Table of Contents

Index of Figures

Index of Tables

ABSTRACT

The most famous mobile operating system to date is Android. It is used by millions of people worldwide and subsequently is the most targeted platform of all, by malicious users. A multitude of new viruses and pieces of malware or variants of existing ones is discovered every day by security companies, that vary – in relation to how much dangerous they are – from just popping up advertisements to stealing credentials and wiping a device completely.

This thesis deals with a category of malware that charges users without their knowledge by sending SMS messages or initiating calls to premium rate numbers, known as premium-rate dialers and SMS fraudsters.

In order to mitigate the risks stemming from this type of malicious programs, a custom mechanism is proposed that compels the user to perform some action before every attempt to send an SMS or place a call so that he is protected against any covert try to do so.

The contribution of this master's thesis is a protective system that is composed of a random CAPTCHA puzzle that a user must solve correctly before the aforementioned attempts finish. Additionally, users can shift through the destination numbers of their actions and store them in databases so the next time they encounter any of these stored numbers, they are made aware that they are potentially dangerous or not.

1. INTRODUCTION

Android Operating System made its first appearance on September, 2008. It was one of the first OSs for smart devices and since then, it managed to evolve to the leading OS in the market, largely because of its open source nature. IDC reports that Android leads the smartphone market with almost 85% of the market share in the second quarter of 2014 [1].

As history has shown, the more popular something is, the more people will try to take advantage of it, for their own personal gain. Android was not an exception to this generic rule. Since its debut, the presence of malware affecting it that was detected in the wild, has grown exponentially. Cisco mentions in its 2014 Annual Security Report that of all mobile malware in 2013, 99% targeted Android devices [2]. Mirroring this, F-Secure states in its Mobile Threat Report for the first quarter of 2014 [3], that 99% of its findings for that period of time were designed to run on the Android platform. More specifically, they discovered 275 new threat families (or new variants of known families) as opposed to just 1 new threat family each on iOS and Symbian platforms. Also, security experts from Kaspersky Lab estimate that about 98% of all mobile malware targets users of Android devices as reported in their joint Mobile Cyber Threats report with INTERPOL in October, 2014 [4].

This continuous increase in Android malware is based on the fact that it's easy to develop an Android application (one can just download the Android SDK and start working), among other reasons. There is also no need for a developer to pass any kind of validation if he doesn't intend to upload his work to Google Play Store. A download link pointing to the application is all that is needed for distribution.

According to [5] and [6], these malicious applications have personal information gathering, remote control of the device or financial gain among others as an ulterior motive for their malicious behavior.

One class of malware, that can cause financial loss to Android users, is consisted of SMS fraudsters and premium dialers. FakePlayer, HippoSMS, Jifake and MouaBad are just a few examples of this category of malware.

- FakePlayer: Discovered in August 2010, AndroidOS.FakePlayer (also known as AndroidOS/Fakeplayer.A [F-Secure], ANDROIDOS_DROIDSMS.A [TrendMicro]) is a Trojan that masquerades as a video player and attempts to send premium-rate SMS messages containing the text 798657 to the numbers 3353 and 3354 [7].
- HippoSMS: Android/HippoSMS.A was discovered in July 2011 and is a cracked version of

a legitimate application, found in third-party alternative chinese app markets. This malware is responsible for sending an SMS message to a premium number and deleting incoming SMS messages from certain numbers [8], [9].

- Jifake: Android.Jifake was first uncovered in July 2012 and is a Trojan horse that, when executed, installs Jimm – a popular messaging app in Russian-speaking countries. When the app is launched it tries to send SMS messages that contain the body 48876374538 to the premium rate number 5537 [10].
- MouaBad: MouaBad.P is a Trojan, variant of the malware family labeled Mouabad by Lookout security team, which gives to remote attackers the ability to make phone calls or send SMS messages to premium rate numbers, without any user intervention. This variant, also, tries to avoid detection employing a clever technique. It waits for a period of time after the screen is locked and then it places the call. If it detects any user interaction (e.g. unlock), it ends the call [11].

While malware presence is still growing especially in third party app stores, the majority of simple users can't recognize the danger they might be in. So they have, at best, a basic understanding of what they could do to mitigate the risk of being infected and have their personal data stolen or their bills overcharged. Thus, it becomes imperative that the companies involved with mobile platforms in general, develop mechanisms to protect users.

Currently, users can be protected against these attacks only by themselves, if they take a number of steps.

- ✔ All apps distributed by Google Play are verified, so it's a good idea to download apps only from there. Installing apps from third-party app stores is best to be avoided.
- ✔ There are several applications that scan a device for unclosed vulnerabilities, which users can use every once in a while.
- ✔ Using a security solution (anti-malware, anti-virus etc.) that can scan files on-the-fly is always a plus for anyone concerned with the security of his device (smartphone, tablet, computer etc.)
- ✔ Applications and firmware should be updated regularly in order for new patches and fixes be applied and security holes closed.
- ✔ Extra caution must be exercised when installing an app, in terms of permissions. Users must pay attention to what an app requires to run, so they can ensure that realistic demands are being made on a device's features.

Even if someone has implemented all possible security measures, he might still be at risk. In 2011,

malware called DroidDream managed to infiltrate the official Google Play Store [12]. In addition, many malware samples can avoid detection from security applications which base their efforts on signature recognition. Furthermore, a malicious app can circumvent the listed permissions.

The purpose of this thesis is to develop a system that can protect mobile users against this type of malware at real time, a built-in mechanism for the Android platform, that intercepts any attempt made to send an SMS or make a phone call and prompt the user to solve a CAPTCHA puzzle before proceeding.

2. BACKGROUND

After careful research, three files were recognized as responsible for the outgoing call sequence:

- /packages/services/Telephony/src/com/android/phone/CallController.java
- /packages/services/Telephony/src/com/android/phone/SipCallOptionHandler.java
- /packages/services/Telephony/src/com/android/phone/OutgoingCallBroadcaster.java

Figure 1 shows the typical sequence of an outgoing call in normal conditions.

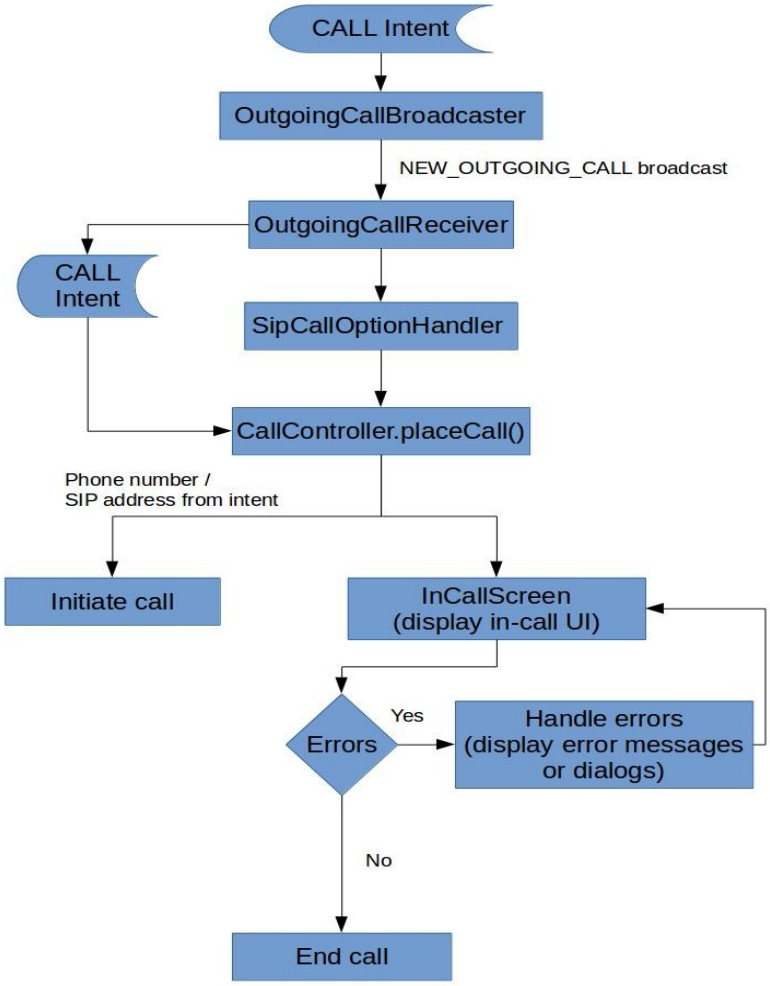

Figure 1: Normal outgoing call sequence

Text-based SMS messages can be sent by one of two ways: either launch the built-in "Messaging" app (standalone or from a third party app), or use SmsManager class and specifically sendTextMessage() or sendMultipartTextMessage() methods.

The following figures 2 and 3 show how a message is sent under normal circumstances.

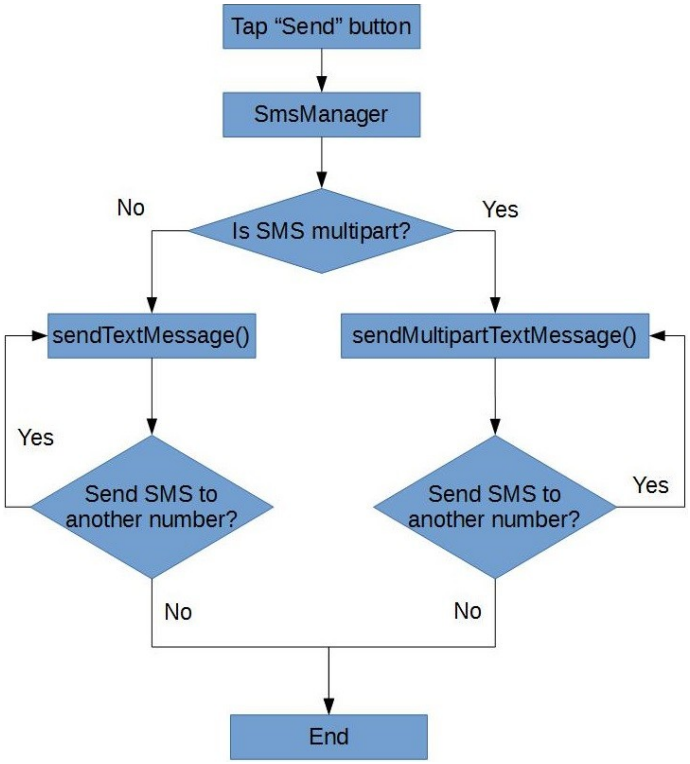

Figure 2: Normal SMS sending from third party apps

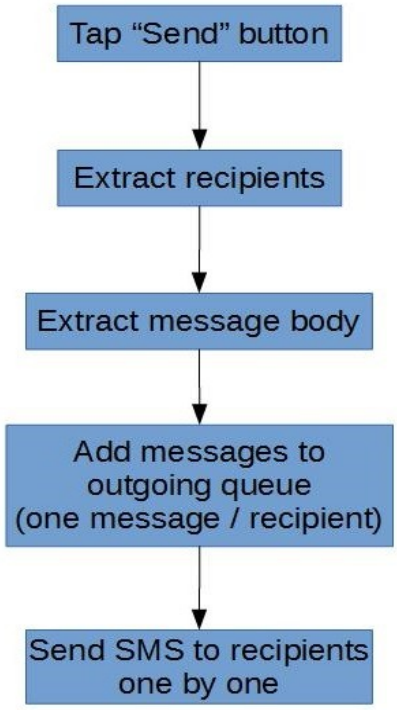

Figure 3: Normal SMS sending from built-in “Messaging” app

In many countries, there are legitimate premium services – via either SMS or phone calls – from where users can obtain specific content (e.g. a theme etc.) or be provided certain services and for which, charges higher than normal occur. When someone wants a particular type of content, they can simply send a text to a special number called short code, get the content they want and be charged for this purchase on their phone bill. Very common premium-rate calls are adult chat lines and technical support. Calling numbers like these induces extra charges beyond the normal cost of a phone call.

Cyber criminals have used premium-rate numbers since the beginning, to defraud unsuspecting users. For them, these types of services are a very easy target to make money off of. The reason theses scams are so common is because of how simple they are. One can just go to various premium services providers around the world, register for a short code or a premium-rate number and if the malware is deployed, start making money easily and quickly.

Typically, malicious users make an app that looks like a very popular, legitimate one or crack an official app and repackage it with malicious code to attract as many users as possible to download it. The trojanised app is very often packaged alongside the legitimate one and requests a lot more permissions than it, when a user tries to install it. The malicious app, then, does all the work in the background. It sends an SMS message or places a call to a premium-rate number

without the user's knowledge or intervention and the phone bill is overcharged.

2.1 Sample gathering and analysis

Before developing the custom CAPTCHA mechanism it was imperative to gather as many malware samples as possible. As a sample pool, all relevant samples from the combination of the Android Malware Genome Project [13] along with Contagio Mobile Malware Mini Dump [14] were used.

To compartmentalize the procedure, it was necessary to identify which of all the available samples were relative to the custom mechanism (the ones that actually send an SMS or make a phone call in the background). To that end, Google's *droidbox* was used to dynamically analyze the samples and single out the pertinent ones. Droidbox [15] is a tool that displays information about the incoming/outgoing network data, file read/write operations, started services and loaded classes, information leakage through network, file and SMS, circumvented permissions, cryptography operations performed, broadcast receivers listed, sent SMS messages and placed calls of the Android application being analyzed.

Table 1 was the result of the analysis, displaying the sample family, number of samples in each family, month of discovery, its category and a brief description. All this information is supplied by Google's malware repository [16] and is compared against the results of the sample pool analysis.

Family	Number of samples	Month of discovery	Category	Description
ADRD	23 (22 in malgenome)	02/2011	Trojan	sending out device info
AnserverBot	187	09/2011	Trojan	install payload and retrieve commands from C&C
Asroot	8	09/2011	Root exploit	
BaseBridge	124 (122 in malgenome)	06/2011	Trojan	sending out IMSI and OS info, send and remove SMS
BeanBot	8	10/2011	Trojan	sending out IMEI, IMSI and phone number, send SMS to a premium number
Bgserv	10 (9 in malgenome)	03/2011	Trojan	sending out IMEI, device info
CoinPirate	1	08/2011	Trojan	sending out device model, SDK version, IMEI, IMSI
CruseWin	2	07/2011	Trojan	delete itself, delete SMS, send SMS to premium number
DogWars	1	08/2011	Trojan	send SMS to all contacts
DroidCoupon	1	09/2011	Malware	root exploit; remote C&C server
DroidDeluxe	1	09/2011	Trojan	sending out account name, authtoken, contacts, etc.
DroidDream	18 (16 in malgenome)	03/2011	Trojan	sending out IMEI, device info
DroidDreamLight	46	05/2011	Trojan	sending out IMEI, IMSI, model, etc.
DroidKungFu1	34	06/2011	Trojan	sending out IMEI, phone info, data on SD card
DroidKungFu2	31 (30 in malgenome)	07/2011	Trojan	sending out IMEI, phone info, data on SD card
DroidKungFu3	309	08/2011	Trojan	sending out IMEI, phone info, data on SD card
DroidKungFu4	96	10/2011	Trojan	sending out IMEI,

				phone info, data on SD card
DroidKungFuSapp	3	10/2011	Trojan	sending out IMEI, phone info, data on SD card
DroidKungFuUpdate	5 (1 in malgenome)	10/2011	Trojan	sending out IMEI, phone info, data on SD card
Endofday	1	05/2011	File infector	
FakeNetflix	1	10/2011	Trojan	steal log-in details
FakePlayer	6	08/2010	Trojan	send SMS to a premium number
GamblerSMS	1	07/2011	Spyware	sending out incoming/outgoing SMS, outgoing phone call
Geinimi	69	12/2010	Trojan	sending out device info, geolocation, connect to C&C
GGTracker	2 (1 in malgenome)	06/2011	Trojan	send SMS to a premium number, sending out phone number, SMS
GingerMaster	4	08/2011	Trojan	sending out device id, phone numbers, cpuinfo
GoldDream	47	07/2011	Trojan	sending out incoming/outgoing SMS, phone call, device id, subscriber id
Gone60	9	09/2011	Trojan	sending out contacts, SMS, call list, visited URLs
GPSSMSSpy	6	08/2010	Spyware	sending out location
HippoSMS	4	07/2011	Trojan	send out SMS to a premium number, delete incoming SMS from a certain number
Jifake	1	10/2011	Trojan	send SMS to premium numbers
jSMSHider	16	06/2011	Trojan	open a back door
KMin	52	10/2011	Trojan	sending out IMSI, phone number, OS version (sending SMS without any report from Google)

LoveTrap	1	07/2011	Trojan	sending out IMSI and geolocation
NickyBot	1	08/2011	Spyware	executing commands via SMS
NickySpy	2	08/2011	Trojan	sending out call list, GPS, SMS
Plankton	22 (11 in malgenome)	06/2011	Trojan	sending out device id, etc.
Pjapps	58	02/2011	Trojan	sending out IMEI, device id, etc.(sending SMS without any report from Google)
RogueLemon	2	10/2011	Trojan	send SMS and subscribe service
RogueSPPush	9	08/2011	Trojan	monitor SMS
SMSReplicator	1	11/2010	Trojan	transmit incoming SMS to another device
Zsone	12	05/2011	Trojan	intercept SMS and send SMS
SndApps	10	07/2011	Trojan	sending out phone number, email address
Spitmo	1	09/2011	Trojan	sending out SMS
Tapsnake	2	08/2010	Trojan	sending out GPS info
Walkinwat	1	03/2011	Trojan	sending out name, phone number, IMEI
YZHC	22	06/2011	Trojan	send SMS to a premium number
zHash	11	03/2011	Root exploit	root exploit
Zitmo	2 (1 in malgenome)	07/2011	Trojan	send out SMS

Table 1: Malgenome samples categorization

Table 2 is the result of the same process for the samples gathered from Contagio mobile malware mini dump.

Family	Number of samples	Month of discovery	Category	Description
FakeFlash	2	Q4 2013	Trojan	attempt to charge a fee for downloading and installing the free Adobe Flash Player
Mouabad.P	1		Dialer / SMS Trojan	premium dialer
Qicsomos	1	01/2012	SMS Trojan	send SMS
Tetus	4	01/2013	Trojan	steal information
VoiceChanger	1	01/2012	Dialer	premium dialer
Dropdialer	2	07/2012	Dialer	premium dialer
Fakebank	5		Trojan	steal information
Fakedaum	1	07/2013	Trojan	steal information
Jollyserv	1		SMS Trojan	send SMS
Roidsec	1	05/2013	Trojan	steal information
Trahanie_ofisnyih_rabot	1		SMS Trojan	send SMS
FakeInst	1	12/2011	SMS Trojan	send SMS
Flash fake installer	1	04/2014	SMS Trojan	send SMS
Fakemart	1	08/2012	SMS Trojan	send SMS
FakeNotify	1	12/2011	SMS Trojan	send SMS
OpFake	20	02/2012	SMS Trojan	send SMS
Xxshenqi	2	08/2014	SMS Trojan	send SMS
Simhosy	3	11/2013	Trojan	steal information
Skullkey	1	07/2013	Trojan	steal information

Table 2: Contagio Mini Dump samples categorization

For the sample families in blue letters no result could be drawn, because no sample was executed. The families in yellow background are the ones that were of interest to this paper (sent an SMS). Some samples weren't executed at all (*no activity* or *failed to execute* in droidbox). Also, the majority of apps was in Chinese, which was an inhibitory factor. Furthermore, even if it was reported as malicious activity, nothing concerning outgoing calls was observed.

The next flowchart diagram depicts how the normal sequence of sending an sms is differentiated when there is malware present.

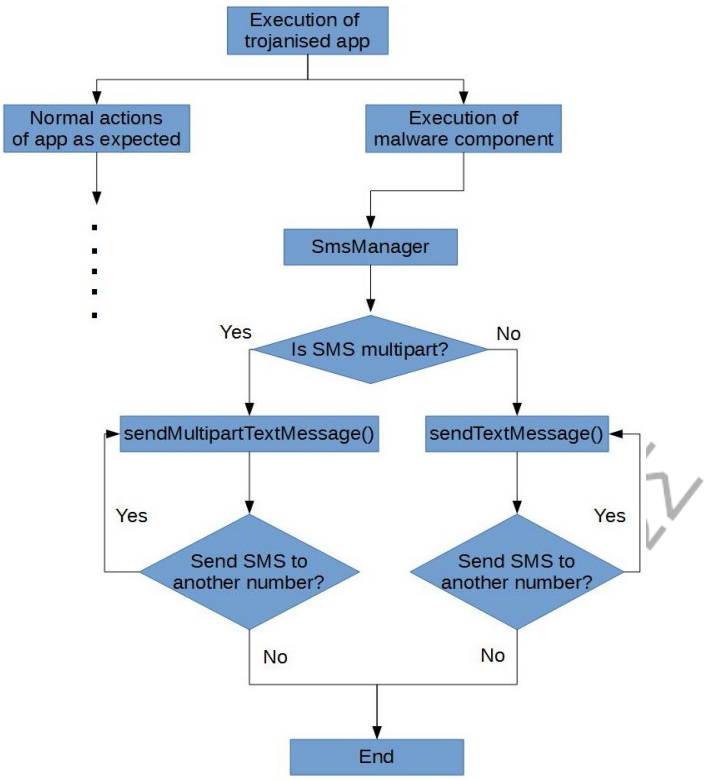

Figure 4: SMS sending with malware present

In a corresponding manner, the sequence is changed when the malware tries to place a call. Figure 5 shows how this is carried out.

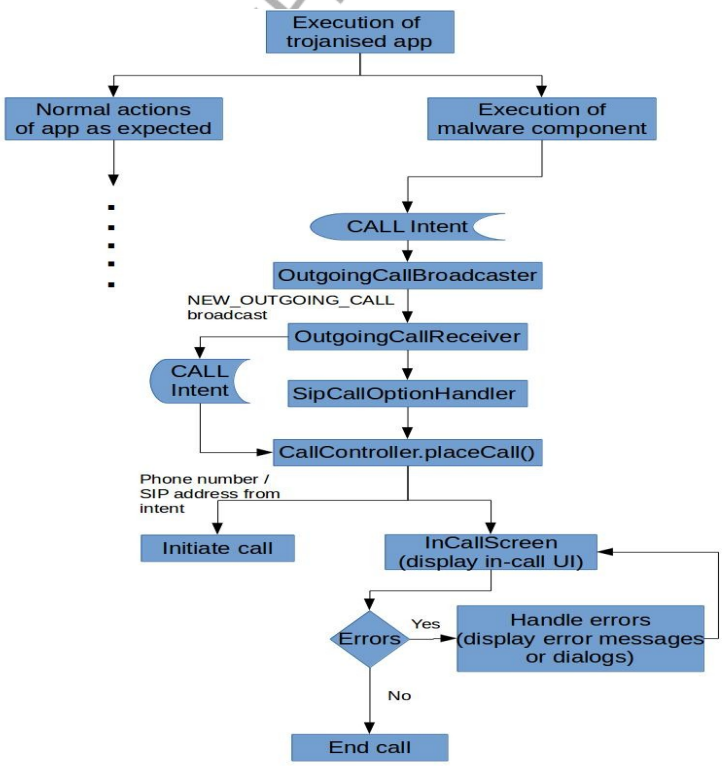

Figure 5: Outgoing call with malware present

3. CUSTOM MECHANISM DEVELOPMENT

The logic behind the design of the custom system that was developed is that all outgoing SMS messages and calls are intercepted before being completed, so that the user can detect all these actions and then decide if an app is malicious or not and proceed accordingly. When an app tries to send an SMS or place a call to a premium-rate number without the user's knowledge, a CAPTCHA [17] is used to prevent this type of malicious behavior of such apps.

3.1 Android source modification

After identifying all the relative samples, the actual coding began. All the tests and modifications were made in Android version 4.4.3.2.1.000.000 (as per the emulator) compiled by the source code following Google's instructions [18] in January, 2014. The CAPTCHA mechanism was divided into two subcategories:

3.1.1 Dialing CAPTCHA

The most typical outgoing call sequence is as follows:

1. OutgoingCallBroadcaster receives a CALL intent and sends the NEW_OUTGOING_CALL broadcast
2. Then SipCallOptionHandler is launched which decides for the type of the call (or lets the user choose) and ultimately calls CallController and the method placeCall() in particular.
3. In CallController.placeCall() the actual call is initiated and the in-call user interface is displayed.

In the custom mechanism, from SipCallOptionHandler.java a dialog is displayed where the user must pass a CAPTCHA puzzle in order to continue with the call. In case of emergency calls, there is no need for a CAPTCHA.

The logic behind this part of the mechanism is that there is a database storing all the whitelisted/blacklisted/emergency numbers. When an outgoing call is initiated, then depending on the type of the called number, a CAPTCHA dialog is displayed or not.

- If the call is characterized as emergency (the called number is an emergency number) then the call goes through without any further user interaction.
- If the called number is whitelisted, the user can either delete the number from the whitelist or continue with the call.
- If the number is blacklisted, the user can delete the number from the blacklist (after solving

a CAPTCHA) or cancel the dialog which ends the call .

- If neither of the above is the case, the user can add the unlisted number to the whitelist or the blacklist, or continue with the call without storing the number (decide later, if the number isn't called very often). Of course these actions also require the solving of a CAPTCHA puzzle.

The following diagram depicts the flow of actions from the user's perspective in order to place a call.

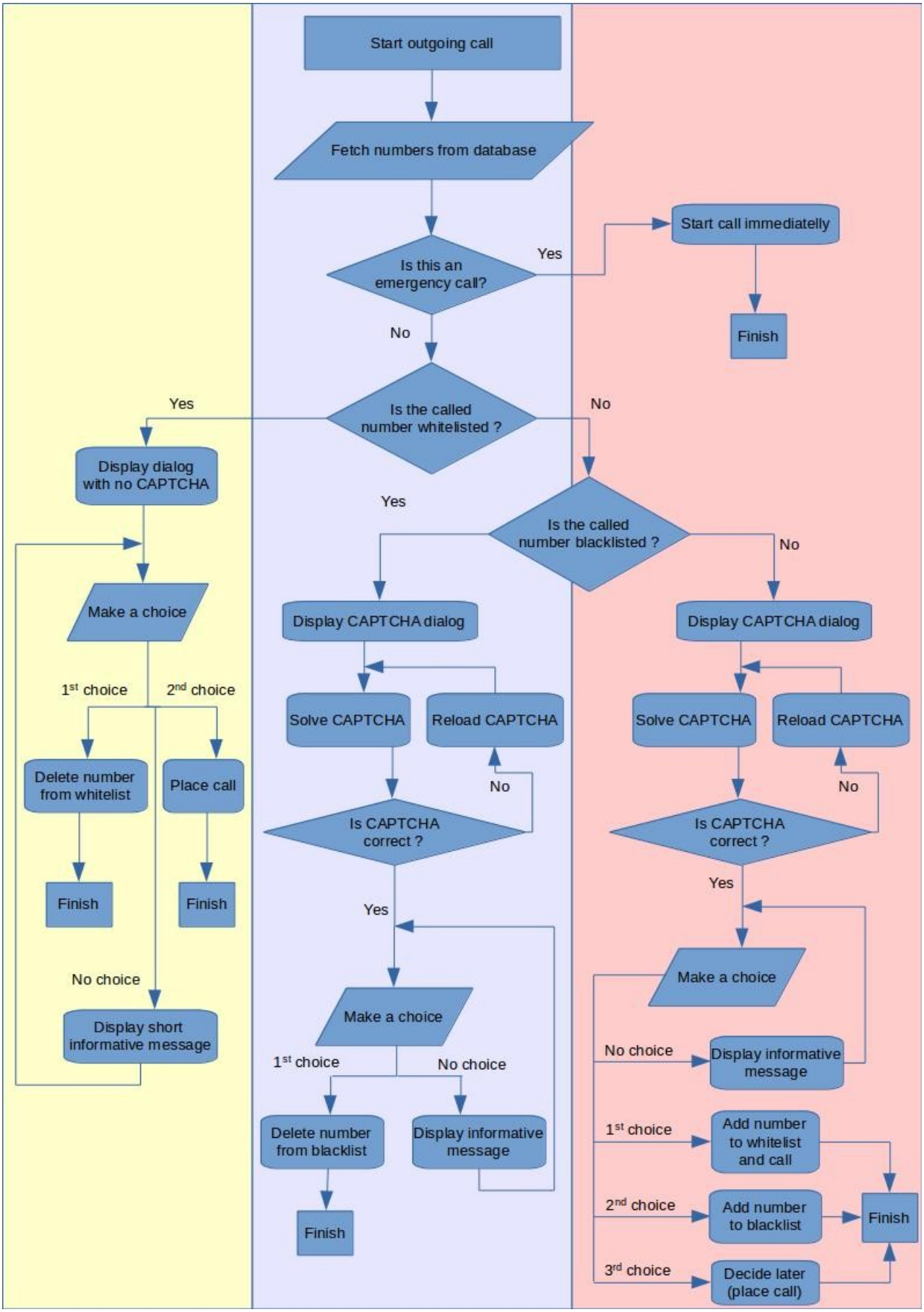

Figure 6: User actions for solving a CAPTCHA puzzle when making a call

3.1.2 SMS CAPTCHA

There are two ways of sending a text-based SMS from a third party app. One can either invoke the SmsManager class and use the *sendTextMessage()* / *sendMultipartTextMessage()* functions, or launch the built-in Messaging app which can be pre-populated with the destination number(s) and the message body. Of course all malware use the first approach, which sends the SMS in the background and doesn't display anything to the user.

So, there were two approaches for the development of the CAPTCHA mechanism, depending on what way of sending an SMS was used. Due to the fact that a dialog cannot be displayed from within the SmsManager class, a custom built-in system app was created that is launched whenever there is an attempt to send an SMS using this way. The customization of the Messaging app was that a dialog is displayed when the user taps on the “Send” button.

- In the case of the “Messaging” app, a database was created where all the blacklisted / whitelisted numbers are stored and it's private to and accessible only from this app.
    - When an SMS is about to be sent, all the numbers are fetched from the database and are compared against the SMS destination numbers. If all these numbers are whitelisted, then a dialog is displayed where the user can choose between deleting one or more of them from the whitelist (using checkboxes) and sending the SMS. These actions are performed without the need to solve a CAPTCHA.
    - If there is no blacklisted number, the user has four choices. He/ she can add an unlisted number to the whitelist or the blacklist (two choices), delete a number from the whitelist, or decide later (send the SMS). The choosing of numbers to add / delete is made using checkboxes in this case, too. Of course, it is necessary to solve a CAPTCHA so that these actions take place.
    - If the destination number is blacklisted, the user can only delete it from the blacklist, after passing a CAPTCHA challenge.
    - When solving the CAPTCHA, if what the user has typed is incorrect, then the CAPCTHA image is refreshed without taking into account if the user has made a choice from the ones available. If the typed CAPTCHA is correct and no choice is selected, a pop-up message is displayed informing the user to make a choice.

The sequence of user interactions with the system is shown in the following diagram.

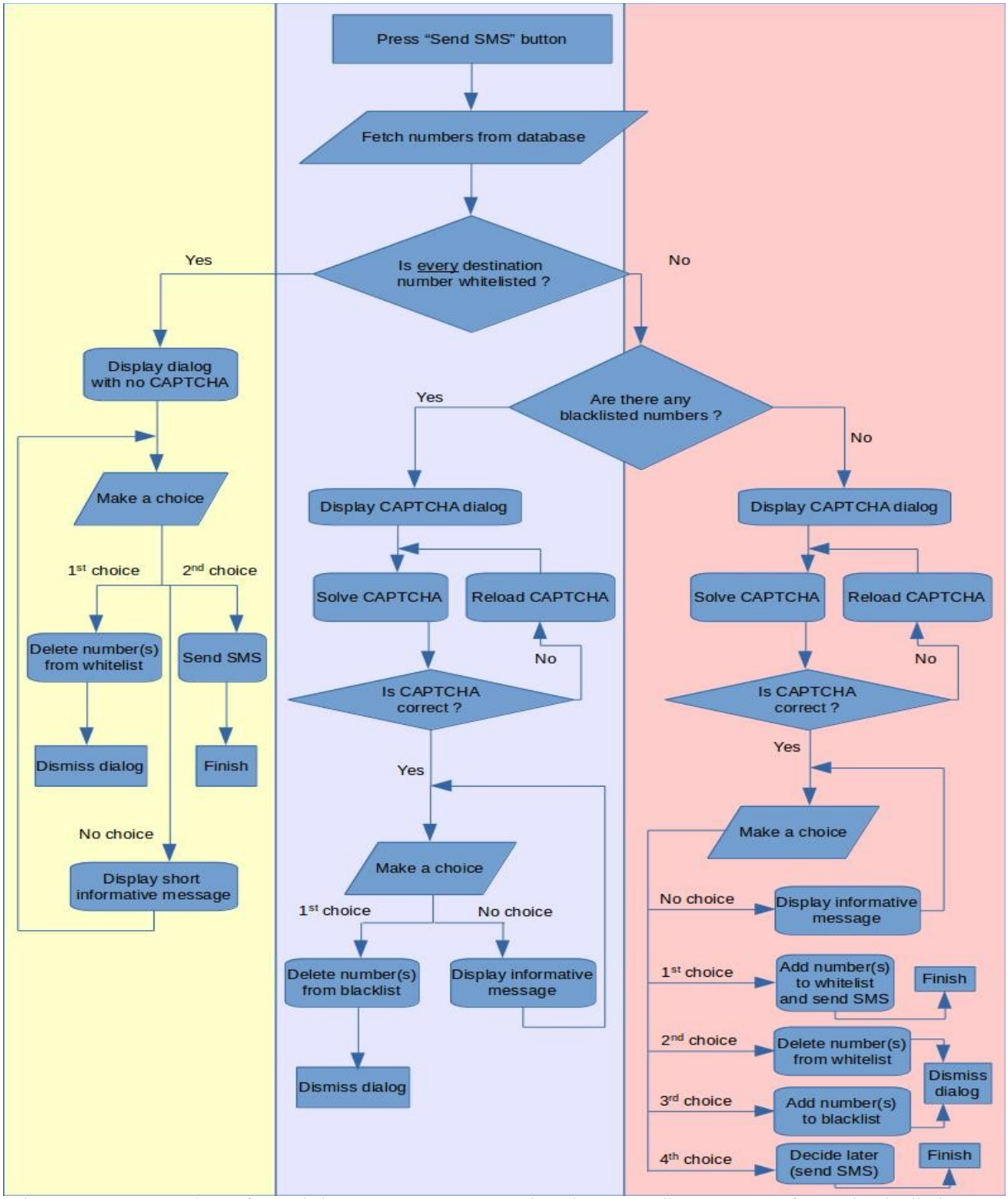

Figure 7: User actions for solving a CAPTCHA puzzle when sending an SMS from the built-in app

- In the case of the SmsManager class, if the application that tries to send the SMS is "Messaging", then if the CAPTCHA is solved correctly, the SMS is sent normally. The same logic applies if the app is the custom built-in system app. In any other case, the custom system app is launched.

Custom system app: The “Solve CAPCTHA” custom system app is launched every time an app besides the built-in tries to send an SMS. The thinking here is the same as the “Messaging” app's, with the sole difference that in this case it is not possible to send an SMS to multiple destination numbers at once (this can be done indirectly with consecutive calls of *sendTextMessage* or *sendMultipartTextMessage* functions) and so there are no checkboxes when performing add / delete functions to the database.

Again, there is a private database to store all the white/black listed numbers. If the destination number is whitelisted, the choices presented are for deleting the number from the whitelist or sending the SMS. If the number is blacklisted, the only available choice is to delete it from the database after solving a CAPTCHA puzzle. If the number is unlisted then the choices are 1) add to whitelist, 2) add to blacklist and 3) decide later (send the SMS). All this, also, requires solving a CAPTCHA.

Following the logic of the “Messaging” app, if the CAPTCHA is incorrect its image is renewed. If it's correct and the user hasn't tapped on a choice, he is informed to do so.

The next flowchart shows these actions in a graphical manner.

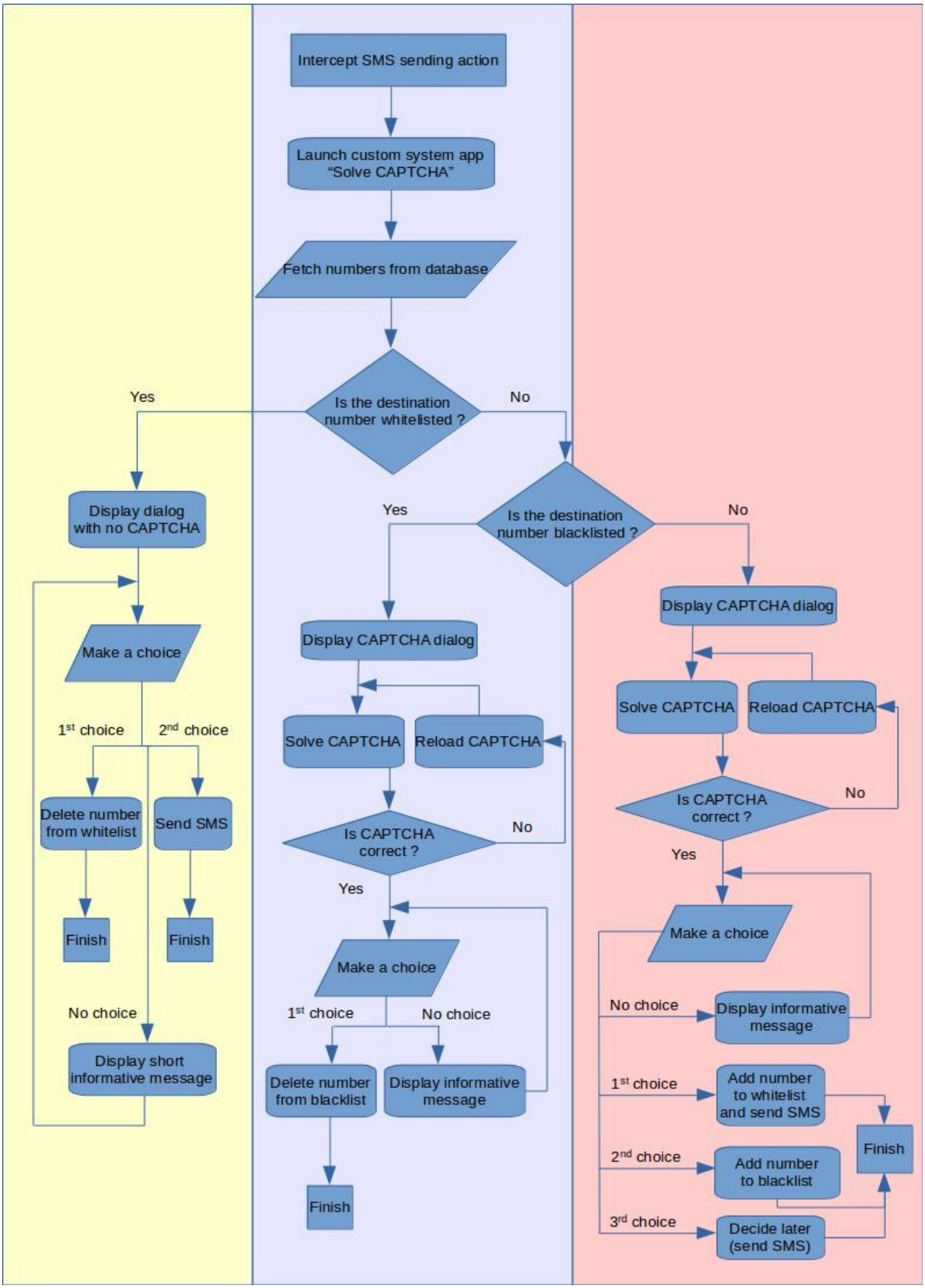

Figure 8: User actions for solving a CAPTCHA puzzle when sending an SMS from a third-party app

4. LIMITATIONS

These modifications / additions of code increase the level of security users can have on their mobile devices by intercepting all attempts made “behind their backs” by malware, to incur high charges to their bills. However, it is still a software “patch” that can be circumvented / tampered with given the right circumstances.

One possible limitation of the proposed CAPTCHA system is tampering with the databases where the whitelisted / blacklisted numbers are stored. Under normal conditions, an app's database is private to and accessible only from that app, unless stated otherwise programmatically by a flag, during the creation of the database. Of course, for the needs of this custom system all the databases are private. Nevertheless, if the Android device is rooted – which means that the user can have root access to the underlying Linux core – there is a possibility that a malicious app could modify the blacklist / whitelist databases.

Rooting a device is performed for customizing that device for optimum use. Root applications are applications that run on rooted phones. There are legitimate root apps that can extend battery life, make the device run faster and smoother and customize the internal systems to increase performance. In order that these applications run, they require root access that a user can give or not. If malicious code is packaged along with a legitimate root app and the app requests and is given root access from the user, then inadvertently root access would also be given to the malware part of the app. Then the malware could insert a number to the whitelist, fooling the user into believing that this number has already been processed and whitelisted, or delete a number from the blacklist so the user doesn't recognize that number as dangerous.

For demonstration purposes, two apps where created. The first one (CreateDB) simply stores some numbers in a database (in correspondence with the private whitelists / blacklists). The second one (ReadDB) is a root app (in correspondence with a malicious app) that tries to access the database of the first one and insert a number. If the user grants root access to the app, then the private database is modified, storing a number that it was not supposed to.

5. FUTURE WORK

As explained in the previous section, one possible limitation is that on a rooted phone when a legitimate root app is given root access to run, the malware that is possibly packaged along with it is also given root access and is free to modify the databases and insert or delete numbers from them.

However, the root access that this requires is given by the user explicitly. When a root app asks permission to run, a dialog is displayed where the user must grant access to it by tapping on an “OK” button. If the malware is standalone and not packaged within a legit app, it could simulate the pressing of the “OK” button to get root access even without the user's approval.

The custom CAPTCHA mechanism could be modified so that the user must solve a puzzle before granting root access to an app in order to prevent the malicious ones from getting root permissions automatically.

6. CONCLUSIONS

For the purposes of this thesis a custom CAPTCHA system for the Android platform was developed, that intercepts all attempts made by built-in or third-party apps to initiate an outgoing call or send an SMS message and displays a dialog where the user must solve a CAPTCHA puzzle in order to continue.

Firstly, all the relevant samples from Android Malware Genome Project along with Contagio Mobile Malware Mini Dump were gathered and analyzed for malicious behavior that was pertinent to the scope of this paper.

Then, the development of the custom system took place. All the Android components responsible for placing a call or sending an SMS were recognized and modified accordingly and a custom system app was developed for the case where a third-party app invokes the SmsManager class.

Closing, to demonstrate that on a rooted device a malicious root app can interfere and tamper with the databases where the whitelisted / blacklisted numbers are stored, two simple applications were created – one that stores some numbers in its own private database and a second one that modifies the contents of the first app's database.

Even though the custom CAPTCHA system is safe and secure on a non-rooted device, the security of this mechanism can be circumvented on a rooted one. It still lies in the hands of the user to grant root access to a root app, so extreme caution must be exercised when downloading and installing apps, especially the ones that require root access to run.

REFERENCES

1. http://www.idc.com/prodserv/smartphone-os-market-share.jsp
2. Cisco, “Cisco 2014 Annual Security Report”, 2014 https://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf
3. F-Secure, “Mobile Threat Report”, Q1 2014 https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014.pdf
4. Kaspersky Lab and INTERPOL , “Mobile Cyber Threats”, 2014 http://25zbkz3k00wn2tp5092n6di7b5k.wpengine.netdna-cdn.com/files/2014/10/report_mobile_cyberthreats_web.pdf
5. Y. Zhou, X. Jiang, “Dissecting Android Malware: Characterization and Evolution”, http://www.csc.ncsu.edu/faculty/jiang/pubs/OAKLAND12.pdf
6. C. A. Castillo, “Android Malware Past, Present, and Future – McAfee White Paper”, http://www.mcafee.com/us/resources/white-papers/wp-Android-malware-past-present-future.pdf
7. http://www.symantec.com/security_response/writeup.jsp?docid=2010-081100-1646-99&tabid=2
8. http://www.csc.ncsu.edu/faculty/jiang/HippoSMS/
9. http://home.mcafee.com/virusinfo/virusprofile.aspx?key=544065#none
10. http://www.symantec.com/security_response/writeup.jsp?docid=2012-073021-4247-99&tabid=2
11. https://blog.lookout.com/blog/2013/12/09/mouabad-p-pocket-dialing-for-profit/
12. https://blog.lookout.com/blog/2011/03/01/security-alert-malware-found-in-official-android-market-droiddream/
13. http://www.malgenomeproject.org/
14. http://contagiominidump.blogspot.com/
15. https://code.google.com/p/droidbox/
16. https://sites.google.com/site/androidmalrepo/
17. https://github.com/floydfix/Android-Easy-Captcha
18. https://source.android.com/source/initializing.html