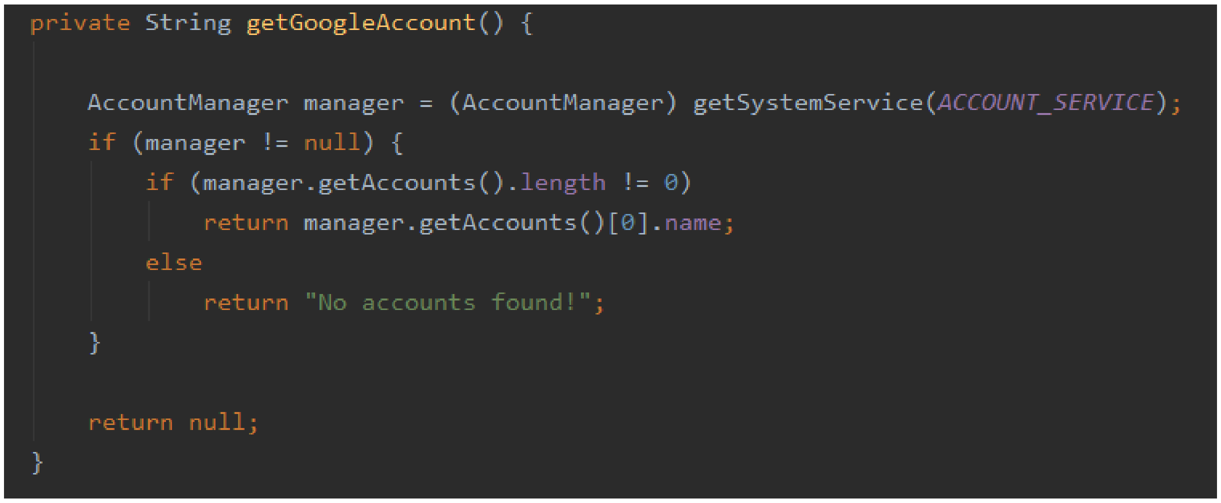
4.1.6 Google account

Application uses "GET_ACCOUNTS" permission to get the list of accounts in Accounts Servise. It checks if there are existing accounts and selects first of them, if it occurs, then puts it to the text file and adds the file to the folder "collectedData" created beforehand, as shown on Figures 13 and 14.



```
// Google account
generateNoteOnSD( context: this,  sFileName: "google_account.txt", getGoogleAccount());
```

Figure 13. Accessing info about Google Account



```
private String getGoogleAccount() {

    AccountManager manager = (AccountManager) getSystemService(ACCOUNT_SERVICE);
    if (manager != null) {
        if (manager.getAccounts().length != 0)
            return manager.getAccounts()[0].name;
        else
            return "No accounts found!";
    }

    return null;
}
```

Figure 14. getGoogleAccount() method

4.2 Device requirements

The minimum supported version of the Android for optimal application performance is Android 4.4 KitKat. As mentioned in Android Studio IDE, according to information gathered by Google during 7-day collection period, which ended on July 13, 2020, versions 4.4 KitKat and higher cover 98.1% of all active devices [15].

The necessary services for the application to work are:

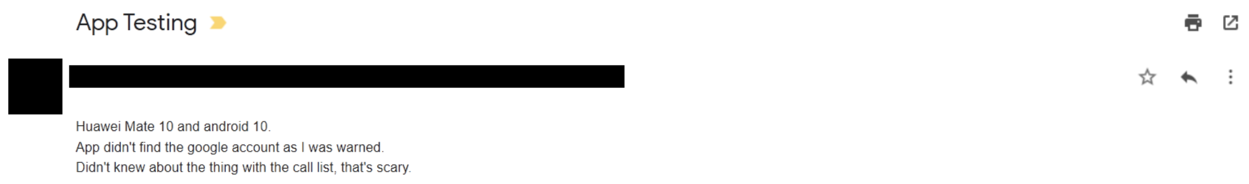
1. location services;
2. Internet connection;
3. ability to make calls;
4. ability to send SMS messages;
5. ability to take pictures with camera;
6. connected Google account.
7. access to storage and read-write permissions in Download directory.

4.3 User tests and evaluation of performance

18 tests were carried out on personal smartphones of ordinary people of different models and different manufacturers, among which were Samsung Galaxy S7 Edge, Huawei P10, Huawei P30 and P40, OnePlus A6003 and Xiaomi Pocophone F1 and others and different Android versions, mostly Android 9 Pie and Android 10. Testing was carried out by installing the apk-file of the application on smartphones, after which users personally checked the correctness of the information and gave written feedback in free form.

Since the application at this moment is platform-dependent, users of other smartphones, except for the Samsung Galaxy line, experienced some problems when working with the application, such as the lack of data on the Google account linked to the system or the mistaken selection of photos from the gallery. On the Android version Oxygen, the application was turned off by a critical error. For the rest, users confirmed the correctness of the information and responded that many of the information on the permissions given in the application was new to them.

There are some emails received from users who tested the application at Figures from 15 to 20.

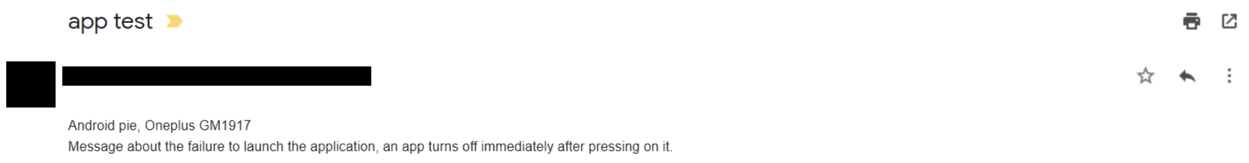

App Testing

[redacted]

[redacted]

Huawei Mate 10 and android 10.
App didn't find the google account as I was warned.
Didn't knew about the thing with the call list, that's scary.

Figure 15. Confirmation email from user #1

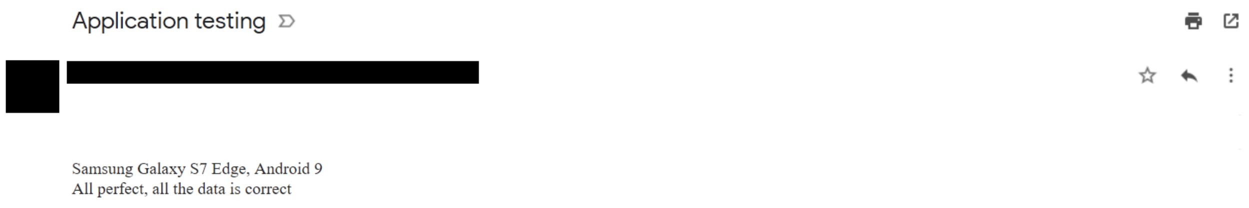

app test

[redacted]

[redacted]

Android pie, Oneplus GM1917
Message about the failure to launch the application, an app turns off immediately after pressing on it.

Figure 16. Confirmation email from user #2

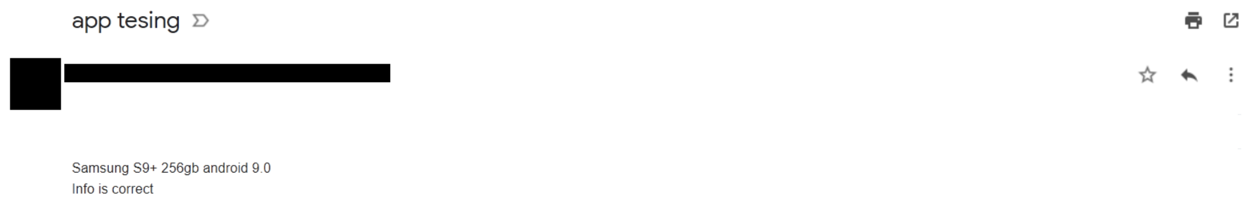

Application testing

[redacted]

[redacted]

Samsung Galaxy S7 Edge, Android 9
All perfect, all the data is correct

Figure 17. Confirmation email from user #3

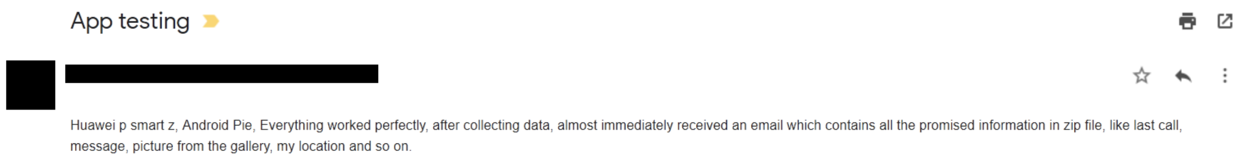

app tesing

[redacted]

[redacted]

Samsung S9+ 256gb android 9.0
Info is correct

Figure 18. Confirmation email from user #4

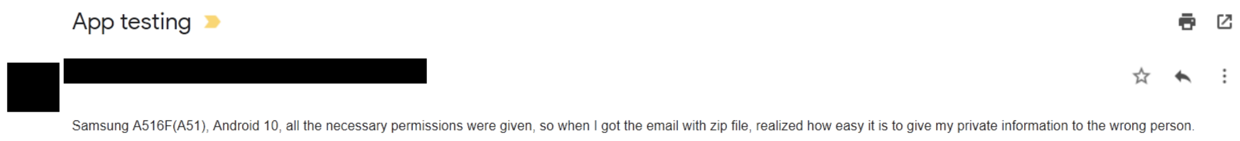

App testing

[redacted]

[redacted]

Huawei p smart z, Android Pie, Everything worked perfectly, after collecting data, almost immediately received an email which contains all the promised information in zip file, like last call, message, picture from the gallery, my location and so on.

Figure 19. Confirmation email from user #5


App testing

[redacted]

[redacted]

Samsung A516F(A51), Android 10, all the necessary permissions were given, so when I got the email with zip file, realized how easy it is to give my private information to the wrong person.

Figure 20. Confirmation email from user #6

4.4 Future work

Like other software projects, this system can be improved. There are several important improvements that will make the system more viable, for example, implementation of platform independence. The system should work the same on all android platforms, regardless of manufacturer or OS fork. Also, it makes sense to consider other more common rights options, such as access to a phone camera or microphone. Furthermore, it is rational to add other popular languages besides English for better comprehensibility to the broad masses of people.

Conclusion

Obtaining rights is a fundamental process of integrating software and its functionality onto the target device. From the developer's point of view, calling the dialog to obtain the necessary permissions is created by adding a dozen lines to the manifest, and, in fact, can be done by simple copying from the manuals.

The OS does not have internal mechanisms of protection, and relies on the postulate that the user understands what he is doing and why. Google's app store has an app validation mechanism, but all devices support the ability to install apps from third-party sources.

Since from the developer's point of view, integrating even potentially malicious code is not that difficult, the security of personal data remains on the user's conscience. If the user does not read the end-user license agreement, then nothing else remains but to clearly demonstrate the danger of an imprudent, thoughtless and simply negligent attitude both to the device and various applications.

As a result of this thesis, a prototype of the Android application was created. This application can be used as an illustration to the users of smartphones to show them how important it is to be responsible when distributing personal information and give some facts about common rights in android using a practical example.

References:

[1] Mobile Operating System Market Share Worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide
(06.04.2020)

[2] Google play annual app downloads 2019. https://www.statista.com/statistics/734332/google-play-app-installs-per-year
(10.08.2020)

[3] Pulse Secure MTC. Mobile threat report, 2015.
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjL2OagpZHrAhUP66QKHUKQD0EQFjAAegQIBhAB&url=https%3A%2F%2Fwww.pulsesecure.net%2Fdownload%2Fpages%2F2819%2F&usg=AOvVaw1CGBSTsI4K_WwcKb5_Pwmj
(10.08.2020)

[4] Mobile Insecurity. https://www.csa.gov.sg/singcert/publications/mobile-insecurity
(06.04.2020)

[5] Pokémon Go can see everything in your Google account. Here's how to stop it.
https://www.cnet.com/how-to/pokemon-go-google-account-access/
(06.04.2020)

[6] Android Developers Docs. Guides: App permissions, Overview.
https://developer.android.com/guide/topics/permissions/overview#additional-resources
(14.01.2020)

[7] Android security by example. https://www.slideshare.net/pragatiogal/android-securitybyexample-22139131
(10.08.2020)

[8] Isikuandmete kaitse seadus. § 4. Isikuandmed. https://www.riigiteataja.ee/akt/862756
(12.05.2020)

[9] General Data Protection Regulations. Art. 4 Definitions. https://gdpr-info.eu/art-4-gdpr
(12.05.2020)

[10] Android Developers Docs. Guides: App permissions, Request app permissions.
https://developer.android.com/training/permissions/requesting
(14.01.2020)

[11] Android Developers Docs. Guides: App permissions, App permissions best practices. https://developer.android.com/training/permissions/usage-notes
(14.01.2020)

[12] Lin, Jialiu, Liu, Bin, Sadeh, Norman, Hong, Jason I. Modeling Users’ Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. School of Computer Science, Carnegie Mellon University, SOUPS 2014, Melano Park, CA.

[13] How Game Apps That Captivate Kids Have Been Collecting Their Data. https://www.nytimes.com/interactive/2018/09/12/technology/kids-apps-data-privacy-google-twitter.html
(01.05.2020)

[14] Утечка персональных данных: последствия. https://searchinform.ru/resheniya/biznes-zadachi/zaschita-personalnykh-dannykh/utechki-personalnyh-dannyh/posledstviya/
(01.05.2020)

[15] Android Developers Docs. Distribution Dashboard. https://developer.android.com/about/dashboards
(17.07.2020)

Appendix

I Additional files and application installation guide

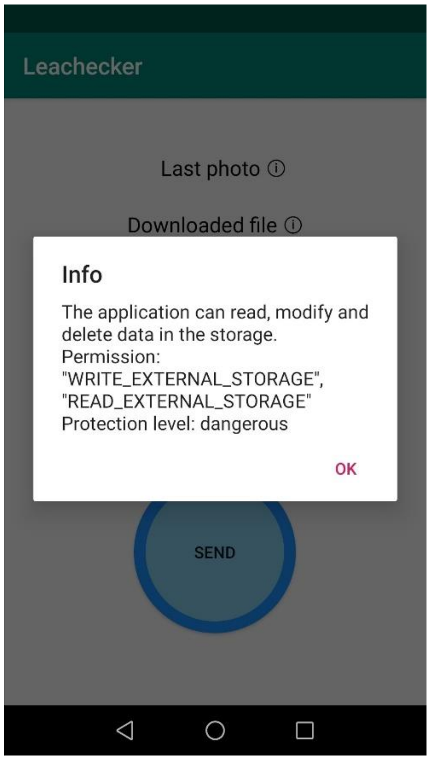
The zip file of the additional files (appendix.zip) contains:

1. Android application package (apk file)
2. Application source code (DataCollection folder)

Installing the application:

1. Unzip the additional files zip file.
2. Transfer the apk file to the smartphone.
3. Launch the file from the smartphone.
4. Follow the instructions of the installation wizard.

II Images of the application



Leachecker

Last photo ⓘ

Downloaded file ⓘ

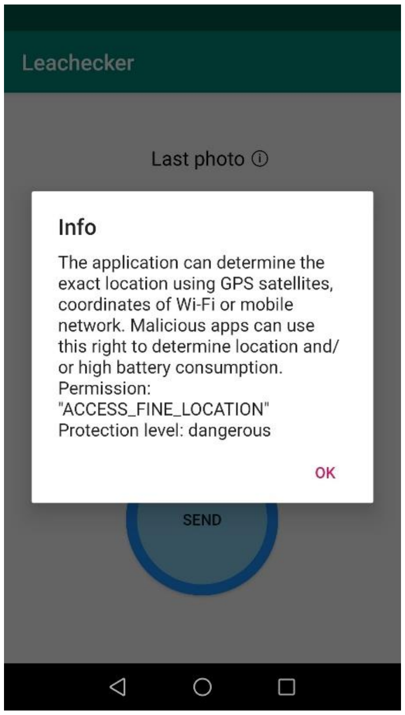
Info

The application can read, modify and delete data in the storage.
Permission:
"WRITE_EXTERNAL_STORAGE",
"READ_EXTERNAL_STORAGE"
Protection level: dangerous

OK

SEND

Figure 21. Last photo and downloaded file permissions info



Leachecker

Last photo ⓘ

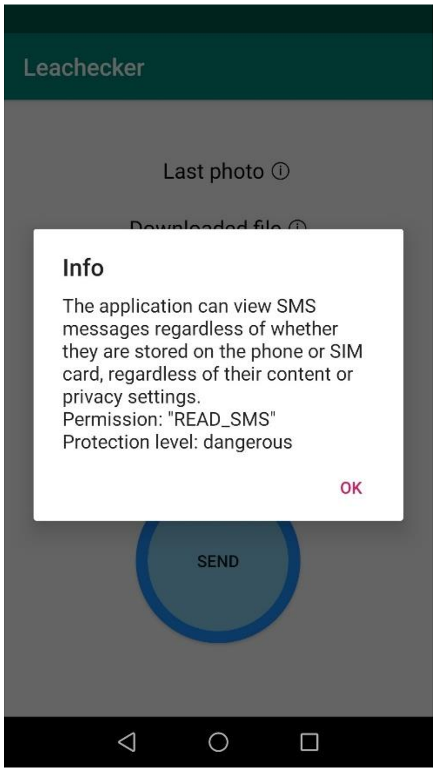
Info

The application can determine the exact location using GPS satellites, coordinates of Wi-Fi or mobile network. Malicious apps can use this right to determine location and/or high battery consumption.
Permission:
"ACCESS_FINE_LOCATION"
Protection level: dangerous

OK

SEND

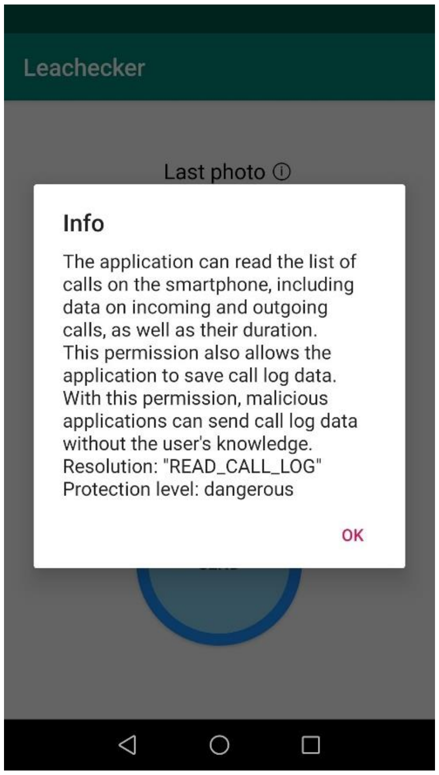
Figure 22. Location permission info



Info

The application can view SMS messages regardless of whether they are stored on the phone or SIM card, regardless of their content or privacy settings.
Permission: "READ_SMS"
Protection level: dangerous

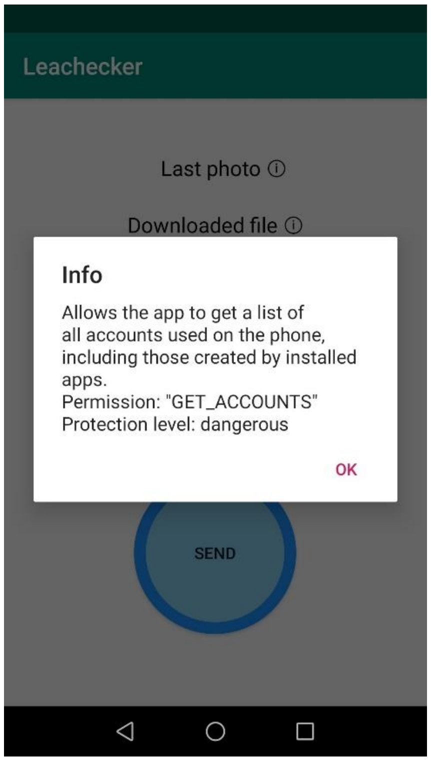
Figure 23. SMS permission info



Info

The application can read the list of calls on the smartphone, including data on incoming and outgoing calls, as well as their duration. This permission also allows the application to save call log data. With this permission, malicious applications can send call log data without the user's knowledge.
Resolution: "READ_CALL_LOG"
Protection level: dangerous

Figure 24. Calls permission info



Info

Allows the app to get a list of all accounts used on the phone, including those created by installed apps.
Permission: "GET_ACCOUNTS"
Protection level: dangerous

Figure 25. Accounts permission info

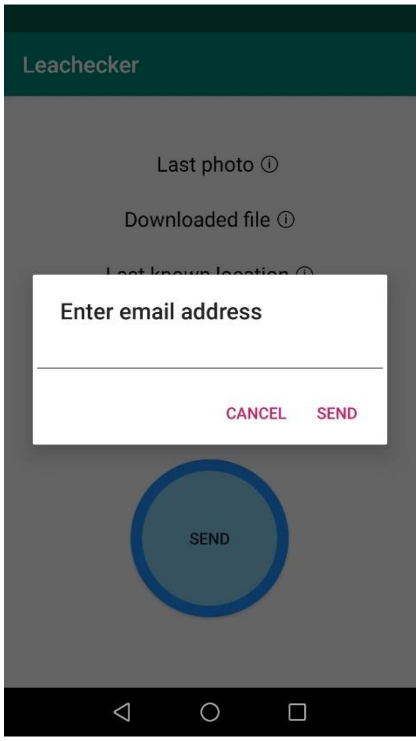
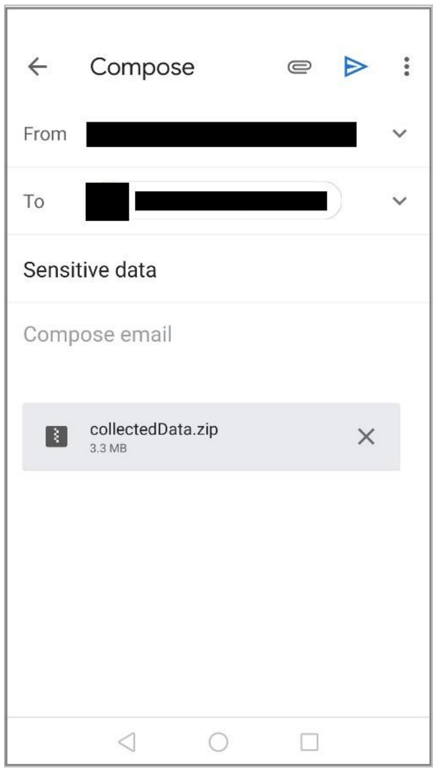


Figure 26. Sending email screen



From [redacted]

To [redacted] [redacted]

Sensitive data

collectedData.zip
3.3 MB

Figure 27. Composed email

III License