# Storage and retrieval mechanisms in mobile spam blocking applications

V.G.J. de Jong
Supervisor(s): Dr. Apostolis Zarras, Dr. Yury Zhauniarovich
EEMCS, Delft University of Technology, The Netherlands

June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

**Abstract**

Applications on Android phones which block spam calls could be argued a necessity for people unfortunate enough to have their number on spam lists. Third-party applications provide extensive, up-to-date blocklists to screen incoming calls. This paper analyses and describes how these types of applications store and access their data. First we analyse which applications keep their data offline on disk, then we take a look at applications which use the internet to query an incoming phone call. We found four applications use a combination of both, two applications using exclusively online resources and four applications using offline resources only.

# 1 Introduction

Spam calls are a common frustration for many people. According to Hiya[1], on average a person receives 16 scam calls a month. After the proliferation of mobile phones in the past decade, they are often seen as a personal device with the phone number often only being known to parties involved in the life of the person. Therefore, it is hard to imagine that an incoming spam call will not be answered. A scam call does not bring any value to the callee, in contrast, it may be a source of anxiety and expenses. What is even more frightening, in 75% of cases, scam callers knew some personal information about their victims[3]. Moreover, by spoofing the Caller ID information, they trick their victims to believe that the call is authentic. For instance, 32% of all victims who lost more than $1000 due to scam calls reported that they had received a call from a known business, e.g., their bank[3]. Not surprisingly, unwanted calls is the number one source of complaints to the Federal Communications Commission (FCC) and Federal Trade Commission (FTC) in the United States[1]. Solutions to this problem are popular, as can be seen by the millions of downloads apps like Hiya have according to the Google play store[10].

The research question posed in this paper answers how several popular applications store and retrieve the information available and how they differ from one another. This paper will investigate those mechanisms and highlight differences between choices made, as well as their positives and negatives.

# 2 Background

Spam call blockers function similarly to other android applications. Android applications are commonly written in Kotlin, which is in turn build on Java and compiled to a special variant of the Java bytecode, which is able to run on the Android Runtime (ART). The ART provides several advantages for Android applications such as a custom garbage collection algorithm and specialized debugging tools. Android applications are modified on install time using the Ahead of Time compiler to optimize the code[2] and uses a Just-In-Time compiler to optimize further during runtime[2].

Where spam call blocking applications usually differ from other Android applications is in the Application Programming Interface (API) used. These applications tend to hook into the phone and SMS services in order to facilitate their advertised blocking features. An example of this can be seen in figure 1, where Truecaller asks for permission to manage phone calls.

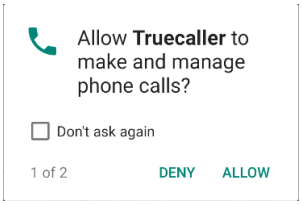


**Figure 1:** Truecaller asking for permissions

An incoming call will be processed by the spam blocking application even before it displays the call screen and rings the phone. This allows the blocking application to make the appropriate decision based on the data it has whether to allow the call to go through or to block it. Some applications overlay the regular incoming call screen with their own custom call screen. This allows an application to verbosely warn the user of any malicious activity.

# 3 Methodology

This section of the paper will describe the tools and setup used to perform the required analysis for answering the research question for each application, as well as referencing a list of the analysed mobile applications.

### 3.1 Analysed applications

The list of applications is used in the research of this paper can be found in Appendix A. These applications are selected as being some of the most popular at the time of writing this paper. The Android Package Kits (APKs) are obtained through an online service such as Apkcombo[19]. The downloaded APK files can be found alongside the instructions in the Gitlab repository [16].

### 3.2 Static analysis

The static analysis part of this research covers the inspection of the source code of the applications listed in Appendix A. For example, this can mean reading through the code to find specific methods which cover functionality offered by an application. Tools such as apktool, dex2jar and jadx[20] can convert an Android application to its source code and allow it to be analysed. Static analysis allows us to find parts of the code which load or retrieve the information used to evaluate spam calls. In section 4 we will discuss why this method was not chosen as the focus for this paper.

### 3.3 Dynamic analysis

For the dynamic analysis part we take the applications and run them in a controlled environment with several tools which allows us deeper insight into the workings of the application. This method runs the applications with full functionality as they would on a phone in a real world environment. As a result, we can allow applications to work as intended and investigate or extract information from the workings of the applications.

With dynamic analysis, instead of analysing the code as you would with static analysis, we analyse the behaviour of the applications and the artifacts they produce. Using a man-in-the-middle attack we can intercept and read the network traffic produced by an application The built-in file explorer of Android Studio allows us to analyse the files stored by the application on the filesystem.

## 4 Comparison of analysis methods

Using dynamic analysis instead of static analysis has several advantages for the research question described in this paper. First, as every application is made differently by various people using distinct tools and frameworks, there are significant differences in source code and structure. Furthermore, several applications have a large code base, reading and understanding the source code is a serious undertaking.

Using dynamic analysis on the other hand, yields tangible results rapidly. A basic setup using only Android Studio already allows the retrieval of databases on disk, which might store information used to block spam calls. Of course, not all application (only) use databases on disk and therefore this method only is inadequate.

Often applications use the internet to communicate findings and query data. To analyse this data, tools such as MITMtool can be used to this end. Several applications make use of an internet connection to evaluate whether an incoming call is a spam call. Many people in the target audience of these applications have a data plan, according to Statista [5] and it seems application developers make use of this fact. In many cases applications make network calls to an endpoint with for example the time, location and phone number of the incoming call and receive information about the number.

Getting the same results using static analysis is prohibitively more work. It takes careful reconstruction of each step in the code and likely partial reverse engineering of the endpoint API to get similar results, if possible at all. Many apps retrieve an API key during runtime or ask you to login first, which further complicates the reverse engineering. In such cases dynamic analysis shows its strengths and advantages.

## 5 Setup and results

### 5.1 Static analysis

Using the setup described in the project repository[18] results in readable source code which can be analysed by hand or using tools. However, it should be noted, this method results in code which cannot be compiled back into a usable APK to install and run on a device or emulator. This is a limitation of dex2jar and jadx. To generate source code which can be recompiled,

apktool is run with different option flags on the command line. This results in Smali code, which is a human-readable representation of the bytecode run on the ART. This Smali code can, if so desired, be recompiled back into an APK and installed on a machine. This allows modifications to the Smali code to be run and tested on the Android emulator.

## 5.2 Dynamic analysis

For dynamic analysis, the official Google emulator[28] is used to run each application like it would in a real world environment. The emulator offers a complete integration and mocks several features like internet or WiFi, SMS, calling and other services by using virtual sensors and a virtual SIM card. The emulator allows interaction with the system through keyboard and mouse, which is automatically mapped to the corresponding input for Android.

Android Debug Bridge (ADB)[29] is used to interact with the emulator through the command line and allows scripting to aid analysis.

Each application analysed is first installed into a clean install of the emulator, since several apps conflict with each other and will ask you to delete the other app first before they will function as intended. Applications can easily be installed using the ADB, provided the APK has been obtained.

After the application has been installed, it can be interacted with as if it were installed on a smartphone and several steps for dynamic analysis can already be taken:

1. Use the integrated calling functionality of the emulator to emulate a call using a known scamming phone number. If the application detects the call without having to add the number to the applications blocklist manually, the application has some sort of means to detect spam calls. Numbers for testing purposes can be found in the paper's GitLab project under the section dynamic-analysis[17].

2. The files the application works with are saved on disk and can be accessed using Android Studio's Device File explorer. Usually, files used by an application are stored in the `/data/data/com.application.name` directory.

### 5.2.1 Analysis of on-disk databases

After extracting an applications' files using the method described above, we can analyse the databases the application uses (if any) by investigating the SQLite databases found in the directory `databases/` as can be seen in figure 2.

```
[victor@victor-laptop database]$ file offlinedata.sqlite
offlinedata.sqlite: SQLite 3.x database, last written using SQLite ver
sion 3007017, page size 1024, file counter 1, database pages 8120, coo
kie 0x1, schema 4, UTF-8, version-valid-for 1
```

**Figure 2:** File type detection shows the Android application's databases are SQLite 3.x

Using a visual database tool such as SQLitebrowser[30] we can investigate databases and determine their purpose. For several apps it can be determined these use offline means of evaluating incoming calls as these store databases with many numbers and include information on these numbers. For example what spam call category they belong to. An excerpt from ShowCaller's database can be seen in figure 3, which shows information about known spam call numbers.

| | ID | tel_number | operator | belong_area | type | name |
|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | 1 | 0f539e4ab... | Telecommunications | India | Fixed line Mobile | Likely Telemarketers |
| 2 | 2 | daf137ae9... | Telecommunications | India | Fixed line Mobile | Airtel |
| 3 | 3 | fa536aa90... | Telecommunications | India | Fixed line Mobile | Airtel |
| 4 | 4 | 1e9e8995... | Vodafone Idea | Gujarat, India | Mobile | Navri Company |
| 5 | 5 | 3127981d... | Vodafone Essar Ltd. | Maharashtra, India | Mobile | Nik |

**Figure 3:** Excerpt from ShowCaller's offlinedata.sqlite database. Phone numbers are hashed.

### 5.2.2 Analysis of network traffic

Several apps are querying the data necessary to evaluate incoming calls on demand, thus a setup is necessary to intercept and log calls over the network. Fortunately, all applications discussed in this paper use secure means of communication (HTTPS/TLS) to send data, which might contain personal information. However, this does mean the traffic is encrypted and thus not immediately readable for analysis. The setup needs to support intercepting and decrypting network traffic.

The tool used in this paper for this setup is MITMproxy[31], a popular tool which supports all requirements listed above. MITMproxy uses a self-signed root certificate to be seen as a trusted endpoint to which the application can connect. The tool is able to decrypt TLS traffic, log it and

forward it to the actual destination.

To properly use this tool, a more sophisticated setup is needed to configure the Android emulator to properly connect through the proxy without certificate errors.

The man-in-the-middle setup in this paper was configured using the steps provided in the dynamic analysis section of the project repository[17] and is largely based on the documentation of MITMproxy, as the tool provides an excellent guide on how to configure TLS decryption on Android machines[32]. The setup in this paper was constructed on an operating system based on the Linux kernel and an Android emulator with API level 28. For the **CallerID Block** application an Android emulator with API level >28 is necessary.

All traffic from the emulator is routed through the host machine and with MITMproxy full TLS decryption and traffic inspection is possible.

#### 5.2.3 Resistance against network analysis: certificate pinning

Some applications perform additional checks to ensure the endpoint they are connecting to is legitimate. In this case applications use a hardcoded entry which contains the hashes of allowed TLS certificates and will only establish a successful connection after the certificate hash has been verified to be valid.

In this paper, **CallApp**, employs such a method. This means further setup is needed before we are able to inspect the network traffic from and to this app.

The first attempt at circumventing the pinning was based on searching and replacing the hardcoded hashes in the source code with a hash of MITMproxy's custom root certificate. This approach did not seem to work. A second attempt was made using Frida[24], which provides a dynamic method hook approach without requiring modifications to the source code.

This tool can be used to inject custom scripts into a process and hook functions using a server component[25] installed on the Android emulator. Using a custom script[26] provided by HTTP-toolkit, which hooks common implementations of certificate pinning, we are able to disable this feature for CallApp. The exact steps taken to accomplish this can be found in the project repository under dynamic analysis[17].

## 6 Results

The results obtained after analysing the data retrieved with the lab setup are divided into two parts. The application stores the data used to detect spam calls in structured databases on disk or the application queries an online resource for each call and provide online screening. In several cases applications use both methods, likely to provide protection in case of no internet access. In figure 4, an overview of the results is shown.

| Application | On disk | Internet |
|---|---|---|
| BelControl | ✓ | ✓ |
| CallApp Contacts | ✓ | ✗ |
| CallBlocker | ✓ | ✗ |
| CallerID Block | ✓ | ✓ |
| Hiya | ✗ | ✓ |
| Should I answer? | ✓ | ✗ |
| ShowCaller | ✓ | ✓ |
| StopMijBellen | ✓ | ✗ |
| TelGuarder | ✗ | ✓ |
| TrueCaller | ✓ | ✓ |

**Figure 4:** Table of the results for each application

Of all applications analysed in this paper, only `TelGuarder` and `Hiya` do not seem to use offline methods of detecting possible spam calls.

From these results it is already clear several apps use both sources on disk and from the internet to evaluate an incoming call. It is unknown whether these are used in tandem or whether one method is preferred over another depending on the circumstances. In the following subsections, specific findings for each application are shown.

## 6.1 BelControl

The on disk database is quite limited with only 38 000 entries and not really interesting. The network calls on the other hand seem to be very verbose, and while the contents were binary encoded and not readable, the URLs are.

```
https://www.everycaller.com/api/android/41/en_US/lookup/
https://www.everycaller.com/api/android/41/en_US/get_caller/
https://www.googleapis.com/plus/v2whitelisted/people/lookup
```

From the API structure it seems likely some information on an incoming call is requested. The last call to googleapis.com is readable, the request and response is structured as follows:

```
# These are parsed query paramaters, there is no payload
includePeople: 1
includeGal:    1
type:          phone
fields:        kind,items(id,metadata(objectType,plusPageType,attributions),names,phoneNumbers(value,type,formattedType,canonicalizedForm),addresses(value,type,formattedType),images(url,metadata(container)),urls(value),placeDetails,extendedData)
includePlaces: 1
callType:      incoming
id:            [REDACTED] # Phone number
```

The following response is received.

```
{
    "kind": "plus#peopleList"
}
```

As in this request the phone number was passed as an argument. It is plausible information on the number is requested, although not verifiable in the response.

## 6.2 CallApp

CallApp uses a small database on disk containing United States area codes. This database has around 190 000 entries. No phone numbers were found in this database, it seems this database purely aids in displaying CallerID[34].

After disabling the certificate pinning CallApp is using, there are no calls to the internet observed. This is allegedly a feature of the premium tier.

## 6.3 CallBlocker

CallBlocker does not seem to offer much in terms of protection against spam callers. Its offline database has very few entries, about 582 and does not provide additional functionality by querying online resources.

## 6.4 CallerID

CallerID uses a modest database of around 20 000 entries and queries API endpoints on each incoming call. This applications seemingly uses a similar binary encoding like BelControl. The following API endpoints were queried.

```
https://app.ayamote.com/api/v1/sea.php
https://app.ayamote.com/api/v1/searep.php
https://ct.ayamote.com/c_n/api/v1/cnwik.php
```

However, this API is formed differently and uses the `application/x-www-form-urlencoded` content-type. This allows us to view the key, even if most of the data remains unreadable.

The following request is an example.

```
cc:          44
uid:         25eba4914c851913d122351889968f1d
tel_number:  K0S3Rzh4RHczPmM1NC==
app_version: 1.6.5
stamp:       9eae757ee95fc7bcf3d835e2b862ab98
device:      sdk_gphone_x86_64_arm64
default_cc:  1
platform:    android
```

With the following response.

```
ezL2hGG2gbMjQl41NEzmdHXvb263eaJmelM6JkLwInbygm2jgJ91bZ1fcpYxYnX1MjpkLmwjdpJtaULAIjKvMmG4bbRielM6JkLwInBzdXKjgK9zKmsiJkzmdInzdSJ8LmItKqV5dIYjbHHldWxkRmIjNFNmZYowZGBognKxfp9td5giPlDwIoTsfWWhf7RidaEiPkLmLDLwg2dkRmJUbZJyZ4jkRnHsfGRkiU==
```

While it would seem at first glance the data is base64 encoded, decoding does not yield readable data.

## 6.5 ShouldIAnswer

ShouldIAnswer uses an offline database to detect spam calls, which is downloaded at startup from the following endpoint:

```
https://aapi.shouldianswer.net/srvapp/
↪ get-database/cached?_dbVer=1991
```

The file that is downloaded as a result, `cached.gzip`, unfortunately does not seem to reveal a parsable database after extraction. However, the file is 12 megabytes and will likely provide data on 100 000 - 200 000 phone numbers if we extrapolate data sizes in relation to the other applications in this paper.

ShouldIAnswer does not perform extra queries to online resources whenever an incoming call is made.

## 6.6 ShowCaller

ShowCaller has access to an `offlinedata.sqlite` database on disk which provides information on about 50 000 phone numbers.

When an incoming call is detected, ShowCaller performs several calls to the internet, all of which use the same binary as found with CallerID. To the following API endpoints, similar `application/x-www-form-urlencoded` content-type requests are made.

```
1 https://app.show-caller.com/api/v1/
  ↪ sea.php
2 https://app.show-caller.com/api/v1/
  ↪ cheact.php
3 https://app.show-caller.com/api/v1/
  ↪ seacomcousub.php
```

The following request is an example.

```
cc:          1
uid:         11093cb4f57c99e5fe68f6a92393cf3e
tel_number:  00447868726250
is_contacts: 1
stamp:       103c6c8d8c6f6cf54fcb50e31971f78d
device:      android
version:     2.2.5
default_cc:  1
cid:         1
```

This request produces a similar response as the one made by CallerID. However, in contrast with CallerID it seems ShowCaller does not apply the encoding to the request, as the parameters like the phone number are plain-text. The response is similar to the response from CallerID.

## 6.7 StopCall

StopCall downloads their entire database on startup from their API endpoint. It contains around 130 000 entries in total and the following endpoints are observed on startup.

```
1 https://srv22.mglabapps.host/app2/
  ↪ app_device.php
2 https://srv22.mglabapps.host/app2/
  ↪ app_download.php
```

No network calls are being made when an incoming call is detected.

## 6.8 TelGuarder

TelGuarder is the first application which takes an all online approach. No offline data could be found, but TelGuarder queries each incoming call on its API endpoints. Such requests are made to the following API endpoint:

```
https://tgedgeapi.telguarder.com/v2/Search/
↪ NumberLookup/Mobile
```

The request looks like this.

```
{
    "keepProportions": true,
    "logoHeight": 132,
    "logoWidth": 132,
    "numbers": [
        "00447868726250"
    ],
    "resizeLogo": true
}
```

The response to this request is large and can be found here.

Furthermore, it seems TelGuarder has a large, community driven system where users mark numbers as spam themselves and can even leave comments. These comments are then queried from the following endpoint:

```
https://api.advista.no/Report/Number/
↪ Comments/Mobile
```

An example of a request and response to retrieve these comments can be found here.

### 6.9 TrueCaller

TrueCaller uses its own APIs to query for each incoming call. Such requests are made to the following endpoint. Note the presence of the incoming phone number on the second line.

```
https://search5-eu.truecaller.com/v2/
↪ search?q=447868726250&countryCode=NL&
↪ type=2&placement=CALLERID%2CAFTERCALL%
↪ 2CDETAILS&adId=f27b7cdd-660e-473a-95f8-
↪ 839885ec1465&encoding=json
```

A large amount of information is returned, an example of a full response can be found here.

But what makes TrueCaller the most interesting application so far is the offline detection method present. In `insights.db` there is a table called `catagorizer_probability` which contains keywords which are assigned probabilities. TrueCaller seemingly uses these keywords and probabilities to automatically detect incoming SMS spam messages without an online lookup. TrueCaller is the only application analysed which offers such functionality. This could not be tested as this is marketed as a premium feature. Figure 5 shows several entries in the `catagorizer_probability` table.

| | word | probHam | probSpam | tfHam | tfSpam | idfHam | idfSpam |
|---|---|---|---|---|---|---|---|
| | Filter | Filter | Filter | Filter | Filter | Filter | Filter |
| 1 | sms | 0.0012857449114812 | 0.00332960227495549 | 5861.0 | 13151.0 | 5861.0 | 13151.0 |
| 2 | pct | 0.00104617863095847 | 0.0109056683091967 | 10549.0 | 95284.0 | 10549.0 | 95284.0 |
| 3 | store | 0.000269039927615058 | 0.00321511030323056 | 1074.0 | 11123.0 | 1074.0 | 11123.0 |
| 4 | lens | 2.2241265828136e-05 | 0.00128976805619092 | 65.0 | 3276.0 | 65.0 | 3276.0 |
| 5 | above | 0.000862699408225487 | 0.00126206052607901 | 2949.0 | 3738.0 | 2949.0 | 3738.0 |
| 6 | com | 0.00309433298578743 | 0.00459983902018716 | 19292.0 | 24848.0 | 19292.0 | 24848.0 |
| 7 | voucher | 0.000176124152431471 | 0.00156129037045695 | 559.0 | 4295.0 | 559.0 | 4295.0 |
| 8 | lenskart | 3.36219173063148e-05 | 0.00152535730962088 | 103.0 | 4057.0 | 103.0 | 4057.0 |

**Figure 5:** Snippet from TrueCaller's database table with probabilistic entries

### 6.10 Hiya

Hiya does not use any sort of offline method for detecting incoming spam calls, but just like TelGuarder it is exclusively online. Each incoming call is screened by Hiya on their own API endpoints. The following endpoints were found.

```
1 https://auth.edge.hiyaapi.com/v2/auth/
  ↪ token
2 https://callerprofile.edge.hiyaapi.com/
  ↪ v3/phone_numbers/eventProfile
```

Hiya immediately queries their APIs with the following request on an incoming call.


```
{
    "event": {
        "direction": "Incoming",
        "isContact": false,
        "phone": {
            "meta": {
                "countryCode": "US",
                "isShortCode": false,
                "isValid": false,
                "parserVersion": "5.0.2",
                "rawPhone": "00447868726250"
            },
            "phone": "1/00447868726250"
        },
        "timestamp":
        ↪ "2022-05-02T16:11:26.559Z",
        "type": "EventProfileCallEvent"
    },
    "profileScope": {
        "identity": true,
        "registered": true,
        "reputation": true
    }
}
```


The following response was received.


```
{
    "attribution": {
        "attributionImageUrl": ["REDACTED"],
        "attributionName": "Hiya",
        "attributionUrl": ["REDACTED"]
    },
    "callId": ["REDACTED"],
    "displayName": "Suspected Spam",
    "profileDetails": {
        "entityType": "UNKNOWN",
        "lineTypeId": "other"
    },
    "profileIcon": "WARN",
    "profileTag": ["REDACTED"],
    "reputationLevel": "SPAM",
    "verified": false
}
```


Several parts were redacted to keep the response compact. The full response can be viewed here.

## 7 Responsible research

The source material used in this paper is in all cases copyright restricted and care is taken into respecting the terms of service. When interacting

with external 3rd parties, no action was performed with malicious intent.

# 8 Discussion

While the results from the previous section cover the storage location of the data, it does not discuss the availability, quality or effectiveness of the data and their implementations in the applications.

Rough metrics about the size of the data, the presence of additional network functionality and amount of record entries in database are given in this paper, but these might not reflect the actual performance of an application in a real world setting. Further research can take in account and compare the data between apps to gauge effectiveness and discuss whether installing a 3rd party application is worth it.

All the applications analysed in this paper are available free of charge and as a consequence they transmit large amounts of data to advertising endpoint, which can be observed in the captured network traffic in the projects repository [16]. Further research might delve deeper into this subject and analyse how personal data is used by these applications.

# 9 Conclusion

The analysis of where each application stores its data was successful for each of the applications researched. Quite a large range of the amount of data available to each application was observed, as some application have access to significant more data and methods than others.

It is interesting to see some applications use a trusted approach of both offline and online databases, which ensures up-to-date information when connected to the internet, while retaining functionality when offline. However, some applications go all in on online data-sets only, seemingly expecting the user to be connected to the internet when the cell phone has reception. Other applications only use offline databases, but these may under-perform compared to their online counterparts as offline databases are not updated as often and do not provide the flexibility of online databases.

TrueCaller even managed to implement some probabilistic method for detecting spam/scam SMS messages, a method which no other application uses. It is interesting to see applications go to such lengths to perform their task. While several applications analysed definitely fall in this category, others lack a large dataset or online features.

As noted in the discussion section, the effectiveness of the data is not taken into consideration in this paper.

# References


[1] Hiya, "State of the Phone Call: Half Yearly Report 2019" (2019), https://assets.hiya.com/public/pdf/HiyaStateOfTheCall2019H1.pdf?v=6b7b682837c56c47656c012c1da0e6a0

[2] Android Runtime (ART) and Dalvik. (2020). Android Open Source Project. https://source.android.com/devices/tech/dalvik#AOT_compilation

[3] First Orion, "Scam Call Trends and Projections Report", Summer 2019, http://firstorion.com/wp-content/uploads/2019/07/First-Orion-Scam-Trends-Report_Summer-2019.pdf

[4] CTIA, "Robocall Abatement Apps for Android Devices" (2020), https://api.ctia.org/wp-content/uploads/2020/01/robocall-resources-for-android-2020.pdf

[5] Ceci, L. (2022, February 22). Mobile internet usage worldwide - statistics & facts. Statista. Retrieved June 19, 2022, from https://www.statista.com/topics/779/mobile-internet/

[6] Call Control - SMS/Call Blocker. Block Spam Calls! - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.flexaspect.android.everycallcontrol

[7] CallApp: Caller ID & Recording - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.callapp.contacts

[8] Call Blocker - Stop spam calls - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.unknownphone.callblocker

[9] Caller ID, Phone Dialer, Block - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.callerid.block

[10] Hiya - Call Blocker, Fraud Detection & Caller ID - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.webascender.callerid

[11] Should I Answer? - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=org.mistergroup.shouldianswer

[12] Stop Calling Me - Call Blocker - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.mglab.scm

[13] Showcaller: Caller ID & Block - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.allinone.callerid

[14] Spam Call Blocker - telGuarder - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.telguarder

[15] Truecaller: Caller ID & Block - Apps on Google Play. Google Play Store. Retrieved June 19, 2022, from https://play.google.com/store/apps/details?id=com.truecaller

[16] de Jong, V. G. J. (2022, June 19). Victor De Jong / SpamCallBlockingApps-Analysis. GitLab. Retrieved June 19, 2022, from https://gitlab.com/Vgjdejong/spamcallblockingapps-analysis

[17] de Jong, V. G. J. (2022, June 19). Victor De Jong / SpamCallBlockingApps-Analysis. GitLab. Retrieved June 19, 2022, from https://gitlab.com/Vgjdejong/spamcallblockingapps-analysis/-/tree/main/dynamic-analysis

[18] de Jong, V. G. J. (2022, June 19). Victor De Jong / SpamCallBlockingApps-Analysis. GitLab. Retrieved June 19, 2022, from https://gitlab.com/Vgjdejong/spamcallblockingapps-analysis/-/blob/main/applications/README.md

[19] APK Downloader - Download APK for Android [2022]. (2022). APKCombo.Com. Retrieved June 19, 2022, from https://apkcombo.com/apk-downloader

[20] Arnatovich, Y. L., Wang, L., Ngo, N. M., & Soh, C. (2018). A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. IEEE Access, 6, 12382-12394.

[21] Tumbleson, C. (n.d.). GitHub - iBotPeaches/Apktool: A tool for reverse engineering Android apk files. GitHub. Retrieved June 19, 2022, from https://github.com/iBotPeaches/Apktool

[22] Pan, B. (n.d.). GitHub - pxb1988/dex2jar: Tools to work with android .dex and java .class files. GitHub. Retrieved June 19, 2022, from https://github.com/pxb1988/dex2jar

[23] GitHub - skylot/jadx: Dex to Java decompiler. (n.d.). GitHub. Retrieved June 19, 2022, from https://github.com/skylot/jadx

[24] Frida. (2022, February 25). Frida - A World-Class Dynamic Instrumentation Framework. Retrieved June 19, 2022, from https://frida.re/docs/android/

[25] Releases - frida/frida. (2022, June 3). GitHub. Retrieved June 19, 2022, from https://github.com/frida/frida/releases

[26] GitHub - httptoolkit/frida-android-unpinning: A Frida script to disable SSL certificate pinning in a target application. (n.d.). GitHub. Retrieved June 19, 2022, from https://github.com/httptoolkit/frida-android-unpinning

[27] Download Android Studio and SDK tools. (n.d.). Android Developers. Retrieved June 19, 2022, from https://developer.android.com/studio

[28] Run apps on the Android Emulator. (n.d.). Android Developers. Retrieved June 19, 2022, from https://developer.android.com/studio/run/emulator

[29] Android Debug Bridge (adb). (n.d.). Android Developers. Retrieved June 19, 2022, from https://developer.android.com/studio/command-line/adb

[30] GitHub - sqlitebrowser/sqlitebrowser: Official home of the DB Browser for SQLite (DB4S) project. GitHub. Retrieved June 19, 2022, from https://github.com/sqlitebrowser/sqlitebrowser

[31] mitmproxy - an interactive HTTPS proxy. (n.d.). MITMproxy. Retrieved June 19, 2022, from https://mitmproxy.org/

[32] System CA on Android Emulator. (n.d.). MITMproxy. Retrieved June 19, 2022, from https://docs.mitmproxy.org/stable/howto-install-system-trusted-ca-android/

[33] System CA on Android Emulator. (n.d.). MITMproxy. Retrieved June 19, 2022, from https://docs.mitmproxy.org/stable/howto-install-system-trusted-ca-android/#1-prerequisites

[34] Wikipedia contributors. (2022, April 8). Caller ID. Wikipedia. Retrieved June 19, 2022, from https://en.wikipedia.org/wiki/Caller_ID

[35] Oracle. (n.d.). OpenJDK. OpenJDK. Retrieved June 19, 2022, from https://openjdk.org/

[36] Python Software Foundation. (2022, June 9). Python. Python.Org. Retrieved June 19, 2022, from https://www.python.org/

# Appendices

## A List of analysed applications

1. Bel Control [6], sha256: 424d796c267812e91cdabd161f4796d99d1333d673f4f13ec417114a156b180d
2. CallApp [7], sha256: 939d6021aa9d7d857b179205af7f49efa40fb750bac5c4d51615d4366c77762f
3. Call Blocker [8], sha256: baecc852b2f68a949166359e474152b70cb8cc4a070df3933a92d0672e187674
4. CallerId Block [9], sha256: 8bd6439dceae65033d466319098f1cb8d29b13e254ba6f51d45cd1cb264109d9
5. Hiya [10], sha256: c5f0b9050e47abbdd7b8aff1c4ae6a30c0bc5a8a67a5bcfd4eb10ede3bc2a5a0
6. ShouldIanswer [11], sha256: 47742d1bc20854818d8589f0f5e9f0606bccebed79f038d254e0329da13f9979
7. ShowCaller [13], sha256: d36a7286768ad34751a64560707a059b5375699f620465aad44009e443ba3290
8. Stop met mij te bellen [12], sha256: 2ec5fa0cc3f6679347ed00da188a08a458d10a533583c11a891bba2c9421813c
9. TelGuarder [14], sha256: 42a6e65fc96c204dcd4f6f8aa60ab92ce7df2588a8f0a28a1cf940ba0d3c6785
10. TrueCaller [15], sha256: 1617441470c65eb40a6236b8d7ff0933846b21ce9c64661577a613c06ac8ef2a

# B Tools used

Several tools are used for static and dynamic analysis. As there is little overlap between the 2 methods, the tools are split as such.

## B.1 Static analysis

- apktool[21] V2.6.0
- dex2jar[22] V2.1
- jadx[23] V1.4.1

## B.2 Dynamic analysis

- Android Studio[27] V2021.2.1-Patch1

```
Android Studio Chipmunk | 2021.2.1 Patch 1
Build #AI-212.5712.43.2112.8609683, built on May 18, 2022
Runtime version: 11.0.12+0-b1504.28-7817840 amd64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
Linux 5.18.1-1-manjaro
GC: G1 Young Generation, G1 Old Generation
Memory: 2048M
Cores: 16
Registry: external.system.auto.import.disabled=true

Current Desktop: X-Cinnamon
```

- android-tools[27] V31.0.3
- android-sdk-platform-tools[27] V33.0.2-1
- sqlitebrowser[30] V3.12.2
- mitmproxy[31] V8.1.0
- Frida[24] V15.1.24
- frida-server[25] V15.1.24
- frida-script.js[26] commit 91422adeda7982e32d761e7efe6ac95286eb8254

## B.3 Operating system

```
Linux 5.18.1-1-MANJARO #1 SMP PREEMPT_DYNAMIC x86_64 GNU/Linux
```

## B.4 Other tools

Listed here are some of the frameworks the tools above rely on.

- OpenJDK[35] V18.0.1.1
- Python[36] V3.10.4