Dynamic analysis of Android applications to extract spam caller IDs

Christiaan van Luik
Supervisor(s): Apostolis Zarras, Yury Zhauniarovich
EEMCS, Delft University of Technology, The Netherlands

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

Spam calls are becoming an increasing problem, with people receiving multiple spam calls per month on average. Multiple Android applications exist that are able to detect spam calls and display a warning or block such calls. Little is known however on how these applications work and what numbers they block.

In this research, the following question is investigated: Can we do a brute force dynamic analysis on Android spam call blocking apps, to extract caller ID information from apps that cannot be or is not extracted through static analysis? A tool is created that is capable of doing such a dynamic analysis, by installing such an app on an emulator, sending emulated phone calls to the emulator, and using screenshot comparison techniques to determine whether the call is classified as allowed or blocked by the respective app. The tool developed in this research can, fully automated, test caller IDs on 8 different Android apps. Apart from a number of initial setup steps to install and configure the apps in the emulator, the tool takes about 1.5 seconds on average to analyze 1 caller IDs on one app.

1 Introduction

1.1 Background

Many people receive multiple spam or scam calls per month. The research by Hiya [8] shows that people in the USA receive 16 spam calls per month on average. Multiple smartphone applications exist to detect spam calls and block them. Currently, there is no insight or research on how these applications work technically. The existing research of Sherman et al [9] focusses on the user experience perspective. However, research into extracting blocked caller IDs has not been done yet.

Finding out how these applications block spam calls, is useful in order to be able to compare effectiveness of multiple methods and to determine what would be the ideal technique or combination of techniques to combat spam calls.

1.2 Research Question

This research will focus on the possibility of extracting caller IDs from Android Applications in real-time. The main research question will be: *Can we do a brute force dynamic analysis on Android spam call blocking apps, to extract caller ID information from apps that cannot be or is not extracted through static analysis?*

Additionally, the following sub-questions will be addressed: *How long does this analysis take and how can it be speed up?* and *Can we include other information, apart from caller IDs, that is used to determine if a call is blocked or not in the analysis?*

2 Methodology

This section focuses on the methodology of this research. First, the used software will be listed, followed by an explanation how these tools were used to do the research.

2.1 Tools

Android Debug Bridge (ADB)
Android Debug Bridge [1] is a command line developer tool that enables communication with an android device or emulator. It allows you to execute certain device actions and also provides a shell for running commands on the device. In the context of this research, its most used feature is ADB's ability to send a simulated phone call to a device.

Android UI Automator
Android UI Automator [6] is a UI testing framework for Android-wide UI testing and automation. It allows interaction with various elements on screen as well as executing specific device actions. It provides various ways to find elements. In this project, UI Automator is mainly used for automating the different steps to set up individual apps, steps like accepting the terms and conditions and granting the required permissions.

Appium
Appium [3] is an open source test automation framework. It serves as an interface to different vendor-specific test automation frameworks. It wraps these frameworks into one API using the webdriver protocol. You can then use a client in any programming language to send the appropriate HTTP requests to the appium server. The Appium server runs locally, and interacts with the locally running emulators. In this project, the Appium Client is used to be able to easily write automation code which uses both ADB and UI Automator under the hood.

Appium Inspector
Appium Inspector [2] is a graphical tool, which loads the Android layout XML file, and displays it next to the screenshot. It can be used to find out how a layout is built up and to find the right IDs or XPATH for selecting an element. In this project, Appium Inspector is used to determine the right resource-ids that actions, like clicks and text input, are applied to in automating the different app setup processes.

OpenCV
OpenCV [4] is an open source machine learning and computer vision library that includes a wide variety of tools and algorithms related to computer vision. In this project, OpenCV is used for checking if a subimage is part of a larger image, in our case a reference subimage with a visual indication of the blocked or allowed status, which is compared to the full screenshot taken.

2.2 App selection

The first step in answering the research question is deciding on which apps to test. We decided on testing 10 apps, which fits the size of this research project and allows for enough variety in the apps under test.

For selecting these apps, the following criteria must be met:

- Android App, preferably in the Google Play Store
- App has a significant number of downloads, in order for the research to be relevant.

- App is free
- App uses a database or algorithm to decide whether a call is a spam call. App should not only use a user-defined blocklist.

Based on these criteria the following 10 apps were selected:

- Call Control - SMS/Call Blocker. Block Spam Calls! (com.flexaspect.android.everycallcontrol)
- CallApp: Caller ID & Recording (com.callapp.contacts)
- Call Blocker - Stop spam calls (com.unknownphone.callblocker)
- Caller ID, Phone Dialer, Block (com.callerid.block)
- Should I Answer? (org.mistergroup.shouldianswer)
- Showcaller: Caller ID & Block (com.allinone.callerid)
- Stop Calling Me - Call Blocker (com.mglab.scm)
- Spam Call Blocker - telGuarder (com.telguarder)
- Truecaller: Caller ID & Block (com.truecaller)
- Hiya - Call Blocker, Fraud Detection & Caller ID (com.webascender.callerid)

These apps were then downloaded in APK format from apkcombo.com to use in the analysis.

2.3 Short overview of automated analysis

To do this research, a piece of software has been written that automates testing whether certain caller IDs are allowed or blocked. This is done for 10 applications. In short, the software works as follows:

As input, the tool requires a list of caller IDs (phone numbers) to be tested. The tool already comes bundled with the apks of the 10 Android apps as well as the required reference images used for image comparison, 2 or 3 per app, depending on which statuses the app differentiates.

Then, in short, the following automated actions are executed:
The tool first loops over the 10 apps, **for each app**:

- The app is installed and setup on an emulator (see section 2.4)
- **For each number** in the input list of numbers:
  - The device screen is locked (turned off)
  - A call is simulated
  - The tool continuously takes screenshots until an occurence of either an allow, caution (if applicable) or block reference image is present, in which case we can decide on the status of the call.
  - The simulated call is cancelled.
  - The number combined with the found status (either allow or block) is saved.
- The app is uninstalled

In the sections below, the above short overview is explained in more detail.

2.4 Automating app installation and setup

Although installing an APK file on an Android device or emulator is easy to do with a single ADB command, this turned out not to be enough. After installing the apk and granting all permissions that the app requests via ADB, more steps need to be taken before the installed app can be used for the actual analysis. All apps require the user to go through some other initial steps before you can use its functions. These steps differ per app, but generally include actions like accepting the terms and conditions, granting permission to have the app run in the background, and registering or logging in with an account. These steps have all been automated for each individual app using Android UI Automator.

2.5 Testing provided phone numbers

When running the tool, the 10 apps are tested one by one. Each time an app is installed and setup as described above, after which the real analysis starts. For each number in the provided list of numbers to test, the following automated actions are executed.

First, a simulated phone call from the number to be tested is sent to the emulator. Then, in a continuous loop, a screenshot of the emulator screen is taken. This screenshot is compared with the allow, caution (if applicable) and block reference images, so see if the reference image occurs in the screenshot. If that is the case, we can decide on the status (either allow or block) of the number under test. We also measure and save the time it took for the app's status popup to appear.

This continuous loop and measurement of the time it takes until the status is shown is done because not all apps display information equally fast. It takes some time for the allow, caution or block popup to appear on the screen, which is why the script loops until this app popup is present on the screen.

2.6 Reference image comparison

In order to detect if a call gets blocked or allowed by a certain app, we make use of image comparison. Using the OpenCV library, we check if the allow, caution or block reference image is part of the larger screenshot.

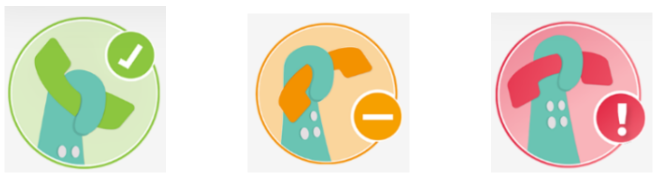

Figure 1: Reference images of ShouldIAnswer app, for respectively the allow, caution and block statuses.

For each app, there are two reference images, one for allow and one for block. In some cases, the spam call block apps also distinguish a third category, a caution status. In that case, a third reference image for this status is added. These reference images are partial screenshots of the popup that becomes visible with the status of the number. For example, the reference image for allow might be a small, cropped green phone icon that is distinctive for the non-spam status, while the block reference image might be a small image of a red icon that is only visible when the tested app regards a number as spam. As an example, the reference images for the ShouldIAnswer app can be seen in figure 1.

For comparing the screenshot and the reference image, OpenCV's Template Matching [5] functionality is used. It allows to check if a smaller image occurs in a larger image. This works if the image to be found is a subset of the screenshot.

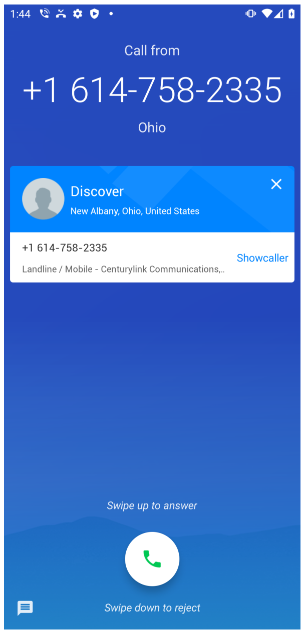

Figure 2: Example screenshot of allowed call

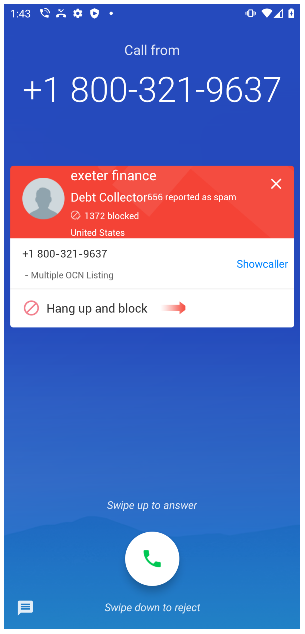

Figure 3: Example screenshot of spam call

Figure 4: Reference image for allow status

Figure 5: Reference image for block status

3 Experimental Setup

In this section, the methodology covered in section 2 will be explained in more detail, and some of the particularities encountered during this research will be explained. In general, this information serves as an addition to the content in section 2. The source code created in this research is available in this GitHub repository: https://github.com/cvl01/spam-call-analysis.

3.1 General setup

The tool built in this research project is built in Python, using the Appium Python Client to send the required commands to Appium, which then will execute these actions on the emulator.

When simulating a call, it is important to first lock the screen, as the app popups with information might differ based on whether the device is locked or unlocked. To make sure the reference images match the screenshot and to guarantee consistency, the testing always happens in the locked mode. The incoming call screen then appears fullscreen and the spam call blocking app's popup displays on top of the native Android incoming call screen.

The tool outputs the results to a CSV file, where each line contains the package name of the tested app, caller ID, status, and time it took to detect.

3.2 Creation of reference images

Reference images are created manually, by taking screenshots of respectively allowed incoming call screens and blocked incoming call screens, like can be seen in figures 2 and 3. These screenshots are then cropped to only show a small, distinctive part of the popup that can be used to determine whether a call is labelled as spam or not.

3.3 Automating app installation and setup

Each individual app requires you to do different steps and actions before the app's functionality can be used, like logging in with a Google account, or setting the app as default caller ID app. Since these steps and also the application layout differs per app, for each app a different automation script was created. This automation script launches the app, and then executes a series of tap, swipe, keyboard or other actions that are required to setup the app.

To find out which buttons to tap, and how to find these programmatically, the record feature of Appium Inspector was used. This record feature allows you to click around in an app, and record what you are doing. These taps on the screen are linked to a certain resource-id or XPATH for programmatically locating the elements on the screen. Additionally, some tweaking needs to be done, since the record feature not always is able to generate stable code.

When needed, the option to log in with a Google account is used, since that is most easy to automate as opposed to creating an application-specific account.

After running this app-specific automation, the app's functionality is active and testing can begin.

4 Results

In this research, it was possible to develop a software tool that can dynamically extract the caller ID information from Android spam blocking apps. It is able to determine whether a caller ID is labelled as spam or not by such an app, as well as measure the time it takes before this information pops up on the screen. The results that are mentioned in this section and shown in tables 1 and 2 are obtained by running the tool on a 100-number dataset of both US and Dutch caller IDs, which can be found in appendix B.

4.1 Incompatible apps

Three of the ten selected apps turned out to be incompatible with the chosen method and experimental setup. Truecaller (com.truecaller) uses some automated number verification in the app, before you can start using it. This fails on an emulator, likely because there is no actual number / SIM card

Package Name	Avg. of delta	Min. of delta	Max. of delta	Min. of acc.	Avg. of acc.
com.allinone.callerid	1,26	1,17	1,46	100%	100%
com.callapp.contacts	3,14	1,79	7,69	96%	100%
com.callerid.block	1,33	1,22	2,01	100%	100%
com.flexaspect.android.everycallcontrol	1,23	0,99	7,53	100%	100%
com.mglab.scm	0,03	0,00	0,20	100%	100%
com.telguarder	1,44	1,14	3,22	100%	100%
com.webascender.callerid	1,13	1,05	1,47	98%	100%
org.mistergroup.shouldianswer	1,86	1,59	2,17	100%	100%
Total	**1,42**	**0,00**	**7,69**	**96%**	**100%**

Table 1: Statistics on running time (delta) per number in seconds and accuracy of the image comparison per number. Timeouts are excluded.

Package Name	allowed	blocked	caution	timeout
com.allinone.callerid	68%	32%	0%	0%
com.callapp.contacts	85%	11%	0%	4%
com.callerid.block	68%	32%	0%	0%
com.mglab.scm	74%	26%	0%	0%
com.telguarder	39%	24%	37%	0%
com.webascender.callerid	68%	32%	0%	0%
org.mistergroup.shouldianswer	12%	82%	6%	0%
Total	**59%**	**34%**	**6%**	**1%**

Table 2: Overview of percentage of allowed and blocked numbers for the 100-number test dataset.

	Dutch Numbers			US Numbers		
Package Name	allowed	blocked	caution	allowed	blocked	caution
com.allinone.callerid	54%	1%	0%	14%	31%	0%
com.callapp.contacts	50%	4%	0%	35%	7%	0%
com.callerid.block	54%	1%	0%	14%	31%	0%
com.mglab.scm	55%	0%	0%	19%	26%	0%
com.telguarder	29%	16%	10%	10%	8%	27%
com.webascender.callerid	31%	24%	0%	37%	8%	0%
org.mistergroup.shouldianswer	2%	53%	0%	10%	29%	6%
Grand Total	**39%**	**14%**	**1%**	**20%**	**20%**	**5%**

Table 3: Comparison of app performance between Dutch and US caller IDs

connected to the emulator. Because of that, it was not possible to automate setting up Truecaller.

Another app, Call Blocker - Stop spam calls (com.unknownphone.callblocker) does not show any visual indication on the incoming call screen. This app is more intended as a community-driven number review app. After being called by a number, in the app you can see what messages other users wrote about this number. Because the app does not have automatic detection features and is not showing a visual indication on the incoming call screen, it was not used in this research.

A third app, Call Control (com.flexaspect.android.everycallcontrol) looks like it will detect spam call numbers at first, but it turns out that in order to enable this functionality, a paid premium subscription needs to be activated.

4.2 App Comparison

Time comparison

A comparison was made between the 10 apps on time it takes to show the allow or spam status. Results of that can be found in table 1. As you can see, some applications definitely are faster than others. In this table, timeouts are excluded from the data and not part of the calculated values. Timeouts can arise when the screen can't be matched to a reference image, like described in section 6.2.

Stop Calling Me (com.mglab.scm)

An app that stands out in table 1 is Stop Calling Me (com.mglab.scm) which has very low running time. This is because this app works a bit different from the other apps. Instead of showing an overlay on the incoming call screen like other apps, it just terminates the call if it is classified as spam. So, if the call is blocked, the call is terminated, if the call is allowed nothing happens. The tool detects if the call is terminated. As you can see in the table, this takes maximum 0.20 seconds. For the allow status there is no real timing, if after a timeout of 2 seconds the call is not terminated we assume it is allowed, but the time in the data is fixed at 0.0 for the allowed calls. This app is the fastest when it comes to detecting whether a call should be blocked.

Call Blocking Statistics

Table 2 gives an overview of the apps and their performance on the 100-number dataset. From these results, one should not draw too many conclusions, since the chosen set of numbers can heavily influence how the apps work and what allow and block percentages they have. However, what can be noted from these results is that 2 applications, com.allinone.callerid, com.callerid.block, both show the same behaviour, not only are the percentages the same in the table, but they block exactly the same numbers. Although com.webascender.callerid has the same percentages in the table, that is a coincidence, as this app blocks and allows different numbers. Seeing the exact match between the behaviour of these two apps, there is a big change that these applications make use of the same dataset under the hood. This can however not be verified using this tool, but static analysis might be able to give a definitive proof of this.

Dutch and US Numbers

Finally, in table 3 a comparison between the app behaviour on Dutch and US numbers in the dataset is made. We can see that on the same dataset, for US numbers all tested apps block at least some numbers, while for Dutch numbers, some apps block significantly fewer numbers than others. This might be because some apps are optimized for the US market, and others have more data on Dutch or maybe European numbers.

4.3 Tool Performance

A slight performance gain is can be achieved by running the tool headless. This gives around a 10 to 15% speed increase. The tool can be further sped up by running the analysis on multiple threads. Using threads decreases the total runtime, but not the time per number that is used in table 1.

4.4 Overhead

Before doing the actual analysis, the emulator needs to be setup. These setup steps, that include installing the app, logging in into a Google account (if needed), and configuring the analysed app, are done each time the analysis runs on a clean emulator. This setup can take up to 1 minute for a single app. This makes, that when testing a relatively small amount of caller IDs, the emulator and app setup takes more time than the actual analysis.

4.5 Accuracy

The core functionality of the developed tool depends on the image comparison that is done in order to determine if a call is allowed or blocked.

The OpenCV Template Matching functionality that is used gives an accuracy score, using OpenCV's TM_CCOEFF_NORMED matching method, on the accuracy of the found match. In the tool, a threshold of 0.95 is set. In the results, we see that all tested numbers have an accuracy in the range 0.96 till 1.00, indicating perfect matches.

4.6 Comparison with other tools

Since telephony abuse and spam calling is growing, multiple researchers have performed research related to spam calling. Related to Android spam call blocking apps, and how these work and combat spam calls, little research has been done. Pandit et. al. [10] investigated multiple data sources that may be used to construct phone blocklists. Additionally, they measured the ability and effectiveness of these blocklists in blocking future unwanted spam calls. Their results show that these blocklists are capable of blocking a significant percentage of spam calls (more than 55%) and that the false positive rate is very low. Unfortunately, it is not possible to say whether the datasources and blocklists analysed by Pandit et. al. are the same datasources that are used under the hood of the analyzed apps in our research. So, it is not possible to say whether the findings of Pandit can be applied to our work aswell.

Other researchers have also set up different frameworks and methods for detecting spam phone calls. But these are new frameworks, and not an investigation in the data or heuristics of existing spam call blocking apps.

To the best of our knowledge, this research is the first research that analyses Android spam call blocking applications by use of dynamic analysis on a device or emulator.

5 Responsible Research

5.1 Mis-use

The tool that is created as part of this research, allows the user to test a phone number on different Android spam call blocking applications. This poses a risk of immoral usage. In fact, a spam caller that possesses a certain amount of phone numbers, might use the tool to check those caller IDs, and see which of them are blocked by which app. A spammer could then choose a caller ID that is not yet part of the blocklists of these applications.

It is relatively easy to run an emulator, install a spam call blocking application on it, send an emulated phone call to the emulator, and observe whether the caller ID is blocked. That is why, in my opionion, the above described risk is still present even without the tool of this research. The only difference this tool makes, is that every manual step goes a bit faster when automated.

5.2 Reproducibility

When doing research, reproducibility is very important. This can be achieved by publishing the source code of the automation scripts that were created in this research. It should be noted however, that the apps that are analyzed in this research will change after updates, or even be removed from the Google Play Store. That is why one cannot guarantee that various parts of the analysis, like the comparison of screenshots, will work on future versions of the apps.

Additionally, some, if not all the apps, use online, external databases to determine whether a caller ID will be blocked or not. These databases are constantly updated, with caller IDs being added and being removed. That is why the behaviour of these apps, even when using the exact same app versions, might not always be the same. This will lead to different results when running the analysis again on a later moment in time.

However, detailed information on our method, combined with the published source code, will ensure that even when apps might visually change, other researchers should be able to create new reference images and replicate the research using the same method that was used in this research.

6 Discussion

The results show that it is possible to do a dynamic analysis of Android spam call blocking apps. In this section, we discuss the effectiveness and the limitations of this work.

6.1 Effectiveness

If one tries to get insight in how certain Android spam call blocking applications work and which caller IDs they block, different ways of static and dynamic analysis can be applied to extract information. When comparing the chosen approach of this work to a static analysis in which caller IDs blocklists are extracted from the application source code, this solution is very costly. One would get thousands or even millions of numbers at once when extracting a database from an application. Testing 1 million numbers using this tool would cost 833 hours for one application.

However, not every application uses a database, databases can be hashed or encrypted, and even when a database exists, it can be very difficult to extract. These and other problems that arise when doing static analysis, are not present when doing dynamic analysis with screenshot-based comparison. Our tool is suitable for every app that displays some visual indicator of the allow or block status, that can be used for image comparison. This makes it a tool that can be applied to many applications, irrespective of how they function under the hood. This tool is very effective as a last option to extract data from an app, when other methods, like different forms of static analysis, fail. However, since testing one number takes 1.5 seconds on average, testing large datasets on multiple apps requires a lot of time.

6.2 Limitations

Versions & UI changes

Because the tool uses both UI Automation for the setup and screenshot comparison for the analysis, the tool is highly vulnerable for changes in the layout of either Android itself or the tested app. Every change in the UI can lead to parts of the tool failing. That is why reproducibility can only be guaranteed when using the same emulator configuration (most important: same android version, same screen size) and using the same app versions.

Caller ID Specific Layouts

Another limitation of the tool is the inability to adapt to some app's changing layouts. Some apps change the layout based on the phone number, by adding a specific profile image or text. This is not a problem as long as part of the screen is suitable for our partial image comparison technique, but when the whole screen changes this might lead to the tool being unusable for that specific screen. An example from the CallApp Contacts App can be seen in figure 6. As you can see, a specific background is added compared to the regular incoming call screen shown in figure 7. The reference image show in figure 8 is not present in figure 6, leading to a timeout when testing this specific number. It is not possible to find a reference image that works for both cases (fig. 6 and 7) while not conflicting with the app's blocked call screen layout.

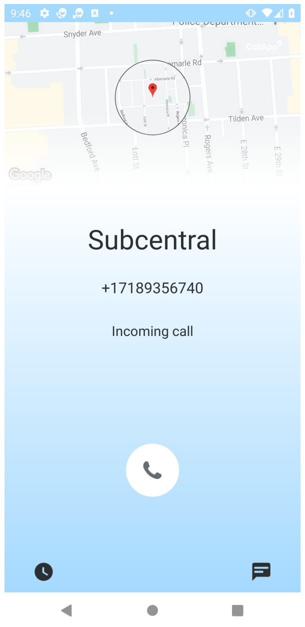

Figure 6: CallApp Contacts with specific background

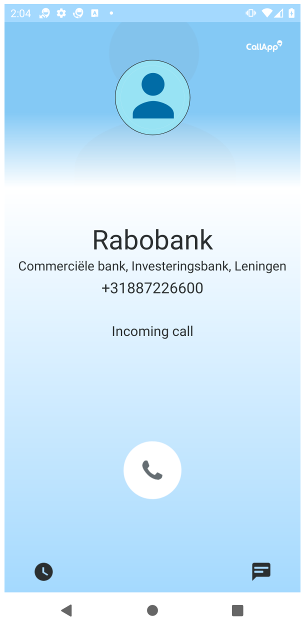

Figure 7: CallApp Contacts regular incoming call screen

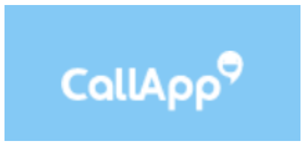

Figure 8: CallApp Contacts reference image for the 'allow' status

Threading and performance

It is possible to run the application on multiple threads and thus on multiple emulators. However, the Android emulator takes quite some resources. On the MacBook Pro (16-inch, 2021) with the Apple M1 Pro processor and 16GB of memory that was used for testing, stable and reliable results could only be guaranteed with maximum 2 emulators running at the same time. Running more than 2 emulators works, but then the emulators may start to lag and will sometimes freeze, which makes the process unreliable.

This issue could be resolved by running the tool with external emulators, on another device, for example on virtual machines (VM) in the cloud. This would enable to distribute more and run the analysis faster. Ofcourse, then the cost will also increase, because more computing devices need to be set up. The article by Bierma et. al. [7] on a Large-scale Android Dynamic Analysis Tool gives some ideas on how to set up a dynamic analysis on multiple VM nodes.

7 Conclusions and Future Work

In this research, a tool was developed that is capable of doing a dynamic analysis of Android Spam Call Blocking applications, and extracting a list of statuses (allow / caution / block) for each tested caller ID. This tool can fully automatically test caller IDs on 8 different Android apps. Apart from a number of initial setup steps to install and configure the apps in the emulator, the tool takes about 1.5 seconds on average to analyse 1 caller ID on one app.

Additionally, the tool was made faster by choosing an efficient image processing library (OpenCV), running the emulator headless and adding the ability to run the tool using multiple threads.

In this research, the initial second sub-question, on other information apart from caller IDs, was not addressed. This could be further researched in the future.

Future work

In this research, an open issue is addressing the initial second sub-question, if it is possible to include other information, apart from caller IDs, that is used to determine whether a call should be blocked or not? One can think of changing the timezone of the device, changing the location, or switching the emulator between online and offline mode, to see if there is any difference on how the apps under test behave when conditions vary. To avoid groping in the dark and spending a lot of testing time trying to discover which variables are part of the app's heuristics, one could do a static analysis of the application code first. During that static analysis, one might find some indications of which variables are part of the heuristics. The hypotheses that arise from doing this static analysis can then be further tested and proven by doing a dynamic analysis using a extended version of this tool, that will test the apps using different setups. One will then likely be able to find some examples that prove the hypotheses and show that the app behaviour can vary based on the setup and conditions the device or emulator is in.

When doing further research, it should be interesting to compare different techniques for dynamic detection. In this research, image / screenshot comparison was used, but it might be possible to use alternative methods to achieve the same functionality. One alternative might be to use Android UI Automator to detect text on the screen, instead of detecting partial images. Once this is developed, a comparison can be made on what is the fastest and most accurate method.

References

[1] "Android Debug Bridge (adb)." [Online]. Available: https://developer.android.com/studio/command-line/adb

[2] "Appium inspector," original-date: 2021-04-06T17:37:49Z. [Online]. Available: https://github.com/appium/appium-inspector

[3] "Appium: Mobile App Automation Made Awesome." [Online]. Available: https://appium.io/

[4] OpenCV. [Online]. Available: https://opencv.org/

[5] "OpenCV: Template Matching." [Online]. Available: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html

[6] "Write automated tests with UI Automator." [Online]. Available: https://developer.android.com/training/testing/other-components/ui-automator

[7] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, "Andlantis: Large-scale android dynamic analysis," *arXiv preprint arXiv:1410.7751*, 2014.

[8] Hiya, "State of the phone call," p. 9, 2019. [Online]. Available: https://assets.hiya.com/public/pdf/HiyaStateOfTheCall2019H1.pdf

[9] K. M. I.N. Sherman, J. Bowers, "Are you going to answer that? measuring user responses to anti-robocall application indicators," in *NDSS*, 2020.

[10] S. Pandit, R. Perdisci, M. Ahamad, and P. Gupta, "Towards measuring the effectiveness of telephony blacklists." in *NDSS*, 2018.

A List of Apps with version

A list of used apps with the used version can be found in table 4.

App name	Package name	Version Name	Version Code
Showcaller	com.allinone.callerid	2.2.8	272
CallApp Contacts	com.callapp.contacts	1.962	1962
Caller ID	com.callerid.block	1.7.7	184
Call Control	com.flexaspect.android.everycallcontrol	2.13.5	40155
Stop Calling Me	com.mglab.scm	2.3.21	382
telGuarder	com.telguarder	1.1.1	791
Truecaller	com.truecaller	12.30.6	1230006
Call Blocker	com.unknownphone.callblocker	2.6.4	147
Hiya	com.webascender.callerid	12.1.0-9677	120100
Should I Answer?	org.mistergroup.shouldianswer	1.0.206	206

Table 4: List of used apps and used versions of those apps

B Test dataset of 100 Caller ids

The numbers below were used for generating the results shown in this paper:
+18009423767, +16058844130, +18666257291,
+18003535920, +18888996650, +13132631171,
+18558440114, +18882224227, +18442069035,
+18665320423, +18558953393, +18003219637,
+18662507212, +18889346489, +18776478552,
+17204563720, +18442068573, +18554197365,
+18669145806, +18009460332, +18662423315,
+17135686986, +18553066998, +18009475096,
+18005812620, +16147582335, +15862500071,
+17189356740, +18776983261, +15052530591,
+12065390456, +18002927508, +18779473639,
+18557077328, +12012010089, +18009149244,
+12064532329, +18008068840, +18669513700,
+18102725120, +13033238692, +18335258081,
+18009378997, +18775875726, +18778235399,
+31103180406, +31103180407, +31103188856,
+31108920400, +31134634900, +31182200013,
+31202091180, +31202119920, +31202134531,
+31202265194, +31202279303, +31207060014,
+31302061014, +31307125776, +31316550003,
+31332096457, +31332096712, +31332096980,
+31332137540, +31352019505, +31402071474,
+31402291122, +31534876888, +31570768301,
+31580502130, +31612616455, +31617407181,
+31620348959, +31623197835, +31625277365,
+31627840569, +31628282952, +31628784203,
+31628877018, +31644196164, +31644677344,
+31646455395, +31682262701, +31683558391,
+31683827962, +31684776454, +31702071023,
+31702071241, +31702171046, +31742045520,
+31850501499, +31850870183, +31850870713,
+31852087866, +31857325695, +31882111460,
+31882716420, +31885016700, +31887226600,
+31887751600