

Final Year Project Report

Full Unit - Interim Report

Detecting Android Covert Channels

Katrina Bell

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science (Information Security)

Supervisor: Jorge Blasco Alis



Department of Computer Science
Royal Holloway, University of London

December 1, 2017

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 9201

Student Name: Katrina Bell

Date of Submission: 1/12/17

Signature:

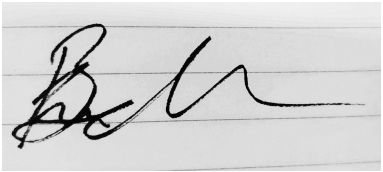
A handwritten signature in black ink on lined paper. The signature is stylized, starting with a large 'K' and ending with a long horizontal stroke.

Table of Contents

Abstract	4
Project Specification	5
1 Introduction to Covert Channels	6
1.1 Covert Channels: Timing Channels	6
1.2 Covert Channels: Storage Channels	8
2 Types of Covert Channels in Android	9
2.1 Audio/Volume Channel	9
2.2 Screen Brightness Channel	9
2.3 Vibrator	10
2.4 Battery	10
2.5 CPU	11
2.6 File Locks	11
2.7 Phone Call Active	11
2.8 Phone Call Logs	12
2.9 Microphone	12
2.10 Review of Channel's Success	12
3 Overview of Detection of Covert Channels	14
3.1 Modeling Normality	14
3.2 Other's Signatures and Methods	14
4 Analysis of Existing Systems	18
4.1 Overview of SoundComber	18
4.2 Audio Channel Implementation	18
5 Project Diary: Objectives and Achievements	20
5.1 Week by week	20

5.2 Self Analysis 23

6 Applying Software Engineering 24

6.1 SVN and Online Repositories 24

6.2 UML 24

6.3 Testing 25

7 Professional Issues in Computer Science 26

7.1 Introduction to Professional Issues 26

7.2 Android Project Specific 26

Bibliography 28

Abstract

Android Covert channels are a threat to most technological devices in the modern day. There are so many that we know about, leaving many more not yet discovered. This Project will be reviewing the threats that they bring us, systematically evaluating which channels are more likely to be used and perceiving which are better at communicating confidential information. The majority of discovered covert channels are either a timing based channel or a storage based channel.

This report will address what timing and storage channels are and how they work. While further evaluating a list of channels: Audio, Screen brightness, phone call logs, Battery, CPU, file locks, active phone calls and microphone. In this particular case, file locks and phone call logs are shown to be the most successful at avoiding detection due to the data transfer's ability to hide their intention well. The report provides a detailed analysis of existing systems as well as an overview of efficiency of the channels and likelihood of detection, while explaining detailed detection techniques and how they can apply to android covert channels.

Project Specification

0.0.1 Aims

To implement an Android app that is able to detect apps using covert channels to communicate.

0.0.2 Background

Inter-process communications in Android are, by design, mediated by the operating system. However, as in any other operating system, some features of Android (and its UNIX foundation) can be used for covert channel communications. These kinds of communications could be used by colluding applications to synchronise and execute attacks. However, most of these channels are accessible by all apps installed in a system, so it would be possible to implement an app that listens for possible communications over those channels and raises an alert if any communication is detected through them.

0.0.3 Early Deliverables

1. A report listing the currently known covert channels in Android and methods to detect them.
2. Proof of concept implementation: Two Android apps that use one of the listed covert channels to exchange information.

0.0.4 Final Deliverables

1. Android app for detecting covert channels in Android.
2. Android apps to send information through covert channels to evaluate the Android Covert Channel Detector.
3. Comparison of the throughput of the different covert channels available in Android.
4. Evaluation of the accuracy of the Android Covert Channel Detector.

0.0.5 Suggested Extensions

1. An option to write random data to the covert channels to avoid other apps using them.

0.0.6 Suggested Reading

1. Schlegel, Roman, et al. Soundcomber: a stealthy and context-aware sound trojan for smartphones. NDSS. Vol. 11. 2011.
2. Chandra, Swarup, et al. Towards a systematic study of the covert channel attacks in smartphones. International Conference on Security and Privacy in Communication Systems. Springer International Publishing, 2014.

Chapter 1: Introduction to Covert Channels

Security has been an increasing issue in technological devices over the last 10 years. Whether it is cars, computers or phones, the ability to get away with no security on commercial devices has decreased massively and now security is inbuilt into most devices and Applications. Now with mobile phones being bigger than ever, security is a major factor in the development of these devices.

Mobile phones have both overt and covert channels. Overt channels are a form of legitimate inter-process communication that is not hidden down a channel, as opposed to Covert channels, which are a form of communication that is hidden down channels; usually the communication is not meant to be legitimate. Even though the development of phones is getting better every release, Covert channels remain in every smart phone to date. Covert channels are the hidden communication between two entities: the medium for two colluding apps; the potentially damaging inter-process communication of sensitive data. Its exploitation is the explicit transfer of data in a between processes that aren't going through overt channels (i.e. the legitimate, non-evasive way of inter-process communication). These channels obscure sensitive data among normal data in its communication, therefore making it difficult to detect if you aren't looking in the correct tunnels.

Due to android being a huge market used today for Smartphones, the security of said devices needs to be impeccable. Holding over 85% of the market for Smartphones (increasing roughly 10% each year), with a flaw like this, any user is at risk of having their data exploited. Whether it's changing your screen brightness, your phone volume or even your battery levels: android covert channels exploitation may send and receive data across applications. As seen in 'Soundcomber' [4], a user's sensitive data may be communicated via these channels to a malicious App. The malicious App then can manipulate the data however it likes, for example, send it via the Internet, in this case, using only the microphone in the device. Other channels like the volume and screen brightness are more obvious, however there are many that are well hidden (e.g. sending TCP packets).

In this chapter, I will be systematically discussing the two main types of covert channels, with examples of which ones I will be implementing, with detection methods to block and discover the possible exploitation of said channels.

1.1 Covert Channels: Timing Channels

A timing covert channel is a channel to which data and/or resources oscillate the specific amount of times it takes to process (slowing down and speeding up processes) and this can be sent via the encoder app. In this example, there is one decoder app and one encoder app, however in reality, often there can be multiple of each. Each manipulating the data as they see fit. The encoder app, in this case, has access to the sensitive data while the decoder app does not. Provided that the decoder app (or the receiver app) is able to detect this use of the channel, it may 'decode' and re-assemble the bits in order to obtain the unauthorised data. This is often executed by concatenating the byte values of the readings of each channel: almost simulating a sensor of byte value readings. The byte value readings could be, for example, the timings of the sent packets or the changed values of the CPU usage. This is all displayed in 1.1.

In Figure 1.1, you may see that the data transfer is a timing channel as `t1` doesn't equal

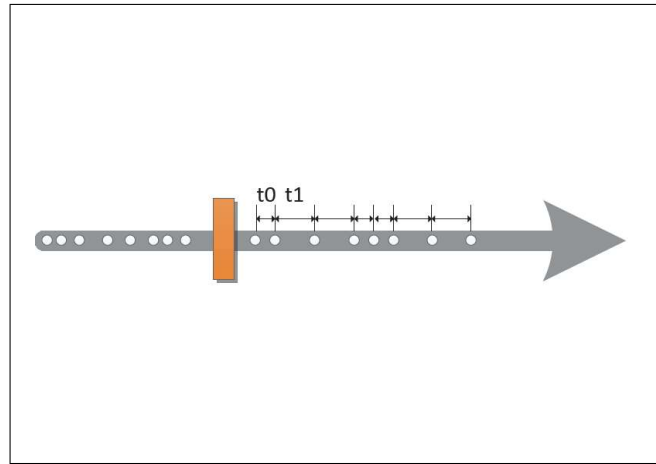


Figure 1.1: The timing channel model mentioned in [1]. It represents the analysis of timing channels: where the white dots represent the data that is being sent and the t_0 and t_1 represent the time difference of sending the data and the further disparity of sending.

t_0 . The white dots are the packets or activity that is being sent, while the rate to which they are sent is noteworthy to the decoder app (using the decoder protocol); therefore, the manipulation of the timings between the sent packets could be sending private data in the case of malicious use. This is usually coded, both the encoder and decoder app, on a timer, e.g. changing the screen brightness with encoded bits in a set time period and reassembled in the decoder app to unveil the data.

```
public void toTime(final byte[] input){
    final Timer t = new Timer();

    t.scheduleAtFixedRate(new TimerTask() {
        int i = -1;
        @Override
        public void run() {
            i += 1;
            if (i >= input.length) {
                t.cancel();
            } else {
                toSend(input[i]);
            }
        }
    }, 0, 1000);
    toSend(input[0]);
}
```

Above is my implementation of the use of timing reference in a channel. Due to both applications needing either a real time clock or a reference to time, this is the latter. As you can see, there is a new timer task and it runs every 1 second. To which calls the `toSend` method (which then, in turn, encodes the byte and sends it via the covert channel).

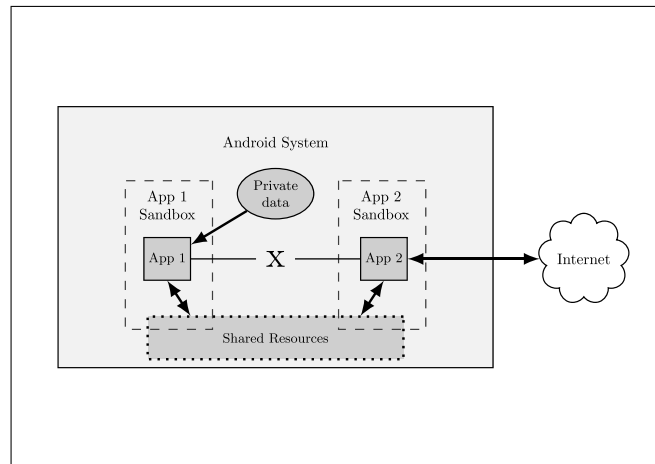


Figure 1.2: The threat model mentioned in [3]: two colluding Apps communicating through a storage channel.

1.2 Covert Channels: Storage Channels

A storage channel is one to which shared resources are changed temporarily, and which can be accessed by the decoder app. Provided both applications can access this shared resource, the decoder app may interpret the shared data however it likes. This is due to the dual access to the data, and the nature of the shared resources. 'Shared resources typically have some form of memory that can store encoded information' [3]. This infers that when using a storage channel, it is easy to hide sensitive data that you will want to maliciously send, using this encoded section of the shared resources. An example of the use of a storage channel would be to hide an image encoded inside another image that is being held by shared resources (a lot of apps have access to gallery hence being exploited).

The Figure 1.2 describes App 1 (the encoder app) communicating with App 2 (the decoder app). As you can see, the two apps cant send the data in question the traditional way due to permissions etc. So, App 1 temporarily changes the shared resources, encoding the private data and App two watches for changes and then uses and re-assembles (i.e. concatenates) the data their end. In this case, then proceeds (or has the ability to) upload it to the Internet for the use of other adversaries.

Chandra et al. [3] discussed and analysed the 5 main properties that one has to abide by in order to succeed in the implementation of a storage channel.

1. The soon to be edited mechanic of the shared resource must be accessible to both Applications.
2. The encoder/sender App must be able to edit and write to said resource.
3. The decoder App must have a function of viewing the shared resource, or the specific component of that resource.
4. The encoder App has a precondition of manipulating the detection time of the receiver App. When implemented, for example, this could be in the form of a timer or the click of a button on the Application, where the receiver knows when you are changing the shared resource.
5. Correctly sequencing the chain of events so that both Applications can initiate and communicate properly (using and establishing the resource to which will be the used channel).

Chapter 2: Types of Covert Channels in Android

The two main types of covert channel are storage and timing, each having many different implementations of the exploitation.

2.1 Audio/Volume Channel

Audio channels are one of the most researched covert channels. They involve using the API and the AudioManager API to hide the encoded data in the bits available to change the volume setting. The available bits there to change the audio volume depends on which component you are changing. For example, as states in [3], the `STREAM_MUSIC` range is 0 - 15 with a bit length of 4 whereas the `STREAM_VOICE_CALL` has a range of 0 - 5 so a bit length of 3. [3] also mentions 5 other volume components that could be changed to form a covert channel. The binary encoded data is used and split up into sections then the volume is changed to those figures in a given time frame t so the decoder app can 'record' the volume setting for each time frame t and concatenate the sequence of decoded volume settings and can reveal the message.

2.2 Screen Brightness Channel

Screen brightness channels work on changing the screen brightness setting that has 256 possibilities. This means you may encode a set of data (broken down into different sections) and change the screen brightness to that figure over a given mutually known time. This can be encoded in any form, unlike audio channels which can only be in binary. For example, encoding to ascii and splitting into sections that allow only 0 - 256, then sending them across every two seconds, where the decoder app reads the screen brightness every two seconds and stores the given data and decodes from ascii. Therefore, revealing the data (private or not) to the decoder app. This issue with this channel (bad for the malicious adversaries and good for security), is that screen brightness channels can be witnessed by the user (e.g. the user being on their phone at the time of the data transfer and witnessing the brightness change. Thus, in terms of covert channels, it is an easy one to spot.

```
public void toSend(int input){
    final int i = input;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            try{
                int bright = Settings.System.getInt(getApplicationContext().getContentResolver(),
                    WindowManager.LayoutParams.layout = getWindow().getAttributes();
                    layout.screenBrightness = (float) i / 255;
                    getWindow().setAttributes(layout);

                Settings.System.putInt(getApplicationContext(), Settings.System.SCREEN_BRIGHTNESS, i);
            } catch (Exception e){
                e.printStackTrace();
            }
        }
    });
}
```

```

    }
  });
}

```

2.3 Vibrator

The vibrating covert channel uses the phone's vibrator at different strengths, similar to the screen brightness channel, in order to transfer information. It usually changes every 1 second when implemented to allow the decoder app to read it. The different values for the vibration settings depend on the android device. The encoder app, when accessing sensitive information, encodes it into different bytes and sets the vibration setting to those values, sending them every 1 second. The decoder app, reads these values and concatenates them. This approach is usually too obvious to be used for sensitive data exploitation.

2.4 Battery

Battery levels decline respectively to the processes that they are running. The overall battery level decline is composed of all the process' battery discharge combined. This parallel use provides a discharge that is encoded [3].

As mentioned in [3], the receiving app must continuously monitor the ACTION_BATTERY_CHANGED variable, in the BatteryManager API, every 1%. The encoder app can manipulate the use of parallel resources in order to use the battery discharge rate as a way of communication. The decoder app will evaluate the discharge rate over a period of time and evaluate the rate while decoding the value. An example of parallel processes that could run are: Location/GPS, music player, high screen brightness, bluetooth or if it's an advanced/new phone, force GPU rendering. Any processes could work provided the same data gets encoded.

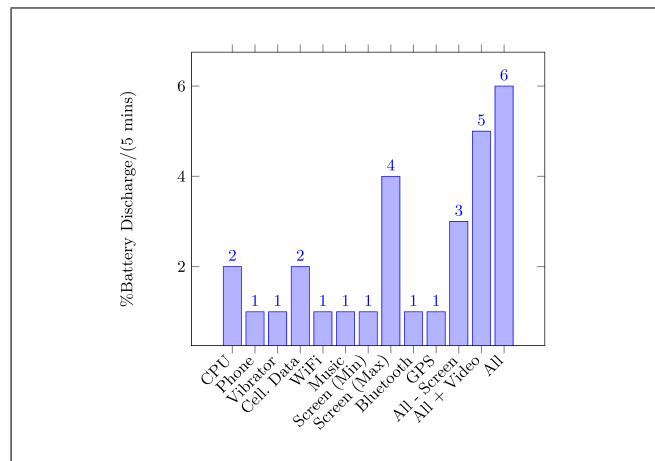


Figure 2.1: The component's level of discharge [3]

Tests using the Samsung Galaxy S, output Figure 2.1 statistics, according to [3]. The graph shows the combination of these processes can change the battery charge significantly per 5 minutes. enabling many different values can be send via this channel.

The issue with this method is, to transfer lots of data, it would take far longer than many other methods, due to the battery discharge measured every 5 minutes. Therefore, transferring

long pieces of data such as a user's address would take quite a while. Not to mention that it does not account for charging the phone in the middle of data being sent, which is a likely occurrence due to the time it takes to take an average

2.5 CPU

The CPU usage channel is implemented in a very similar way to the battery charge channel. It exhausts different processes at different intensities in order to differentiate between encoded values [9, 8, 12]. Processes must be evaluated by how timely they are to execute and are, in turn, evaluated on their CPU speed. The data is sent and received in the same way to battery channel, i.e. over a period of time established between the two apps, the encoder app manipulates the processes and the decoder app reads the CPU usage levels; further decoding that into plaintext.

The CPU channel is quite a diverse channel as it is not always obvious to the user that data is being sent. However, when this method is implemented, using a web application or otherwise, it uses a significantly higher bandwidth than other methods of covert data transfer. For example, the use of javascript in a browser can be implemented with a lower bandwidth, however task's bandwidth can go up to 4.88bps on an ideal system [12].

2.6 File Locks

In works [12, 4], both mention file locks as a particular occurrence of using covert channels. This is implemented by sending over a 1 or 0 by locking a file shared by both apps. One after the other, the encoder app locks the file to represent a 1 in the data that is being exploited, the decoder app then attempts to lock the same file, if it returns an error, then the outcome bit is 1, while returning 0 if the lock goes through. The methods that the implementation would be using is `READ_LOCK` and `WRITE_LOCK`. This particular covert channel is known for avoiding methods of detection and discovery such as Bell-LaPadula model. For further explanation, see section 4.1 with how the soundcomber app implements the microphone and file lock channel.

2.7 Phone Call Active

The phone channel uses `CALL_PHONE` permission to start and end calls for broadcasting sensitive information [3]. The `TelephonyManager` API states the use of `READ_PHONE_STATE`, `CALL_STATE_OFFHOOK` and `CALL_STATE_IDLE` in order for the decoder app to see if a call is active. The decoder app can see if calls are being taken place and then determine the frequency of those calls. The frequency determines the binary digit that is being encoded. This can happen due to the call state can be quickly turned off and on. Provided both the encoder app and decoder app are synchronised, the channel data can be decoded into the sensitive data.

2.8 Phone Call Logs

The Phone call logs channel is a storage channel that manipulates the encoded private data into phone logs [3]. Due to the `READ_CALL_LOG` permission, a decoder app has the ability to read the number that was last called, while the encoder app can call and end call quickly enough that it may go unsuspectingly. Within the phone log, each entry is an arbitrary length ASCII string, therefore allowing easy manipulation.

2.9 Microphone

The microphone channel works by changing the shared resources that both processes have access to. In the example of a microphone in an android device, the app's would need the permission: `RECORD_AUDIO`. The use of `MediaRecorder` means that the functionality of the covert channels works similarly to the way file locks work. Using the `start()` and `stop()` methods, the decoder can attempt to 'start' a recording and it throw an error which will dictate the encoder's app intention to send a 1 or 0.

2.10 Review of Channel's Success

This issue with some covert channel is, that some are too obvious for an actual implementation to go unnoticed. For example, the brightness channel, a user is easily able to spot that brightness changing and therefore not subtle for an attacker to implement. The audio channel would also be a bad implementation, due to the volume slider being visible to the user; while also being wholly obvious if the user was listening to music. In order to make this channel more stealthy, the adversary coding the app would need to put a check on if the user was listening to music or on the phone. This would enable this channel not to be used in a time when in use. however, all in all, these channels are not particularly stealthy channels. In report [4], they discovered a device that, due to wake-locks, is able to transfer information without the screen from actually being turned on (the G1 Phone). The G1 phone is notorious for having exceptional delay in executing processes therefore the phone is able to send information when the screen is turned off via the screen brightness channel.

In terms of the battery channel, it is too slow to implement. It usually takes (in the example above) around 5 minutes to get one reading of what applications takes what amount of battery life. Unless you use a model based on the user's phone, which is fairly unreasonable as you'd need a model for every phone and hardware combination to date. Once this occurs, you are able to send the data fairly easily however it is the process for modeling processes battery life that that makes efficient data transfer obsolete. With large amounts of data, it is not usable to transmit over this type of channel. This is fairly similar to the response of the CPU channel: measuring output of processes and acting accordingly. The analysis of both battery and CPU are similar, however although that is the case, this channel is one of the most stealthy. Not many users constantly check their CPU speeds or check why their battery decreases so fast, implicating, the high level of stealth this channel provides.

In paper [3], it conveys the phone call log channel may store arbitrary length messages in ASCII, therefore implying it being the quickest way to pass the largest amount of sensitive data across application through covert channels. With sensitive data such as an address, it is entirely reasonable to hypothesise that the channel would be the quickest at transferring

that data. The only issue I'd mention, is that apps have far more reason to have access to the audio permissions than phone-call logs therefore, making it difficult to not be spotted by anti-virus software for having unneeded permissions. This channel is reasonable stealthy in it's approach, especially in comparison to other channels. However a user may be able to spot some 'odd' information in a phone log that deems unusual, and therefore may be under suspicion by the user.

The file Locks channel has been claimed to be very difficult to detect, both by the user and detection software. In reality that is the case. Many applications have access to the same resources and from that many have file lock abilities. In order to detect the use of a file lock channel you'd have to predict the intention of a process when using locking files. This inadvertently is very difficult to distinguish. To add to this, most applications error check, therefore to the user and the program, usually it will go unnoticed. The communication using this channel can usually bypass methods of detection; One being the Bell-LaPadula model. Not only that, Android system detection only usually performs static permission checks, implying that the app would go unnoticed, whereas permissions such as a phone-call logs access maybe be spotted. The microphone channel's analysis is somewhat the same however is far more likely to be detected by static permission checks.

Chapter 3: Overview of Detection of Covert Channels

3.1 Modeling Normality

The detection process is based on actively monitoring covert channels at any given point. One method is to model the normal. This is when you measure what would be 'normal' for a device and everything outside that range is deemed suspicious. In the example of the screen brightness, the model of normal would be the timing of fluctuation between the levels of brightness. When the screen fluctuates often, it would be considered suspicious. An example would be if the screen brightness changes four times during 2 seconds.

Once this method is put into an application would be alerted if the brightness changed multiple times under a certain time parameter. In the same case of the screen brightness channel, an app that models the regularity of screen brightness finding the average and the upper and lower mark of the usual figures. In the case of an audio channel, modeling the normal would be harder. This is due to users having different volume settings for different ideal scenarios. For example, if headphones were plugged in then the volume levels would be lower than if the volume was controlling a speaker or on a call. Therefore, modeling the normality of volume would, for the most part, be less accurate however still doable. Modeling the normal time, where the volume changes every 10 seconds would be best. This is due to sensitive data such as a phone number is more than 10 digits long. If the average (in this example 10) is exceeded then the detector knows the channels may be transferring suspicious data and then proceeds to monitor it.

3.2 Other's Signatures and Methods

3.2.1 Regularity Tests

Another method of detection is to use others signatures of detection. As mentioned in [5], there are many different detection tests. Regularity tests are one of them. Regularity tests measure the constancy of timing based channels. In the example of a delaying packets to form a covert channel, the detector will measure if the packets have alternated the time to which they are sent.

$$regularity = STDEV \left(\frac{\sigma_i - \sigma_j}{\sigma_i}, i < j, \forall i, j \right). \quad (3.1)$$

This algorithm is mentioned in above, where it explains the regularity of inter-packet delays. If the resultant standard deviation of the channel is not within the bounds of regularity, then it would not be deemed regular and therefore suspicious.

3.2.2 Covert Flow Trees

A method that has already been modeled is a fault/flow tree. This is a detection process for storage channel which evaluates the data sent, in a tree. As seen in Figure 3.1, Porras and Kemmerer evaluated this tree with 5 main nodes: The Or-Gate, And-Gate, Goal Symbol, Operation Symbol and the Failure symbol.

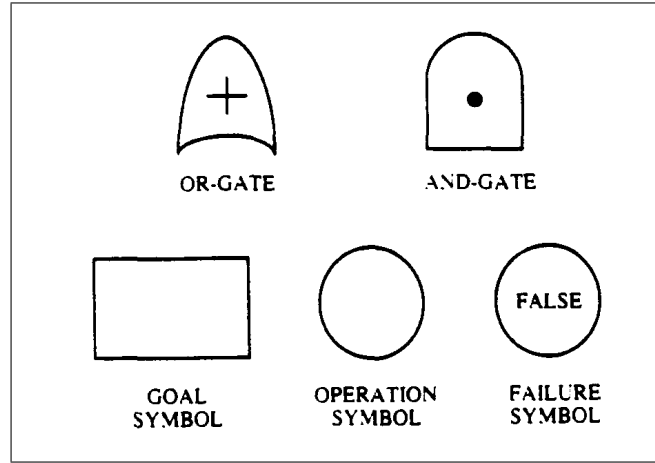


Figure 3.1: The Symbols that are implemented in the Flow Tree Approach [6]

1. This method describes the AND gate as when the child processes meet the requirements indicating the outward process also meets the requirements. It is primarily used for showing the relationship between the processes' Goal symbol, Operation Symbol and Failure symbol. The AND symbol works when there are at least two processes and one isn't already at it's goal symbol. This, of course, is all provided that the child processes are operations used in shared resources, as this model only works for storage channels.
2. The OR gate is quite similar to the AND gate in terms of requirements: the child processes must be operations, that are within the shared resource and that one isn't already at the goal symbol. The OR gate computes the outward process when at least one of the child processes meet the requirement.
3. Mentioned in [6, 7], 'Goal Symbols specify six possible subgoals: modification, recognition, direct-recognition, inferred-recognition, inferred-via and recognize-new-state'.
 - (a) Modification is when the goal of an attribute is modified.
 - (b) Recognition is the detection of of the change in the attribute's state.
 - (c) Direct-Recognition is the same as recognition however the change of an attribute occurs when the operation references it or calls a function that returns said attribute.
 - (d) Inferred-Recognition is the opposite of Direct recognition: where there is no direct reference or call to a function that returns it. The Example in [7] executes a method has more than one computation that depends on and relates to the attributes value.
 - (e) Inferred-via is a goal who's dependent on data being passed using a primitive operation from one attribute to another.
 - (f) Recognise-New-State occurs the Inferred-Via goal's data passing where multiple attributes have modified the goal.
4. The Operation Symbol represents System defined primitive operations. These are exclusively used as leaves of the tree.

5. The Failure symbol is for the the paths that have not been terminated with other symbols and an Operation Symbol, i.e. where all of the goal symbols do not recognise the condition.

As stressed in [7], the whole method is based on measuring information passing through attributes and processes, creating the tree using algorithms with primitive operations. This is formed by lists of operations, each with their reference list, modify list and return list. After all this information is computed, the tree is able to form and then proceed to be traversed into a form that is reasonable to read. Tree traversal is executed by splitting up the sequence of operations and the operations themselves. This traversal of data is designed to show possible covert channel exploitation areas.

In the example of a file system channel (more explained in paper [10]), this detection method would split the attributes of the system so they can assign the correct symbol and chart to the scenario. The three sections are file attributes, process attributes and a state attribute. For example (File_ID, Process_ID, Current_Process) [6]). For all of the operations processed, a tree is made to see what routes finish at which nodes. For example ending at an 'lock_file' node while another ending in an 'unlock_file' node would mean that there is a possibility of a covert channel there that could be exploited. A group of scenario's are made up in order to calculate the possible channels, which then can be assessed whether they are in use.

3.2.3 Shared Resource Matrix

A shared resource matrix is a detection implementation that maps different processes and files to what access and permissions in order to discover possible exploitation between processes and applications [10].

The methodology states that each resource's attribute is carefully examined for what permissions/primitives it has access to and further what can be used to transfer data. In this section, shared resource is defined as methods or objects that can be accessed (read or written to) by more than one process. Hence, why all of the individual attributes from the processes are shown in Figure 3.2.

RESOURCE ATTRIBUTE	PRIMITIVE	WRITE	READ	LOCK	UNLOCK	OPEN	CLOSE	FILE	FILE
		FILE	FILE	FILE	FILE	FILE	FILE	LOCKED	OPENED
PROCESS	ID								
	ACCESS RIGHTS			R		R		R	R
	BUFFER	R	M						
FILES	ID								
	SECURITY CLASSES			R		R		R	R
	LOCKED BY	R		M	R				
	LOCKED	R		R,M	R,M	R		R	
	IN-USE SET		R	R		R,M	R,M		R
	VALUE	M	R						
CURRENT PROCESS		R	R	R	R	R	R		

Figure 3.2: This is an example resource matrix shown in [10]

In this methodology, before the matrix could identify read/write pairs or write/write pairs for different processes, it must evaluate the different primitives each attribute could hold. For example, WRITE.FILE, READ.FILE, LOCK.FILE AND UNLOCK.FILE. This would allow the resource attribute to be narrowed down by their respective column as seen in Figure 3.2.

There are many approaches the matrix could take in order to successfully pair the exploited attributes, all mentioned in [10]. Some of the criteria for a storage channel to be found is:

1. Both processes must have access to the same resource.
2. The sending process must be able to alter or write to the attribute.
3. The receiving process must be able to sync with the sending process, i.e. know when the attribute has changed.
4. The processes must be able to initiate the communication, enabling the exploitation to synchronise between processes.

If there is a pair that meets the criteria, then that pair is found and a storage channel exists.

While the criteria for a Timing channel is as follows:

1. Both processes must have access to the same resource (similar to storage channel).
2. Both processes must have access to either a clock in real time or a time source.
3. The ending process must be able to manipulate the time to which the receiver process performs a response.
4. The processes must be able to initiate the communication, enabling the exploitation to synchronise between processes (similar to storage channel).

The issue is, storage and timing channels are a major part in the running of a system, therefore, once either of these channels are identified, it is not sensible to block it. The only time to block a channel is once the typical bits per second have been identified for a particular channel, and have been discovered overusing said channel. Although you could block it, you may also reduce the bits flowing through the channel by adding noise (random bits being sent) to the channel. Otherwise, if neither of the above are true, the channel's data transfer can be ignored.

An example of this could be analysing the decoder and encoder app's primitive function. For example, if the encoder app has an attribute that has access to writing to a file and the decoder app simultaneously has access to reading to a file, then the resource matrix would detect there being a possible channel between the two applications.

Chapter 4: Analysis of Existing Systems

4.1 Overview of SoundComber

SoundComber [4] is a proof of Concept program that uses only the microphone to transmit information through a covert channel. While one app listens in on call to sensitive data such as bank details, the other sends the sensitive information to the server. Doing speech recognition over the call recording takes too much power for the device and is unreasonable in its computation intensity, so soundcomber only works on selected times to process the speech and/or the buttons on screen, for the are a lot less intensive.

In order for this to work, the sender app must have the `RECORD_AUDIO` permission. This is so the app has the correct permissions to access the microphone. The receiver app must concurrently have the `INTERNET` permission, in order for it to send the stolen information to the desired server.

Soundcomber will be automatically initiated when a phone calls starts, by the operating system. The app doesn't need to be running in the background in order for it to run later on a call. The whole conversation is recorded unless there is a fixed size the data has to be, however this is no issue due to most banking etc. requiring you to speak or type you details in first, the colluding app is likely to store those details despite the fixed length. Hence, soundcomber is likely to have you account numbers, card numbers or even passwords.

After the recording has been made, it determines the number that the user was on the phone with, from identifying the type of receiver, it is able to distinguish if the receiver is likely to ask for account details or bank details, E.g. if it a sales company. This is further matched to which company it relates to by comparing the start of the recording with hotlines (further explained in [4]).

Soundcomber then uses a range of techniques to get the information from the voice clip: tone recognition and speech recognition combined to then further measure the result against a threshold of sound to receive the exact digits and letters used.

The soundcomber sender and receiver is a stealthy approach to sending data via a covert channel. In [4], there were two tests with seperate anti-virus software with neither of them detecting soundcomber as malware, as the anti-virus scans deemed it unsuspicious. Not only that, soundcomber was successful 55% of the time (over 20 different recordings) which is a relatively high success rate for an app that is exceptionally stealthy at using covert channels; if it wasn't as stealthy, that success rate would be quite insignificant.

4.2 Audio Channel Implementation

A user on github created an sender and receiver app that communicated over an audio channel [11]. When looking over there code, a few methods and coding lines described perfectly how an exploitation of a covert channel works.

This first snippet is in the sending application. This is where the creator initialises a broadcast receiver that will send off the the information through the channel. This application uses the `AudioManager` and the `AUDIO_SERVICE` to set the data within the variable `dataB` to the

volume of the device. This of course comes with checks to see if it's the first value which would turn to be null.

```
BroadcastReceiver receiver=new BroadcastReceiver() {
    AudioManager am = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
    int mode;
    boolean first=true;

    @Override
    public void onReceive(Context context, Intent intent) {
        mode = am.getRingerMode();
        if(first) {
            first=false;
        } else if(!first && mode==2 && i<dataB.length()) {
            send(am,Character.getNumericValue(dataB.toCharArray()[i]));
            i++;
            mProgressStatus=(int) (((double)i/(double)dataB.length()*100);
            mProgress.setProgress(mProgressStatus);
        }
    }
};
```

This second snippet is the receiver application which also initiates an AudioManager instance. When it runs it's receiving code the mode of the volume is checked before manipulation. Each value is acknowledge and is registered using the `registerReceiver()` method. This is then further displayed in a UI and thus, successfully transporting data via the audio channel.

```
BroadcastReceiver receiver=new BroadcastReceiver() {
    AudioManager am = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
    int mode;
    @Override
    public void onReceive(Context context, Intent intent) {
        mode = am.getRingerMode();
        if(mode==0) {
            bas=bas+0;
            am.setRingerMode(2);
        } else if(mode==1) {
            bas=bas+1;
            am.setRingerMode(2);
        }
    }
};
IntentFilter filter=new IntentFilter(AudioManager.RINGER_MODE_CHANGED_ACTION);
registerReceiver(receiver,filter);
```

The system when used successfully, transports data across a covert channel. It's implementation allows for simple use. MAYBE ANALYSE MORE

Chapter 5: Project Diary: Objectives and Achievements

My project diary was frequently updated throughout the past 10 weeks. Project diaries, as well as day books, are essential when trying to keep on track of a project's goals, so this section will go through the week by week, what I have done.

5.1 Week by week

In accordance with my diary at <https://pd.cs.rhul.ac.uk/2017-18/>, here is a more in depth analysis of what I did each week, take further detail from my day book, comparing my project plan to my real time progress.

5.1.1 Week Beginning: 2nd of October

The week prior to this was the delivery of the project plan that took roughly 5 hours to complete. The week of the second, I set up my SVN repository, creating new folders for my project: like a title folder IY3870-Project, and inside it a trunk and branches folder. Where I started and committed the first part of my report 'Android Covert channels and How to Detect them': Doing small sections of the description of the two main types of channel, while evaluating some individual channels such as the audio volume channel and screen brightness channel. According to my plan, this all fits accordingly and I was on schedule to continue my project and complete the next tasks. During this week I'd roughly done 7 hours total on my project.

5.1.2 Week Beginning: 9th of October

On the 9th of October 13:00, I had a meeting with my advisor. We discussed how reasonable my reports and there respective times were. Then we talked about the structure of my first covert channel report was going to be: including an introduction, 4 different channel descriptions, detection methods like the shared resource matrix and non interference method and issues with detection. All in all, it was very helpful in improving the flow of my report.

The week of the 9th furthered my covert channels report, completing the analysis of storage and timing channels while also starting some of the professional issues report. Although I was on schedule, there was only two weeks to do this report. I realised that, due to it being the only early deliverable report, needed to be long and in as much detail as possible. I made the decision to start the professional issues report early so I could write both the professional issues report and the covert channels report side by side in order to research more and go into further depth when explaining essential background knowledge. This would mean that the deadline for both would end on the 24th of October as oppose to the 16th. Another factor I ran into this week was: in my explanation for the two main types of channels, I wanted to insert some code as examples of the differentiation. This would entail that, for the report to be finished in it's entirety, it'd need to be after I did some of my coding for my proof of concept programs. In this week, I'd done roughly 8 hours of work including research time.

5.1.3 Week Beginning: 16th of October

This week, I decided to stay on schedule finishing the reports and leave spaces for coding and explanation related to the coding. Throughout the week I had come to finishing a few of the sections of the 'Covert Channels..' report. Some being the explanations of Screen brightness and audio channels, while researching more into the SoundComber [4] application and how it uses the microphone channel. This also opened my eyes to new channels like: file lock channels and explained them in more depth. In this week, I'd also done half of my professional issues report. Ive done a total of 4 hours on the covert channels report and 1 hour on the professional issues report. Throughout this week, I struggled to properly structure the professional issues report and decide what sections should be included, so I organised a meeting with my advisor to further improve structure.

5.1.4 Week Beginning: 23rd of October

I organised a meeting with my advisor on 24th of October 13:00. This meeting was to further my knowledge of the flow of my Professional Issues report. In the meeting, we talked about ethics and specifically human ethics of implementing covert channel apps onto a phone. Data privacy, plagiarism and licensing code using GPL. Further going into the mention of trust, access and testing when referring to privacy and ethics.

This further meant I finished my professional issues report and started my coding for my initial program (proof of concept programs). Id done a total of 10 hours that past week and a half on various different aspects of my project, including report and programming. When I finished my professional issues report, I realised I needed to refer more to my own project so that delayed me somewhat in my plan. From this, I calculated that I was behind slightly due to the original deadline being the 24th for both my reports. Therefore, taking an average of 10 hours of work per week, would make me roughly 10 hours behind in my work.

5.1.5 Week Beginning: 30th of October

In my plan it said to do my 'Analysis of Existing Systems' report at this stage however I thought it'd be better if I switched the tasks in my plan so I had more knowledge of implementation when reviewing other's work. Meaning the new due date would be the 28th of November.

During this week, I decided to use my SVN repository in order to copy and move my folders/files over; start working on my project and apply my knowledge I'd gained from research. As my program was already set up from the previous week: with an APK level of 21, Android 5.0 and with a nexus 5 emulator running Android 7.0. One of the main aspects I'd done this week, was moving my current reports into overleaf: applying Latex to my project and continuing it in said program. Overleaf has a auto save function enabled so there was no need to move my document to the repository. Due to other coursework pieces, I was unable to meet a target of 10 hours that week, meeting less than 6 hours work. In hindsight, this was an incredibly bad idea due to the future hours I've had to put in, in order to complete everything.

5.1.6 Week Beginning: 6th of November

In the advisor meeting on the 9th of November 10:30, we discussed how I could neaten my reports, writing them to improve the flow and producing the document in latex.

This week Ive continued my proof of concept programs, attempting to adapt ways to implement multiple different covert channel exploitations. The previous week had been very busy with other pieces of coursework so I am behind, and during this week, my code came across an error when attempting to change the screen brightness. In the later entries, I found out why. I also needed to change some repository folders as I believe when I create the second app, the files from the first will be mixed up and may corrupt the program: due to the locations, being the same. This week I had only done roughly 5-6 hours of work rather than the ideal 10.

5.1.7 Week Beginning: 13th of November

This week, I found one reason why my code wasn't working: I hadn't changed the `AndroidManifest.xml` file to contain the `WRITE_SETTINGS` setting. This is one of the components that means the app may write to the screen brightness setting. Clearly being an obstacle in my sending program. I also wrote some of the aspects to my 'Analysis of Existing systems' report (with the modified due date of 28th of November). During this week, I'd done roughly 10 hours of work, however most of them were being stuck on the permissions that the android app needed to be acknowledged.

5.1.8 Week Beginning: 20th of November

This past week, I have almost completed my final report including all sections other than the diary and the software design section. I managed to fix another error that was correctly asking for the permissions in the application, however the 'pop-up' did not appear when I ran the program. Meaning my code for setting the brightness could not run as the app didn't have permissions. In my advisor meeting, on the 21st, we attempted to fix this by running it in the Ui's thread, however that didn't work either. At the end of the week Ive done roughly 20 hours this week on: research, reports and coding combined.

5.1.9 Week Beginning: 27th of November

within this week, I fixed the bug that occurred in my code: due to my project plan meaning my App had to be applicable back to android 5.0, some of the functions I was using caused errors that wouldnt occur in android 6.0 and onwards. To solve this I simply changed the APK level to 23 rather than 21. The errors that had occurred with android.systems.design now dont exist, and the functions to check for permissions now work.

Due to the project the changing of the minimum version of android, my project would not merge or commit due to errors in the apk files. Seeing as they arent editable, I was unable to revert far enough to change that and commit. I saved a private copy of my project, deleted my local copies, checkout the new trunk and changed the versions of android straight to the trunk. While coping my code from the private copy over to the trunk (as I know it works). I proceeded to get error when running the new code as it threw an error running the apk however I changed the instant run setting in the settings menu and it fixed it. Unfortunately, this set me back quite far in terms of svn commits, however I fixed my project so it runs.

However, now that the APK has changed, my Applications weren't committing to SVN. I had spent many hours attempting to fix this SVN error, which seems unfix-able. I attempted to solve this by re-checking out my project and changing it from the trunk however that also didn't work. As a back-up, I have saved both my projects on my personal Github account, until I am able to fix this error.

Within this week I also wrote up some UML diagrams as well as my Software Engineering section, edited all my sections so they were the most descriptive way I could think of. ++ This week I have done roughly 35 - 40 hours of productive work.

5.2 Self Analysis

I believe, although my project has gone well and my plan was ideal, I didn't account for other coursework during each week. Not only that, I didn't account for major issues such as SVN commit issues. Because of this fact, when I spent less time on my project at the beginning, my plan was delayed so I had less time to fix errors and also had to spend over double the amount of time to complete my project.

Chapter 6: Applying Software Engineering

6.1 SVN and Online Repositories

Subversion is an online repository system that allows users and user groups to update, add and delete files to a project. It splits working code and edited code separate: a trunk folder and a branches folder. The trunk folder has one copy of the program that runs and has no errors, while the branches folder can have many versions of the program all editing different parts of said program. When making a branch, you copy folders from your trunk into the branch, and once edited, merge back into the trunk. Once a final working program works with a fully complete added feature, with all the intended features desired, you may tag your project. This can happen many times in a program's development.

My use of SVN has been to mainly back up my work I've created. I've branched, copied and merged throughout the last 10 weeks of somewhat frequency. It would have been better if I committed more however that is an improvement I shall have to do next term.

6.2 UML

6.2.1 UML for Sending/Encoder App

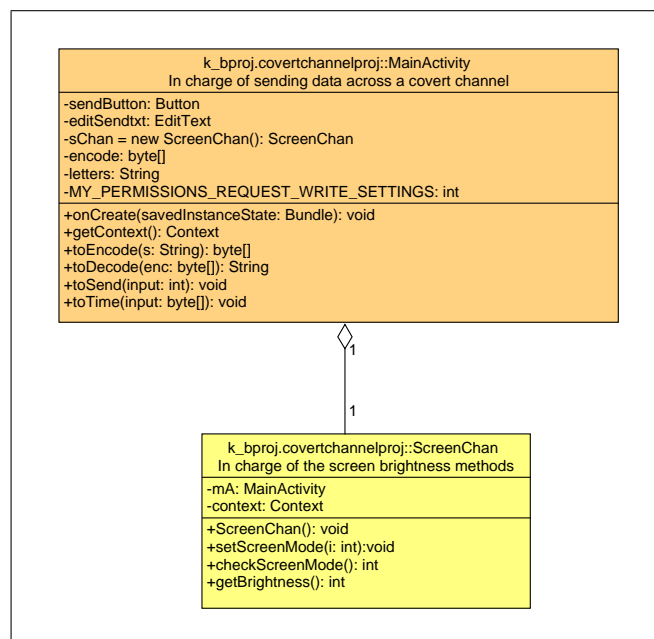


Figure 6.1: UML for Encoder application

The 6.1 UML diagram consists of two classes, one that deals with all the screen brightness specific methods and the other being a general main method that can implement multiple channels if I want to further adapt my program. There are no design patterns implemented due to it being such a small program, however, I may have been able to implement an adapter pattern so that MainActivity implements ScreenChan directly but of type ScreenChan. As you can see above, ScreenChan has an aggregation relation with MainActivity. This simply

means that the ScreenChan 'has a' MainActivity.

6.2.2 UML for Receiver/Decoder App

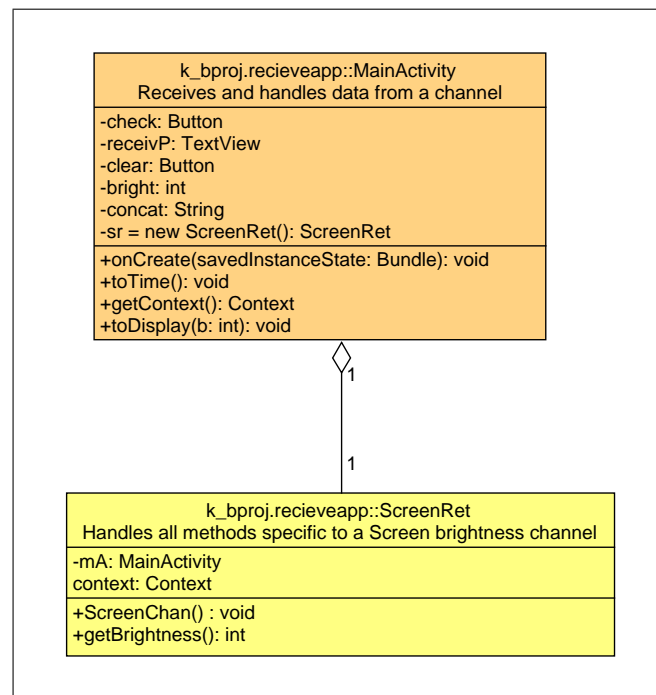


Figure 6.2: UML for Decoder application

The 6.2 UML diagram is very similar to the first, in terms of relations with each other. It's the same aggregate relation between the two classes like in 6.1. One of the only differences is within the the methods where one is sending information and the other is receiving this information.

6.3 Testing

Testing is a major part to software development as you need to quarantine all parts of the code for possible failure. This produces good stable code and coding practices. Keeping up maintainability of software is also key, so clear concise TDD improves many people software.

In this project, Junit tests were made on each method. A Junit test is a form of unit test for programs that use the java language. As my project was android development, not only did I have to do unit tests and flag tests, I had to adapt to doing non documented test using an emulator or my phone.

Chapter 7: Professional Issues in Computer Science

7.1 Introduction to Professional Issues

It is easy to be complacent when it comes to ethics and issues in computer science. The repercussions of ones actions are, for the most part, unknown to the average software developer. However, there are some cases where the repercussions are emphasised a lot: for example, when programming systems on an aircraft, where the safety of the masses are at risk, therefore are considered highly when programming. However, for the average software developer: security, users safety and wellbeing (physically and emotionally) are often forfeited over usability and design.

7.2 Android Project Specific

As my project is relative to android devices, and the majority of people use their phones to hold very sensitive data including name, address, phone number etc. My android app allows me to send and receive data across the device to different apps including data such as names, addresses etc. Therefore, this project has many privacy factors that one must be ethical with. A few examples that will be mentioned are: testing using the testing ethics, Access ethics and human ethics.

7.2.1 Artificial Data Use

One of the main examples that is displayed, is testing with sensitive data. The concept of exploiting covert channels is to send sensitive data from one app to another in order to use that data in an app that may not have permissions. In terms of the user, or owner of the testing device, the data may indeed be the users address, phone logs or contact numbers. The app I make specifically will send a phone number in the form of screen brightness channel, to an app that doesnt have permissions. Professional issues occur when a software engineer is manipulating real users data irresponsibly. To avoid this risk of abuse or misuse of sensitive data, I will use artificial data instead of the users private data, e.g. phone numbers, contact information or call logs.

7.2.2 Human Ethics: Opting Out

Invasiveness of this project's tests are a risk that may arise. Using others devices to test on has a level of human ethics that needs to be abide by. As my project is specifically about sending, receiving and detecting sensitive data exploitation, I may use others devices to test on in order to check that my programs work across different platforms. However, the devices I use will only be from close friends who always have the option of 'opting out' of the testing process, where they will receive their device back.

Abuse of access is another major component of professional issues for my project; although

the user of artificial data is often applied, a software engineer still has access to many of the files and sensitive components in the android system. In the case of a storage channel, the editing of shared resources could greatly affect the phone while allowing the misuse of sensitive data. In this project, there will be tests that temporarily change shared resources, i.e. encoding artificial data among the mechanics and data that is situated in the shared resources. Similarly to the previous point, when testing on others android devices, the user must have a certain level of trust in me to potentially change the shared resources on their phone. As mentioned before, the testing devices will be either my close friends or blank phones. If the former is to be true, then the users may opt out any time; if its the latter, it would avoid adding the risk of badly manipulating shared resources as well as being better for human ethics.

This project is backwards compatible to android 5.0, therefore, I would require different version devices to test on. The point's I've made previously before apply, however as I need to check every version of android, it would be better if I avoided this risk using an emulator instead.

7.2.3 Plagiarism: Citing

A common issue in computer science is plagiarism. In the industry of software engineering, coders do use code snippets from others work and but must use the correct citation due to possible licencing disputes. There are no exceptions for my project as I must obey the licencing laws and correctly cite. When researching possible covert channels to implement and the ways to implement them, there are a few projects that possess functions that will be useful and are seemingly efficient. Professional issues would occur if I dont cite my code correctly, therefore, if/when I use snippets from these repositories, I will correctly cite their code and whether it is licenced or not (including acknowledging the creator, the date made etc).

The majority of programmers use online repositories for their work and from that a large number of them use GitHub with their repositories open and public, making it is easy to steal and plagiarise someone elses code. Even though I am using a private repository (Subversion) rather than a public GitHub account, I must consider is licencing my own code using GPL. This GNU (general public licence) would obey copyright laws, so should others ever attempt to use my work commercially, they must cite me. This is a huge issue for many creators and indeed the Computer Science industry.

My project is based almost in it's entirety on security. Analysing, discovering and producing flaws in an android system to then produce a detection software as a result. Malicious users may be interested in the coding I produce in order to steal private sensitive data from other's phones. There is also a possibility that they could use my code, to give them ideas on how to implement their own malicious program / Applications. Thus my programs will be in a private repository that no malicious user may be able to breach and use my code. This is in order to create a controlled environment and reduce the risk of malicious use elsewhere.

Bibliography

- [1] Wade Gasior and Li Wang. *Exploring Covert Channel in Android Platform*. International Conference on Cyber Security, Alexandria, VA, USA, 2012.
- [2] Ahmed Al-Haiqi, Mahamod Ismail, and Rosdiadee Nordin. *A New Sensors-Based Covert Channel on Android*. The Scientific World Journal, Volume 2014, Article ID 969628, 14 pages: <http://dx.doi.org/10.1155/2014/969628>.
- [3] Swarup Chandra, Zhiqiang Lin, Ashish Kundu, and Latifur Khan. *Towards a Systematic Study of the Covert Channel Attacks in Smartphones*. International Conference on Security and Privacy in Communication Systems, 2014.
- [4] Roman Schlegel, Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala, Apu Kapadia, XiaoFeng Wang. *Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones*. NDSS Vol 11, pp. 1-17, 2011.
- [5] Steven Gianvecchio and Haining Wang. *An Entropy-Based Approach to Detecting Covert Timing Channels*. CCS '07 Proceedings of the 14th ACM conference on Computer and communications security, Alexandria, Virginia, USA, 2007.
- [6] P.A. Porras and R.A. Kemmerer. *Covert flow trees: a technique for identifying and analyzing covert storage channels*. Proceedings. 1991 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, USA, 1991.
- [7] Matt Bishop. *Introduction to computer security*. Addison-Wesley Professional, 2004.
- [8] Jorge Blasco and Thomas M. Chen. *Automated generation of colluding apps for experimental research*. Journal of Computer Virology and Hacking Techniques, 2017
- [9] Claudio Marforio, Hubert Ritzdorf, Aurlien Francillon and Srdjan Capkun. *Analysis of the Communication between Colluding Applications on Modern Smartphones*. In Proceedings of the 28th Annual Computer Security Applications Conference, pp. 51-60. ACM, 2012.
- [10] Richard A. Kemmerer. *Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels*. ACM Transaction on Computer Systems, Vol. 1, No. 3, August 1983, Pages 256-277.
- [11] Audio channel Exploitation App. <https://github.com/unclealf/covertChannelApp>. Accessed 2017
- [12] EPSRC funded project *App Collusion Detection (ACID)* December 2014.