An intentionally insecure Android Image against Inter Component Communication attacks

Alexandru Dascălu
965337

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Bachelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

April 23, 2021

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

I would like to dedicate this work to the Hypnotoad.
All glory to the Hypnotoad.

Abstract

In your abstract you should aim to summarize the core contributions of your work in the context of the problem domain. Start by outlining the domain and the problems posed within it. Discuss how the methods you focus on approach the relevant problems. You should end your abstract by concretely stating the tangible outputs and deliverables you have created in order to complete your work on this document, and whether those outputs represent and improvement or alternative approach to existing methods.

Your abstract should be a couple or so paragraphs long, and roughly approximate the order and flow you then use for structuring the main document. If a viewer has read your abstract then they should already understand at a high level what it is you have created and delivered, and whether it is better than or comparable to existing methods. If your project is driven by a research hypothesis then the reader should know what that is at a high level from this section. Reading on, little should surprise the viewer.

For paper submission of your thesis you should physically sign your name on each of the above declaration statements and date them in black ink. For digital submissions you should sign and date them digitally using a touch or stylus input if available. There are pieces of software that allow you to write directly on PDF documents, or alternatively you can bring a signature into your document as a figure with a transparent or white background. If you do not have a stylus input / tablet like device you should ask your supervisor, as many in the department do their grading / work on digital tablets.

Acknowledgements

First of all, I would like to express my deep gratitude to my mother for all the emotional support and encouragement that she has given me over the past academic year and for being there when I needed her.

Furthermore, I would like to thank Dr Phillip James for being a great supervisor, being easy to contact and guiding me towards realistic project goals.

I am thankful towards my colleagues Constantinos Loizou and Avi Varma for their support and suggestions, and want to thank Avi for sharing with me things he learnt when doing his project a year ago. Moreover, I want to thank them for participating in an informal user study to help me evaluate my project.

I sincerely thank Phillippos Pantekis and Bessam Helal for advising me during the process of picking my project in May 2020.

Finally, I would like to thank Dr Mihaela Cojocaru for helping me to improve my mental health during this difficult time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Current State of Mobile Cyber Security

Ever since the release of the iPhone in 2007, smart phones and other mobile devices have played an increasingly larger role in our lives. We are using mobile devices for education, social interaction, commerce, mobile banking, entertainment, work, and more. Moreover, 52.6 percent of global web traffic came from mobile devices in 2019, up from 31.6 per cent in 2015 [5].

Meanwhile, cyber security of mobile apps is underwhelming. A 2018 study done by cyber security consultancy Positive Technologies analysed 17 mobile apps, 8 for Android and 9 for iOS [6]. It found that only 11% of the apps had an acceptable level of security, with 56% having a below average level of security, and 43 percent of Android apps had critical vulnerabilities. A more worrying piece of information is that the average Android client app in the study contained 3.7 vulnerabilities, of which 1.1 were of high risk [6].

Against the lacklustre security of mobile apps, attackers are becoming more relentless and are using more elaborate techniques [1]. In figure 1.1 you can see how the monthly number of reported cyber attacks has increased since 2017.

Meanwhile, security of Android apps is not improving. A study conducted on over 400,000 APKs released between 2010 and 2016 belonging to over 28,000 apps analyzed how Android app security evolved over time [7]. It discovered that critical vulnerabilities of various types have always been common in apps. Moreover, app updates generally do not improve security, and can even re-introduce previously fixed vulnerabilities. This is the largest and most comprehensive study of its time to date. Another study on the Google Play ecosystem found that

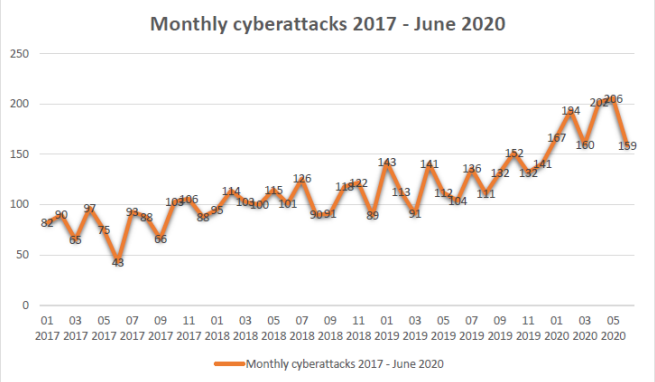


Figure 1.1: Increasing number of monthly cyber attacks since 2017. Data is compiled from [1], [2], [3] and [4].

the proportion of malware has dropped significantly in 2017 compared to 2014 [8], though it studied fewer APKs than [7].

The remainder of Chapter 1 outlines the document structure, the motivations behind undertaking this project, the aims and objectives of the project and the main contributions of this piece of work.

1.2 Motivations

I chose to explore Inter Component Communication vulnerabilities on Android in this project for reasons based on the current state of mobile application security and how this vulnerability category is touched upon in related work.

1.2.1 ICC-related vulnerabilities in current mobile apps

One of the categories of vulnerabilities that the authors of [6] analysed was Inter Component Communication related vulnerabilities. Android components are the logical building blocks of an app, and components can communicate with other components, which can belong to the same app or to another app on the same device.

In the study conducted in [6], 29 percent of client apps used insecure inter-process communication, and the average Android app had 1.1 ICC-related vulnerabilities [6]. Insecure inter-component communication is a high-risk vulnerability, and it is more dangerous than insecure storage, inadequate brute force protection or other common vulnerabilities [6], and therefore deserves attention from developers and cyber security specialists.

For Phil : Should the first and second paragraph of this subsection be moved to the Current State of Cyber Security section?

Moreover, while the overwhelming majority of vulnerabilities of other types are introduced in apps through the use of third-party libraries and rarely through developer code, ICC vulnerabilities differ in this regard. According to [7], between a third and a half of ICC vulnerabilities come from code written by the developer directly. Therefore, education regarding this type of vulnerability addressed to the average developer would greatly help with minimizing the occurrence of these faults.

1.2.2 ICC-related vulnerabilities in related work

Vulnerabilities and cyber attacks concerning communication between Android components are not showcased in detail or intuitively in real word projects similar to mine. These projects either only cover some types of ICC related vulnerabilities, none at all, or are not easy to use, do not have good guidance, or do not show a realistic scenario.

Damn Insecure and Vulnerable App has challenges 9 and 10 that show examples of hijacking intents meant for a vulnerable activity [9]. DIVA also contains challenge 11 where a malicious user can send intents to extract data from an unprotected content provider.

Purposefully Insecure and Vulnerable Android Application, also called PIVAA from here on, has three challenges on components that are insecurely exported and thus can be sent malicious intents. The challenges show this vulnerability in a broadcast receiver, a service and a content provider, respectively.

Another intentionally insecure Android app used for educational purposes is InsecureBankv2 [10]. Similarly to PIVAA, it contains vulnerable components that can be started by a malicious app which injects a specially crafted intent, though it has a vulnerable activity, broadcast receiver and content provider and no service, unlike PIVAA [11]. Furthermore, another ICC related vulnerability it contains is intent sniffing, which allows a malware to intercept intents sent by components of the vulnerable app.

Damn Vulnerable Web Application, an intentionally insecure web application with vulnerabilities such SQL injection or Cross Site Scripting, does not cover any Inter Process vulnerabilities. TryHackMe.com is an interactive website for learning cyber security and penetration testing. Although easy to use and very useful, it has no dedicated activities on IPC.

None of the projects mentioned in this subsection cover all types of ICC vulnerabilities. Furthermore, DIVA, PIVAA and InsecureBankv2 only offer one application which has various

components for each individual vulnerability they implement, they do not offer standalone apps for each vulnerability. More importantly, these applications are meant to be used together with other tools such as ADB or Dozer on a computer in order to perform each cyber attack. Thus, these projects do not offer real examples of malicious apps that can attack vulnerable apps. Another area in which these projects are lacking in is the educational instructions provided in the app. They provide little to no instructions, and only some hints as to how to solve each vulnerability, they do not show detailed explanations and code examples of the vulnerability and how to fix it. In addition, they do not provide much interactivity, and have to be used together with a computer. Finally, some of the apps contain major UI bugs, such as the About project activity in PIVAA.

Damn Vulnerable Web Application, or DVWA, had multiple security levels for each vulnerability, where in each one the vulnerable program uses code that is increasingly secure. It tells the user what each level does, how secure it is, and shows relevant source code for it. This is a very useful feature that is not shared by the three insecure Android app projects mentioned earlier in this subsection.

1.2.3 Motivation Conclusion

We have shown how vulnerabilities related to Inter Component Communication in Android are widespread and that they offer a dangerous attack surface through which applications can be hacked. Furthermore, current projects that implement intentionally insecure Android apps do not cover the domain of Inter Component Communication well and have a lot of room for improvement.

I have therefore chosen to create an educational tool that explores vulnerabilities related to insecure Inter Component Communication in Android that provides clear guidance, is interactive and combines the best parts from each of the related projects that I have looked into.

For Phil: Is the above paragraph good? Is this whole Motivation section too long? Do I add personal motivations?

1.3 Project Aims and objectives

In this section, we will discuss the causes of the precarious security of apps and why intentionally insecure programs are a good solution. Following that, I will present the aim of this project, and the actions or objectives that will be undertakes to achieve said aims.

1.3.1 Overview

Why is it that Android app security is often neglected? The reality is that software developers are not always trained in cybersecurity, may lack costly automatic analysis tools, or do not have the time to design and implement secure software due to tight deadlines [12]. These issues can be caused by upper management not being aware of the importance of cyber security. Therefore, there needs to be more awareness of cyber security, and of vulnerabilities related to ICC in particular, in the software development industry, especially among developers, penetration testers and project managers.

To solve these issues, more education regarding the importance of cyber security in Android is needed. An Android image that is intentionally insecure could be designed as an educational tool. Such software could be used to highlight how android malware works and what it can do. It could teach people how to exploit vulnerabilities and how to fix them.

Such a product would be aimed at developers, penetration testers and senior technical management. For developers, it would make them aware of ICC vulnerabilities, what attackers can do and how to fix these issues. Penetration testers would be able to practice how to hack vulnerable apps using inter component communication. For the upper technical management of businesses, it would raise awareness of cyber security and the need to train staff and allocate proper resources.

1.3.2 Project Aims and Objectives

The aim of this project is to create an educational tool that can be used by penetration testers, developers, and tech savvy people alike to learn about Android mobile application vulnerabilities related to Inter Component Communication.

The primary objectives of the project, that will make up the minimal viable product, are:

- Develop a home application through which the user can access all educational material and can interactively learn about each vulnerability and how to exploit it and fix it.
- Develop a series of apps that act as either a vulnerable app or a malicious app. Each challenge has two apps, one that is vulnerable and one that is the malware that exploits the former.
- Make sure that the experience of using the product is cohesive and pleasant.

1.4 Contributions

The main contributions of this work can be seen as follows:

- **An intentionally insecure Android image focused on Inter Component Communication**

  During this project, we developed a series of Android apps that showcase a wide range of vulnerabilities related to Inter Component Communication, which are not the main focus of existing related work.

- **An educational tool for Android vulnerabilities that does not require the use of ADB shell**

  The finished product can be used to learn about ICC vulnerabilities on one's smartphone, and does not require one to use the ADB shell in order to attack the vulnerable app.

- **An intentionally insecure Android image which includes malware**

  Unlike existing projects of this type, our project has developed real malware apps that attack other apps in the image, and does not rely on the user to use ADB Shell to attack the app.

- **An intentionally insecure Android image with multiple security levels for each vulnerability**

  DVWA has multiple security levels for each vulnerability, as mentioned in subsection 1.2.2. No other related work on Android except ours implements this, to our knowledge.

Chapter 2

Background

This chapter will cover the necessary technical background regarding Inter Component Communication. It will first give an overview of important concepts of the Android Operating System. Following that, we will explain what components and permissions are in Android, and then go on to discuss how components communicate between each other. Subsequently, we explore the various types of vulnerabilities related to Inter Component Communication.

2.1 Basics of the Android Operating System

In Android, each application is by default assigned a unique user ID known only by the OS. Each app runs in its own process by default, and each process runs in its own virtual machine [13]. Consequently, apps are generally separated from each other, which enhances the security of the system.

Throughout the continuous development of Android, the API is modified to introduce new features and improve security or performance. Therefore, in order to identify each incremental version of the API, a unique integer is assigned to each version or level. Over the years, there have been changes to the API that improved software security, and therefore some vulnerabilities are harder or impossible to exploit in current API levels.

The manifest of an Android app is an XML file that gives the system information about the app's structure, capabilities and needs. All Android app components, except broadcast receivers, need to be declared in the manifest file, and for each component you can define permission requirements and the capabilities of the component [13]. Moreover, the developer can say in the manifest file what hardware or software system features the app uses. For

example, an app would not be installed on a device if its manifest said it required a microphone and the mobile device did not posses microphone hardware. Components will be explained in detail later in section 2.2, and permissions in section 2.3.

2.2 Android Components

Android mobile apps are made up of logical building blocks called components. A component is an entity which allows the user or the operating system to access the application [13]. Therefore, a component does not necessarily correlate with other computing concepts such as processes or threads. When any component of an app needs to be run, the system starts a process for that app. There are four types of components in Android: activities, services, broadcast receivers and content providers. We will detail these in the rest of section 2.2

2.2.1 Activities

Activities represent the individual app UI screens through which a user interacts with the app. For example, a news aggregator application might have an activity for viewing a list of news articles. Activities are used by the operating system to keep track of what the user sees on screen, what information they are interested in, and the information of minimized apps that might be needed later [13].

2.2.2 Services

Services are components used for running long-term operations in the background. Importantly, a service does not represent a separate process or thread, but an interface for the system to let the app work in the background [14]. A service does not have a user interface itself. Examples of the usage of services include VPN apps that maintain a VPN connection in the background.

There are three types of services: foreground services, which perform tasks that are noticeable to the user and must display a notification, background services, which do things that are not noticeable to the user, and bound services, which act as servers responding to requests made by client components [15].

2.2.3 Broadcast Receivers

Broadcast receivers are components used to receive system wide broadcasts. These broadcasts are messages sent by the operating system or by other apps. Applications can react to various events by using broadcast receivers. For example, the system can send a broadcast to let apps know that the device's battery is low. An app can use a receiver to listen for an event even when the app is not running. Receivers do not have a user interface but can display notifications. In addition, it is worth noting that they do not have to be declared in the manifest but can be created programmatically as well.

There are three types of broadcasts, two of which are relevant to our project:

- Normal broadcasts – These are sent to all receivers at the same time, and each receiver can react independently of other receivers.

- Ordered broadcasts – These are sent to receivers one at a time. Unlike with a normal broadcast, the receiver currently processing the broadcast can change what information the broadcast contains, and can even cancel the broadcast, so that it will not be sent to further receivers [16]. Broadcast receivers can be registered with a certain priority for getting broadcasts.

2.2.4 Content Providers

Content providers are interfaces through which apps can access data stored in persistent storage such as a remote server, an SQL database or local file storage. A provider can be used by components of the same app or by components of other apps. Therefore, they are used by the system to manage access to shared data. Content providers can restrict access to the data to apps with certain permissions and give temporary access to certain files only [13].

2.3 Permissions

Android follows the principle of least privilege. This principle is enforced through a system of permissions, meaning that an application can only access sensitive data, system features or components of other applications if it possesses the necessary requirements [17]. For instance, an application needs the correct permission to access the user's contacts.

Moreover, developers can protect a component of an app with permission requirements by adding an `android:permission` tag in the manifest file. Only components in apps that have that permission will be able to send an intent to the protected component.

2.3.1 Types of permissions

There are four types of permissions, three of which are relevant to this project:

- Normal permissions – Permissions for unimportant resources, such as the permission to set the time zone [17]. They are granted automatically at install time.
- Dangerous permissions – They are for important resources such as private user information, or that can affect the state of the system or of other apps. The user needs to give explicit permission in the app to utilise these resources.
- Signature permissions – These are special permissions designed for use among a group of apps created by the same developer. An app is automatically granted a signature permission at install time only if it is signed by the same certificate as the app that defined the permission. The certificate can be self-signed by the developer. Its purpose is to identify the author of an app [18].

2.3.2 Custom permissions

Applications can declare their own permissions. These can be used to restrict access to components of an application, or protect broadcasts of that app. This is done by declaring a permission in the manifest file of the app, as you can see in listing 2.1.

```
1 <permission
2   android:name="uk.ac.swansea.alexandru.icc_education.permission.BANKING_ACTIVITY"
3   android:label="@string/lab _bankingActivity"
4   android:description="@string/desc_bankingActivity"
5   android:protectionLevel="dangerous" />
```

Listing 2.1: A declaration of a custom permission in an Android manifest.

2.4 Inter Component Communication

So far, we have seen that each Android application runs in its own sandbox, and by default can not see what other applications are doing. Sometimes, we need the system to communicate

with the apps, and applications can enrich the users experience by collaborating. Moreover, an application component can be used by other apps to provide extra functionality. For example, a browser lets you select which social media or messaging app to use for sharing a link.

Intents are a class in the Android API that are used as messages for communication between application components. More specifically, intents are used to start new activities, start and stop services, bind or unbind a component to a service, and they also represent the broadcasts that are sent to receivers. Intents can carry data in the form of a URI, as well as other data in the form of key value pairs [19].

2.4.1 Exported Components

By default, app components are not accessible to outside apps through intents. However, a component can be exported and thus receive intents from other applications. To export a component, you can set the `<exported>` tag in a component in the app's manifest to true. However, if the component has an intent filter defined in the manifest, the component will become automatically exported unless the exported tag is explicitly set to false. Intent Filters will be fully explained in section 3.5.3.

Further complicating component exportation is that developers can configure an application to use the same Android User ID as other applications created by them. This means these apps can run in the same process, and they can access each other's components regardless of the exported tag or the presence of intent filters. This is shown very well in Table 1 in [20].

2.4.2 Explicit Intents

Explicit intents directly specify the application that should receive the intent and handle it. This is done by setting either the package name of the receiving application, or the full name of a component of said app [21]. Explicit intents can contain other information, such as data or the intended action to be performed, as you can see in Listing 2.2.

Using an explicit intent means that only the targeted app or component can receive the intent. Explicit intents are usually used for communication between components of the same app, such as when one activity starts another when the user clicks a button. That being said, explicit intents can be used to start components of other apps as well. As explained in subsection 2.4.1, an app component must be exported so that other apps can send explicit intents to it.

```
val noPaymentUri : Uri = Uri.parse("santander_pay://uk.ac.swansea.dascalu.dvmicc.
    santander/pay")
val intent = Intent(this, LogInActivity::class.java)
intent.setDataAndType(noPaymentUri, "text/plain")
startActivity(intent)
```

Listing 2.2: Kotlin code to make an explicit intent, add data to it and start an activity with it

2.4.3 Implicit Intents

Unlike explicit intents, implicit intents do not directly specify what application or component it should be sent to. Instead, the Android system decides who to send it to based on the information in the intent and what other components have declared they can handle.

A component defines what intents it can handle by specifying Intent Filters in the manifest file, with an example in Listing 2.3. An Intent Filter defines the type of intents an application can handle. A filter can say what actions the component can perform, what intent categories it accepts, the MIME data types it accepts or the kind of URI resources it can handle. A component may declare multiple Intent Filters, and it is recommended that this is done for each task the component can do [21].

```
<activity android:name=".LogInActivity">
    <intent-filter>
        <action android:name="uk.ac.swansea.dascalu.dvmicc.santander.intent.action
            .LOGIN" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.HOME"/>
        <data android:mimeType="text/plain"
            android:scheme="santander_pay"
            android:host="uk.ac.swansea.dascalu.dvmicc.santander"/>
    </intent-filter>
</activity>
```

Listing 2.3: Declaration of an intent filter that the intent in Listing 2.4 will match.

When an implicit intent is sent, like you can see in Listing 2.4, the Android System compares its attributes against all intent filters of all components. For the intent to be matched with a filter, three tests are performed: the Action test, the Category test, and the Data test [21]. In order to pass the Action test, the Intent's action must be amongst the actions of the filter. It passes the Category Test if all of its categories are found in the filter's declaration, and the Data Test is passed if the data URI or MIME type of the intent matches one of the data elements in

the filter. If the component has multiple filters, the intent only needs to match one of them for it to be passed to the component.

```
1 val noPaymentUri : Uri = Uri.parse("santander_pay://uk.ac.swansea.dascalu.dvmicc.
      santander/pay")
2 val intent = Intent("uk.ac.swansea.dascalu.dvmicc.santander.intent.action.LOGIN")
3 intent.addCategory(Intent.CATEGORY_HOME)
4 intent.setDataAndType(noPaymentUri, "text/plain")
5 startActivity(intent)
```

Listing 2.4: Kotlin code to make an implicit intent, add data to it and start an activity with it

If only one intent filter matches the implicit intent, the operating system will start that filter's component automatically. However, if there are multiple matches, a dialog will be displayed to the user so they can manually select the component to handle the intent.

For example, if there are multiple browsers installed on a device, and within an app the user clicks on a web link, they will then see an Android dialog letting them choose what browser to use to open that page. This is because the parent app sent an implicit intent, and all browsers had filters that matched with the intent.

2.5 Inter Component Communication Vulnerabilities and Attacks

In this section, we will explain how the way components communicate using Intents can be exploited by attackers, and what developers can do to fix these vulnerabilities.

Most of the vulnerabilities that will be explored in this project happen due to the misuse of implicit intents or intent filters. Because implicit intents do not directly state what component they target, it is possible that an intent is delivered to a malicious app. Moreover, an attacker could create malicious intents that could launch other components in ways that could compromise the victim app.

2.5.1 Intent Hijack

The Android documentation recommends that explicit intents are used for intra-app communication, and implicit intents for inter-application communication [21]. However, developers sometimes use implicit intents to start a component within the same app. An attacker can make an app with an intent filter designed to match with said implicit intents, which can direct the intent to the malicious application. The process in which an intent is matched against a filter was described in section 2.4.3. When receiving an intent, a component can read all of its data.

Therefore, even if the implicit intent is meant for external use, if the developer puts sensitive information in it, that data could be intercepted.

In general, vulnerabilities against this class of attacks are fixed by using explicit intents instead of implicit intents in order to send broadcasts, start activities or services or grant access to a content provider [22]. Another way to mitigate these attacks is to not put sensitive information in implicit intents.

2.5.2 Intent Spoofing

While Intent Hijack attacks work by accidentally activating a malicious component when an intent is intercepted, Intent Spoofing attacks happen when a victim component is unexpectedly activated by an attacking component using an Intent. Often, this attack targets components that are not meant to be accessible outside of their apps, but because they have an intent filter and the `<exported>` tag is not set, they are exported automatically. Because exported components are accessible to other apps, the attacker can create an explicit intent targeting it and does not need to worry about having to match intent filters. The developers are usually unaware of this.

This attack is dangerous because components often extract information from the intents they receive. An Intent Spoofing attack could insert malicious information into the victim component. The attacker could send invalid information to crash the victim's app (a DoS attack), corrupt the user's data, or inject commands to retrieve data. Often, the launch of an activity, service or broadcast receiver leads a change in the application's state [22], even if the victim component takes no input from the intent. And since activities and services can return information to the component that activated them, they could leak sensitive information. This attack is often done against components that were not meant to be accessible to other apps, and these components are thus less likely to check the input data of the intent or make sure they do not return private data.

Developers can secure their application from this class of attacks in a number of ways [22]. First, they should not declare intent filters for internal components, and they should use explicit intents to launch them. Secondly, if they need to use intent filters for internal components, they should explicitly declare them as not exported in the manifest. Thirdly, they can protect components with permissions of various types. Fourthly, state-changing operations should not be done in exported components. And finally, developers should make sure that exported components do not return information that should be private.

Chapter 3

Implementation

This chapter will explain my implementation for this project. It will first give an overview of the architecture of the software product. After that, we will present each one of the challenges that have been implemented so far.

3.1 Architecture

The central part of the software implementation consists of the Damn Vulnerable Mobile - Inter Component Communication app. This Android app contains all of the educational material needed to understand Android ICC and complete each challenge, encourages the user to learn interactively, and it provides access to each challenge and to settings to control the challenge or the learning experience.

The rest of the project consists of a series of Android apps that play the role of either a vulnerable app or a malware. These apps are made to appear to be authentic apps with real-world functionalities, though many of them are nowhere close to full-feature app. Each challenge consists of an authentic scenario of a malware app attacking one of the vulnerable apps through one particular ICC vulnerability.

The DVM-ICC app communicates with the challenge apps through the use of files in order to apply various settings to control the behaviour of the apps. These settings are explained in subsection 3.2.2. Moreover, each pair of vulnerable and malicious apps communicate between each other as a result of the cyber attack, through the medium of Intents. The manner in which this communication takes place will be explained in detail for each challenge in section 3.3.

3.2 DVM-ICC Application

In this section, I will explore the features of the DVM-ICC Android app. While doing so, I will explain the intended workflow that a user needs to follow and how app provides an educational experience. The follow description applies to the Beginner mode. The app has three different operation modes, which will be fully explained in subsection 3.2.7.

3.2.1 Home Menu

The first screen the user sees when starting the app has 3 subpages: Introduction, Info and Challenges, as you can see in figure 3.1. The Introduction page gives an introduction of the project and explains the intended workflow for using the product. The Info page provides all of the necessary technical background to understand ICC vulnerabilities and attacks. It covers what components and permissions are, how components communicate between each other and the major types of ICC based attacks: Intent Hijacking and Intent Spoofing. The Challenges page lets the user select what challenge they want to start, as shown in figure 3.1.

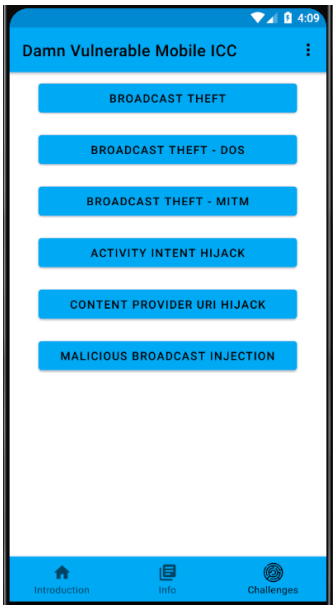


Figure 3.1: The first menu in the DVM-ICC app.

3.2.2 Challenge Settings

When clicking on a challenge, the user is taken to a menu where they can change the settings that define how the challenge will be undertaken. Here, you can set the operation mode for the challenge. Moreover, the user can enable or disable the malware of the selected challenge. If disabled, the malicious app will not perform any cyber-attack and therefore it will not interfere with any vulnerable app or the rest of the system.

The most important setting on this screen is the security level of the vulnerable app. Inspired from DVWA, as described in subsection 1.2.2, these levels define how secure the vulnerable app is against attacks from the malware. Each challenge's vulnerable app has between 2 and 5 security levels, with each successive level using more secure code. This culminates with the Impossible level, where the vulnerability is fixed and the malware can not perform the attack. When the malware is enabled, it will overcome the defences of the current security level,

except for the Impossible level. The number of security levels and their meaning depends on the challenge. In general, each security level means that a component or broadcast is protected with a more powerful permission, that components are no longer exported or that an intent is now sent explicitly.

These settings are written to a text file that is located in the DVM-ICC's app specific directory on the device's external storage, from where it can be accessed by any other app, including the other apps that are part of this project. The security level setting will dynamically change the behaviour of the vulnerable and malicious apps, and the user does not need to restart them, with some exceptions. The challenge settings can be changed at any time while doing a challenge.

3.2.3 Identifying the malware and the vulnerable app

Once the user applies the settings described in subsection 3.2.2, the application starts the Challenge activity, which you can see in figure 3.2. The first task of every challenge is to identify the pair of vulnerable and malicious apps for that challenge from amongst all of the apps that are part of the project, except the DVM-ICC app itself.

The Info page gives a technical description of one of the two major attack categories, Intent Hijacking or Intent Spoofing, followed by a detailed description of the specific cyber attack related to this challenge. After reading and understanding this information, the user needs to go the Manifests page, where they can view for each app the contents of its manifest and all snippets of code that sends intents, as shown in figure 3.2. Based on the technical background the user was given in the Info page and in the Home Menu from subsection 3.2.1, they need to look through the code in the Manifests page and try to identify the vulnerable app and the malware for this challenge.

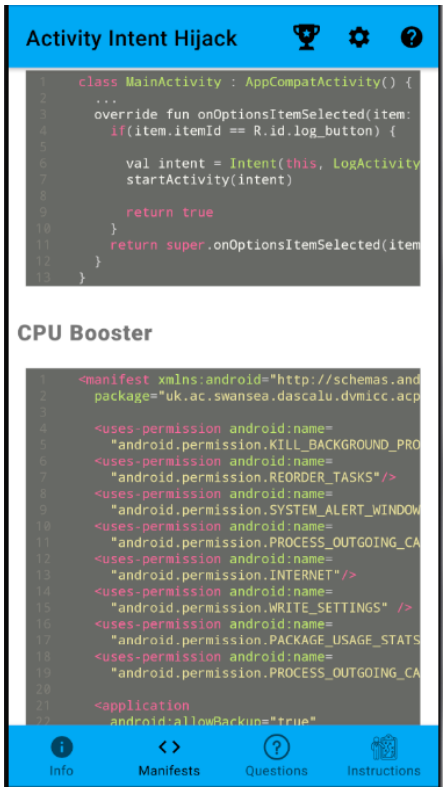


Figure 3.2: The Manifests page in the Challenge activity.

The user should be able to detect vulnerable apps by looking for intents being sent implic-

itly or for exported components, for example. Malwares can be usually identified by the use of intent filters matching intents that are only sent by a vulnerable app, or by sending an explicit intent to another app that is part of this project. The features that identify the vulnerable app and the malware depend on the selected challenge.

The Questions page contains text boxes where the user can type in their answers and be told if they have correctly identified the apps.

3.2.4 Performing the cyber-attack

Now that the user knows the correct pair of apps for this challenge, the next task of the challenge is to use these apps to witness the cyber-attack take place.

Once you have correctly answered the questions regarding the identity of the malware and vulnerable app, the Instructions page will display detailed instructions for using the apps such that the specific cyber-attack for that challenge will take place. This page can be accessed from the activity shown in figure 3.2.

In general, the user is asked to take a look at the two apps to familiarise themselves with them and what they can do. Then, the user should make sure that the security level is set to low. After that, they should use the vulnerable app and malware according to the provided instructions in order to trigger the attack and observe it.

Following that, the user needs to go the malware and look in its Log, which is accessed from its app bar. If the attack was successful, the malware should have written in its log the flag associated with this specific challenge and security level, along with any relevant data it managed to steal from the vulnerable app. These would be hidden amongst the output of Android Log.

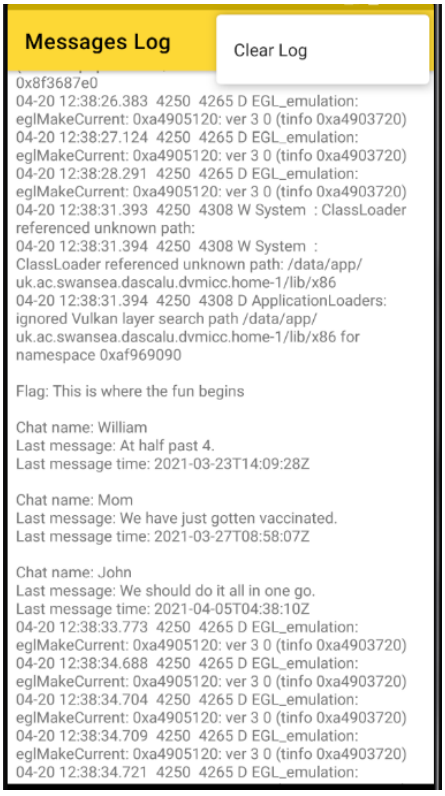


Figure 3.3: Log of the malware for Content Provider URI Hijack after it successfully attacked the vulnerable app.

You can see an example of this in figure 3.3, where the messages from the Whatsapp vulnerable app have been stolen by the SMS Messages malware. The user should then copy

the flag and submit it to the flag question for the current security level in the Questions page in the DVM-ICC app, to prove they have completed one level of the challenge.

At this point, the user should clear the log of the malware, to see the effects of the next attack more easily, and then change the security level to the next value. They should repeat the process described in this subsection for every security level of that challenge.

Through performing the task described in this section, the user will witness an example of a cyber attack happening in an authentic scenario, with a real malicious app exploiting an insecure app.

3.2.5 Security Levels Explanation

While completing the task described in subsection 3.2.4, the user can click on the Help button that you can see at the top right in figure 3.2 to view an explanation of the security levels for the current challenge. This view is hidden until the user has identified the pair of apps for the current challenge, as explained in subsection 3.2.3.

For each level, it will explain how it works and how it is more secure than the previous level. Following this explanation is a view of the full manifest of the vulnerable app, as it would be used for that security level. Finally, this view might show code snippets that introduce a vulnerability through the way they send an intent.

You can see part of the Security Levels explanation for the Broadcast Theft challenge in figure 3.4, as an example. The top code snippet shows the code used to send a broadcast in the News Aggregator app in the medium security level. Since this code send the broadcast implicitly, it is insecure.

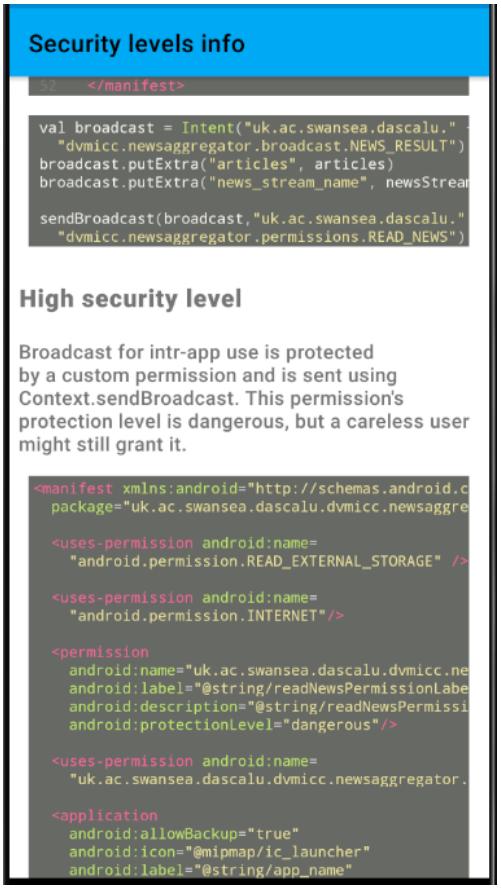


Figure 3.4: Part of the explanation of the security levels for the Broadcast Theft Challenge.

This activity teaches the user how and where the vulnerability is introduced and what steps a programmer can take to secure the vulnerable

app, complete with code examples from the app itself. Some solutions fail to completely remove the vulnerability, and this activity teaches the developer why they are inadequate. The explanation of the Impossible level shows how to completely remove the attack surface.

By looking at these explanations while performing the task described in subsection 3.2.4, the user will be able to see how the cyber attack of that challenge can still succeed despite the programmer implementing some security defences.

3.2.6 Challenge Conclusion

Once the user has performed the cyber attack and has submitted the correct flag for every security level, they have completed that challenge. As a reward, they can now click on the trophy button on the app bar of the Challenge activity, which you can see in figure 3.2.

This will open an activity where the user can read the conclusion to the current challenge. This text will give a summary of the whole challenge, going through how the vulnerability is introduced, how the attacker can create a malware that takes advantage of it, and how to fix the vulnerability. Moreover, the conclusion will comment on the authenticity of the scenario presented in the challenge, covering how or why the vulnerability was introduced in the first place and how likely it is for the scenario might to be encountered in the real world. The purpose of this conclusion is condense all of the information that the user should take away from this challenge.

3.2.7 Operation Modes

As mentioned at the beginning of section 3.2, the DVM-ICC application supports three modes. The Beginner mode has already been described in subsections 3.2.1–6. It is meant for less experienced users, who need to given detailed explanations in order to understand the ICC vulnerabilities of each challenge.

The Expert Mode works in the same way as the Beginner mode, but it hides a lot of information from the user to increase the difficulty. In the Challenge activity from figure 3.2 on page 17, the Info page will be hidden, and the Security Levels Explanation activity from figure 3.4 will only contain code snippets, without any of the explanations.

The Make your own Malware mode disables the malware app of the current challenge in order to allow the user to create their own malware and attack the vulnerable app without interference from the included malware. In this mode, the user only has access to the Info page described in subsection 3.2.3 and the Security Levels activity described in subsection 3.2.5.

3.3 Challenges

Section 3.2 covered the general workflow for using the software of this project, which applies to challenges. In this section, we will focus on the vulnerability explored by each challenge and the pair of malware and vulnerable app that demonstrate it. The first five challenges involve variations of the Intent Hijack attack presented in subsection 2.5.1, while the sixth is a variation of the Intent Spoofing attack explained in subsection 2.5.2.

For all challenges presented here, the description in the Implementation subsection of each challenge describes the interaction between malware and vulnerable app when the security level is set to Low. The other security levels are presented in the Security Levels subsection of each challenge.

3.3.1 Broadcast Theft

When a broadcast is sent, the sender does not receive any indication of what components have received that broadcast. A malicious app can register a broadcast receiver with as many intent filters as possible to listen to many public broadcasts [22]. The malware can read the data in the broadcasts without the user knowing it, and could therefore be used as spyware.

Moreover, if an implicit intent is used to make a broadcast meant for an app's internal use, that broadcast could be sent to any receiver on the device with a matching filter. An attacker could reverse engineer an application, see the intent filter that matches the broadcast, then add an identical filter to a broadcast receiver in their malware.

3.3.1.1 Implementation

The vulnerable app for this challenge is News Aggregator, an application that looks for online news articles from various sources and shows them to the user. When its background service has downloaded the news articles, it will send a broadcast with the JSON data, which will be received by a broadcast receiver in the app. This receiver will process the data and update the UI. The broadcast is sent as an implicit intent, and any receiver with an intent filter that includes the action string of the broadcast will get it.

In a real world application, developers might want to use an implicit broadcast for intra-app communication in order to make the application more loosely coupled, and thus more modular and flexible. Another reason might be that you want other apps made by you to know when the News Aggregator has gotten some articles.

The malware for this challenge is the Call Logger app, which displays the user's call history. In it, the attacker has added a broadcast receiver with an intent filter that is identical to that of News Aggregator's receiver. If Call Logger and News Aggregator are installed on the same device, the broadcast sent by News Aggregator will be received by both its receiver and the malware's receiver. Call Logger can therefore see what news articles a user is looking at and what they might be interested in.

3.3.1.2 Security Levels

When sending a broadcast, the developer can specify the permission that a receiver's app needs to have in order to be able to get the broadcast. This can be used to guard against Broadcast Theft. A developer can declare their own custom permission, as seen in subsection 2.3.2, which must be obtained by receivers to be able to get broadcasts sent by the developer's app.

If the developer does not set the protection level of the permission, it will default to "normal". This means that the malware will get it automatically at install time if it asks for that permission in its manifest. This is what happens in the Medium security level. In the High security level, the protection level of News Aggregator's custom permissions is set to "dangerous", meaning the malware needs to ask the user to grant that permission. This is quite secure, but a careless user may still grant it. The most secure protection level for the custom permission is "signature", but if the private key of the certificate is not stored securely, it can be stolen and be used to sign the malware, which would be able to get the permission when it is installed and receive the broadcast. This is demonstrated in the Very High security level. The protection levels of permissions were fully explained in subsection 2.3.1.

Because permissions with a normal or signature permissions are granted at install time, they can not be changed dynamically. Therefore, for the Medium and Very High security levels, the user needs to use the Call Logger 2 and 3 app, respectively. These are identical to Call Logger, except they ask in their manifest for the relevant permission for their security level, and they have a different app icon.

In the Impossible security level, the broadcast is sent as an explicit intent, as the News Aggregator service sets the name of the app whose components can receive the broadcast. This completely removes the vulnerability [22]. However, it also means that the broadcast can not be received by other apps, which the developer might want, depending on the situation.

3.3.2 Broadcast Theft - Denial of Service

The challenge presented in subsection 3.3.1 focused on normal broadcasts. Ordered broadcast are sent to receivers one at a time, as explained in subsection 2.2.3. The use of ordered implicit broadcasts can not only enabled eavesdropping as described in subsection 3.3.1, but denial of service attacks as well [22]. A malicious app can register a broadcast receiver with a very high priority to ensure it is the first to receive it. It can extract the data from the broadcast, and then abort the broadcast, ensuring the intended receivers do not get it and thus perform a Denial of Service attack.

3.3.2.1 Implementation

The Call Redirect app is the vulnerable app for this challenge. This app redirects phone calls by automatically adding a customizable country code prefix before the number is dialed. It achieves this by having a broadcast receiver which listens for broadcasts with the action NEW_OUTGOING_CALL, sent by the OS whenever a call is being made [19]. This broadcast is ordered, and it contains the dialed phone number. The Call Redirect reads this number, adds the prefix and then sets the new number to the broadcast.

The malware for this challenge is the CPU Booster app, which pretends to improve the performance of the device, but does not actually do anything. It will also have a receiver that listens for broadcasts with the NEW_OUTGOING_CALL action. This receiver will have the maximum possible priority declared in the malware's manifest. Consequently, it will get the broadcast before Call Redirect. It reads the dialed in phone number, then cancels the broadcast. The legitimate app's receiver will not receive it and the phone call will be canceled.

3.3.2.2 Security Levels

This challenge only has Low, High and Impossible security levels.

Since the broadcast is sent by the system and not by the app, we can not protect it with a permission. The developer of the Redirect app can set their receiver's priority to the maximum possible value, like the malware does. This way, the receiver of this app will hopefully be called before the malware's receiver, change the number in the broadcast, and then cancel it so the malware does not receive it. The number will still be dialed because when the broadcast was sent by the OS, a final receiver was specified, which gets the broadcast even if it is canceled. This is what the vulnerable app does in the High security Level.

According to the Android documentation, receivers with the same priority are executed in an arbitrary order [16]. From our experience, on devices on Android 7.1 or 8, receivers with the same priority are called in alphabetical order of their app package names. For this reason, the package name of the malware is acpu_booster, to maximise the chance its receiver will get called first, before Call Redirect's receiver can cancel the broadcast. Consequently, the attack can still happen in the High security level. Moreover, other legitimate receivers do not get the ordered broadcast, as Call Redirect canceled it before they had the chance to receive it.

Therefore, the best way to fix this vulnerability rests with the user. CPU Booster can not receive the broadcast if it does not have permission to access phone calls from the user, and it has no obvious reason it would need it. In the Impossible level, the user should see that the app is suspicious and they should uninstall it or remove its permission.

3.3.3 Broadcast Theft - Man In the Middle

The vulnerability explored by this challenge is the same as the for the Broadcast Theft - DOS challenge in subsection 3.3.2. The difference is that the cyber atack has a different outcome, it is a Man in the Middle attack instead of Denial of Service.

3.3.3.1 Implementation

This challenge's vulnerable app is Call Redirect yet again. The malware for this challenge is the Battery Booster app, which pretends to improve the battery life of the device, and yet again does not actually do anything. It will also have a receiver that listens for broadcasts with the NEW_OUTGOING_CALL action, in the same way CPU Booster did. It will get the broadcast before Call Redirect, read the phone number and replace it with a different phone number, then send the permission to the next receiver. The legitimate receiver will receive the broadcast with the injected number and the country code to it. This phone number will then be dialed. Battery booster can be used to stealthily re-direct phone calls to malicious phone numbers.

3.3.3.2 Security Levels

The Security levels for this challenge are Low, High and Impossible, and they function in the same way as the security levels of Broadcast Theft - DOS, described in subsection 3.3.2. The only differences are that the malware does not cancel the broadcast and instead changes its data, and that the package name of the malware is abattery_booster. The latter is to make sure the malware's receiver gets the broadcast first.

3.3.4 Activity Intent Hijack

In this type of attack, a hacker takes advantage of the use implicit intents so that a malicious activity is launched instead of the intended one, by declaring a matching intent filter for their malicious activity in the manifest.

In a sophisticated version of this attack, the attacker can use phishing to steal user credentials [22]. If an app uses an implicit intent to start its Login activity, a hacker can use an identical intent filter for a fake identical looking Login activity in their malware. The implicit intent will match both the fake Log In activity and the legitimate one, and thus prompt the user to choose between them. The attacker can give a confusing name to their malware, or make the icons of the two apps similar, to make the user choose the malware. The unaware user then types in their credentials on the fake Login screen, which are then be sent to the hacker.

3.3.4.1 Implementation

The vulnerable app for this challenge is Santander, a banking app. After they log in, the user can view details of fictitious bank accounts and can make fictional payments to another account, which will decrease the balance one of the accounts displayed in the app.

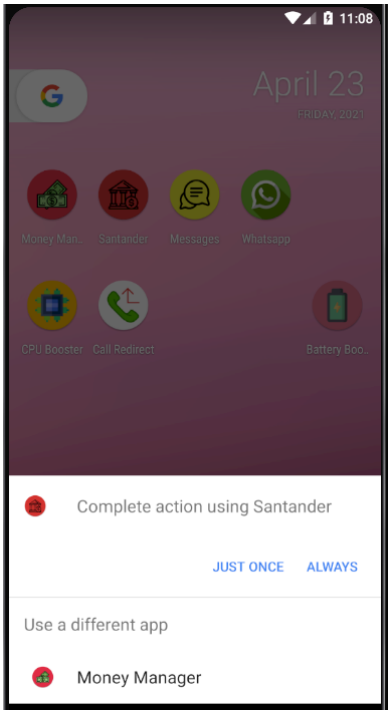


Figure 3.5: Choosing between activity of vulnerable app and malware.

Santander has an intent filter for its Login activity so that the user can click on special link that starts the bank app to make a payment. The link contains details of the recipient of the payment, such that these details are already filled in when the app launches and therefore making the payment easier. An implicit intent with this link can be sent from another app. It will match Santander's intent filter and start the Login activity to authenticate the user, before going to the payment screen.

The attacker decompiled the Santander app to see the code for the Login activity and its intent filter. They made an activity in the Money Manager malware that looks identical to the one in Santander, and added to it an intent filter. Money Manager is an app for tracking and categorising one's income and expenses, but it does not have full functionality.

When you open the app, the user sees a Loading screen, which sends an implicit intent to start the Login activity when it ends. There will be two filters that match with this intent, and therefore the user will be asked to pick between Santander and Money Manager, as you can see in figure 3.5. Both are financial apps, and Money Manager has a similar icon and colour theme, and therefore a careless user might select the malware. Money Manager's Login activity looks identical to Santander's, so the user types in their credentials, which the malware can then send to the attacker. To make it as if nothing suspicious happened, it will say the credentials are incorrect and then start the legitimate Login activity with an explicit intent.

3.3.4.2 Security Levels

When sending intents to start an activity or service, you cannot protect them with a permission like you can when you send a broadcast and as we did in subsection 3.3.1.2. Therefore, in the Impossible security level, the solution to fix the vulnerability is for the Santander Loading activity to use an explicit intent to start the Login activity. By doing this, the intent can not be sent to the malware. Explicit intents should be used for communication between components of the same app.

However, when the user clicks on a link in another app to make a payment, the malware's intent filter will still match that implicit intent and the attack could succeed. For this case, the solution is to use App Links such that when a user clicks on a web link, they are either taken to a web page in a browser or they are taken to an activity in your app [23]. These are not implemented in the Santander app at the current moment.

3.3.5 Content Provider URI Hijacking

In section 2.4 we saw that intents can transmit data using a URI. These URIs can point to data stored using a Content Provider. When declaring a Content Provider in the manifest file, the developer can set the `android:grantUriPermissions` tag of the `<provider>` element to true. This means that the content provider can allow temporary access to data linked to in a URI in an intent for a component in another app that received that intent. Therefore, external apps can access the content, even if the provider is not exported and thus not normally accessible by external apps.

In order to do this, the intent that transmits the URI link must have the `FLAG_GRANT_READ_URI_PERMISSION` or `FLAG_GRANT_WRITE_URI_PERMISSION` flags added to it. If this

intent is an implicit intent and is intercepted by a malicious component, then that component can read the data and perhaps modify it.

3.3.5.1 Implementation

The vulnerable app for this challenge is Whatsapp, an internet messaging app named after the popular real world app. This app lets you view some hard-coded chats, it does not have real functionality. These chats are accessed by the app through its content provider, that is not exported, but that has `android:grantUriPermissions` set to true in its manifest. The user can click on a button to go to another activity where they can select what chats they want to delete. When they click the delete button, the activity will send an implicit intent that contains the selection of chats, a URI to where chats are stored in the provider, and flags for temporary read and write permissions. This intent is received by a service of Whatsapp, which is responsible for accessing the provider and deleting the messages.

Because this intent is implicit, it can be intercepted by the malware, which is the SMS Messages app in this case. This app only reads SMS messages on the device, it can not send new messages. The malware contains a service with an intent filter identical to that of the Whatsapp service. The malicious service will get the intent, and because the intent has temporary access flags, the malicious service can query the Whatsapp content provider and read messages. You can see these messages in the malware's log in figure 3.3 on page 18. Because Whatsapp's provider is not exported, if the malware queried the provider without having that intent, an exception would be thrown.

3.3.5.2 Security Levels

This challenge only has Low, High and Impossible security levels. In the High level, the provider is not exported and has not enabled temporary permissions overall. However, temporary permissions are enabled only for the path where chats are stored. Although the rest of the data in the provider can not be accessed by other apps, the data in this path can still be accessed by any component with a temporary permission. The temporary permission is still transmitted through the implicit intent sent by the Delete activity in Whatsapp.

In the Impossible level, the intent sent by the Delete activity, which contains URIs to messages and has temporary permission flags, is sent as an explicit intents. The Messages malware can not get these intents and therefore can not get permission to access the provider. If it tries to query the provider, an exception will be thrown.

Chapter 4

Finding and citing resources

The university has subscriptions to a vast number of major academic journals spanning a wide range of subject areas. By accessing the internet from a university network connection (Eduroam or Ethernet), the paywalls of many journals will simply vanish without any need for login credentials.

4.1 Tunnel your internet connection via the university internet

When you are working from outside of the university then connecting to an on campus machine via remote desktop (RemoteDesktopProtocol, TeamViewer, ect) or via port forwarding (ssh, ssh tunnel, ect) can allow you to access papers that would otherwise be behind a paywall.

If you do not have individual access to a machine that is exposed for ssh on the university network you can always use the computers in Linux Lab CF204[1] for the purpose of setting up an ssh port tunnel to proxy your internet through. These machines have fixed IPv4 addresses and respond to ssh using your student account credentials. While in use your internet will be routed[2] to the university and then out to the internet, granting you transparent access to journals without a paywall.

[1] One caveat of using computer lab machines for remote tunnelling is that a environmentally conscious student who has worked late in the computer lab might choose to switch off the machine you were using...

[2] Painfully slowly.

4.2 Practice your Google Fu

The internet is big [24]. Knowing how to phrase a question to a search engine is therefore an invaluable skill. If the request is simple enough, even a poorly structured query will likely return usable results. For more difficult to find resources you can leverage the language of the search engine to gather relevant papers and resources for your research more efficiently.

`https://www.gwern.net/Search`

"Internet Search Tips" [25] provides an excellent review of methods and tips for scouring the internet for hard to find resources. You will also be less likely to get caught behind journal paywalls when working remotely without a tunnel as your queries can be made to look for raw pdfs that are often released by the authors directly.

4.3 Organizing your citations in BibTeX

BibTeX is a language for specifying resource citations. Every time you access and read an academic paper, take code from an online repository, or source the media such as images from existing works you should create a BibTeX entry in a file that you keep throughout your research. Software such as Mendeley [26] can help automate the process of building your BibTeX library of citations.

```
@INPROCEEDINGS{kaj86,
    author    = {Kajiya, James T.},
    title     = {The Rendering Equation},
    booktitle = {Proceedings of the 13th Annual Conference on Computer Graphics
        and Interactive Techniques},
    year      = {1986},
    series    = {SIGGRAPH '86},
    pages     = {143--150},
    address   = {New York, NY, USA},
    publisher = {ACM},
    isbn      = {0-89791-196-2},
    numpages  = {8},
    acmid     = {15902}
}
```

Listing 4.1: An example BibTeX entry for an academic paper published in conference proceedings [27].

The BibTeX code listing above (listing 4.1) shows an example of how to cite an academic paper, in this case one of the central papers in Computer Graphics research. The key **kaj86** is

an arbitrary name chosen as a meaningful identifier for the resource. In the document text we can call on this resource as an inline citation using the LaTeX command `\cite{kaj86}` which produces [27] at the location it is called. As long as a citation has been used at least once somewhere within the document then a formatted full citation will be created in the bibliography at the end of the document with the same citation number that is shown inline.

It is considerably easier to be disciplined in methodically taking note of the resources you access and make use of as you access them, than it is to try and hunt them all down again at the time you need to write about them in your document. Invest time in being organized and consistent up front and it will be easier when you come to write up.

4.4 Properly using and formatting citations within the text

Usually you would not put the URL of the resource you are citing directly in the text like is done previously in section 4.2. The citation for the resource [25] is sufficient to reference it within the text given that full details of its location are then kept neatly within the bibliography at the end of the document.

In normal usage the purpose of a citation is not to direct the reader away from your thesis, but to justify and back up assertions you are making about the state of the domain. If a reader questions your assertions then they can follow the rabbit hole of papers which will likely also make and justify assertions with even earlier papers from the literature.

In the above case the intention is for the reader of this template to actually go to that resource and read what it has to say directly. The link is therefore shown clearly within the main text to indicate that the reader should visit it. This as opposed to wanting the reader to purely acknowledge that the facts which are within the resource legitimize the points made in this document, in which case a simple inline citation is the best way to back up your assertions. Section 5.3.7 specifically touches on the best practice for how to cite images which you are importing from existing work.

Chapter 5

Typesetting your thesis

This document is intended as both a LaTeX thesis template and as a tutorial on structuring and typesetting your thesis in the LaTeX programming language.

The following are some powerful online resources for learning about LaTeX:

- **Overleaf Documentation for LaTeX**

  Overleaf [28] is an online browser-based LaTeX IDE which stores your document in the cloud and provides live recompilation as you type. The documentation on Overleaf's website has a good knowledge base of examples for how to typeset things cleanly and simply in LaTeX code.

  See: `https://www.overleaf.com/learn`

- **TeX StackExchange, the StackOverflow site dedicated to TeX questions**

  TeX StackExchange [29] is sub-community of the StackOverflow network dedicated to questions about the TeX family of typesetting tools including LaTeX, BibTeX and others. A vast majority of the time it is unlikely that the question or issue you are facing is one that has not been encountered before, and this site more than likely to be able to point you in the correct direction.

  See: `https://tex.stackexchange.com`

5.1 Referencing items within this document

In section 4.3 we saw examples of how to typeset citations for resources we had stored in an external BibTeX file. However, often we would like to accurately refer to the location of a resource or region of text stored somewhere else within this document[1]. To do this we need to annotate our LaTeX code with `\label{key}` statements which will take on the numeric (or otherwise formatted) identifier for the current chapter, section, figure, table, equation, ect where they are directly defined. To insert an inline reference to the label you can use the `\ref{key}` command which works similarly to the `\cite{key}` used for external references. In the event we chose to reorder or add additional content to the document, which would change the section numbering, the document will still compile to a pdf with the correct references inserted for each `\ref{key}` command.

5.2 Equations

Typesetting equations is one of the things that LaTeX does best. It has packages for different fonts and symbols for many different mathematical notations. However, to person learning how to typeset in LaTeX for the first time it can be a daunting and unwieldy user experience. Almost all LaTeX packages have documentation available in pdf format online, and documentation for packages specifically relating to fonts and symbols usually have tables enumerating the names and codes for all of the fonts symbols, organized by intended usage.

5.2.1 Inline equations

Small equations like $x = 0$ can be written directly within the text by using LaTeX's maths mode shorthand controlled by dollar signs `$ math mode $`. As long as it is not becoming cumbersome to the reader, equations such as $\mathbb{P}(A \cap B) = \mathbb{P}(B \cap A)$ are quite neatly displayed in this fashion.

[1] Like at the beginning of the last sentence when we referred to section 4.3.

5.2.2 Block equations

For long equations it is best to provide a break in the main text of the document and format the equation using a `\begin{equation}...\end{equation}` environment.

$$|a| = \left\| \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \right\| = \sqrt{a_0^2 + a_1^2 + \ldots + a_n^2} \tag{5.1}$$

Equation 5.1 demonstrates formatting a larger equation and uses an `\begin{array}...\end{array}` environment to structure a column vector of sub-equations. Block equations should be located at a relevant point directly as they are being referred to in the text. When referred to from other locations in the document you should use the `\ref{key}` command to insert the correct equation number.

5.2.2.1 Aligning multi-line block equations

When equations become even larger they may need cross over multiple new lines. When this happens it is desirable to align relevant parts of the equation on each line to one another for aesthetic reasons and to help imply structure to the reader.

$$\begin{aligned} \mathscr{L}_o(x, \omega_o, \lambda, t) &= \mathscr{L}_e(x, \omega_o, \lambda, t) \\ &\quad + \int_\Omega f(x, \omega_i, \omega_o, \lambda, t)\, \mathscr{L}_i(x, \omega_i, \lambda, t)\, (\omega_i \bullet n)\, d\omega_i \\ &\quad \text{where} \quad \mathscr{L}_i(x, \omega_i, \lambda, t) = \mathscr{L}_o\left(x', -\omega_i, \lambda, t\right) \end{aligned} \tag{5.2}$$

Equation 5.2, known as Kajiya's Rendering Equation [27] demonstrates the use of the `\begin{split}...\end{split}` environment which uses a single un-escaped & symbol placed on each line of the equations LaTeX code to indicate where each line should be co-aligned. In this example the &'s were placed on the =, +, and w (in where) characters.

5.2.3 A masochistic approach to learning to typeset mathematics in LaTeX

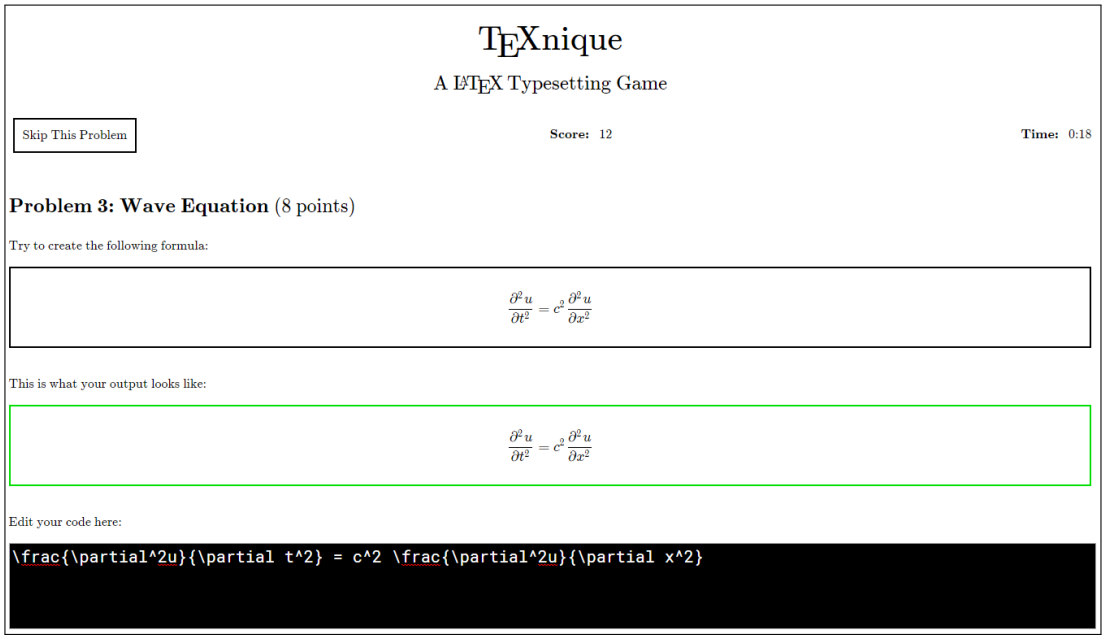


Figure 5.1: TeXnique, a game about typesetting equations [30]. (Top) The game presents you with a rendered equation, (Bottom) the task is to enter LaTeX code that produces the same rendered equation. The green border on the lower rendering indicates it is a valid solution.

TeXnique [30] is web-browser based game for practising how to typeset equations in LaTeX. The game will present you with a rendered equation and your task is to type LaTeX code into the box below it such that your code produces the same (or closely matching / pixel equivalent) rendered equation. Figure 5.1 shows the game during play, the bottom rendered equation is bordered in green to indicate it is a valid match with the target.

`https://texnique.xyz`

This is one of the more painful parts of typesetting a document, so it really takes a special kind of sadism to come up with such a game. Least to say, graduate students and researchers can be an odd bunch, and when we found this it was surprisingly addictive to compete over.

5.3 Figures

In this template figures are numbered starting with the current chapter number followed by a figure number that resets to 1 each new chapter. As you can see below, the first figure is

labelled Figure 5.2 because we are in Chapter 5.

Figures in LaTeX are defined using a `\begin{figure}...\end{figure}` environment and often immediately begin rendering in centre aligned mode by calling `\centering`. Listing 5.1 below shows the LaTeX code used to typeset figure 5.2. Figures 5.3 and 5.4 are defined similarly and make additional use of the `\subfloat` command to position multiple images within a single figure environment, each with their own automatically incremented labels and individual captions.

```
\begin{figure}[H]
  % [H] means put the figure HERE, directly when you input this code.
  \centering

  % We set the width of the figure based on the width of one line
  % of text on the page. The value can be tuned to any value in
  % [0.0, 1.0] to scale the image while maintaining its aspect ratio.
  \includegraphics[width=1.0\linewidth]{./graphics/dragon.png}

  % Caption is defined with a short and long version. The short
  % version is shown in the List of Figures section, and the long
  % version is used directly with the figure.
  \caption[Short caption.]{Long caption and citation \cite{whittle15_dragons}.}

  % For figures, \label should be defined after the caption to ensure
  % proper figure numbering.
  \label{fig:dragon}
\end{figure}
```

Listing 5.1: An example LaTeX excerpt demonstrating how to typeset figure 5.2 with a simple caption.

5.3.1 Consistent presentation throughout the document

Figures work best in a document when you use a consistent style for formatting and captioning them and make sure that figures always actively support the content of the main text.

5.3.2 Justified use of space in the document

All figures must be referred to directly in the main text of the document and discussed with meaningful and in depth critical analysis. If you don't need to use the figure to leverage and support your discussion then it is just taking up space and padding out the document. For example, you can use a command like `\ref{fig:dragon}` to automatically get the figure number for Figure 5.2.

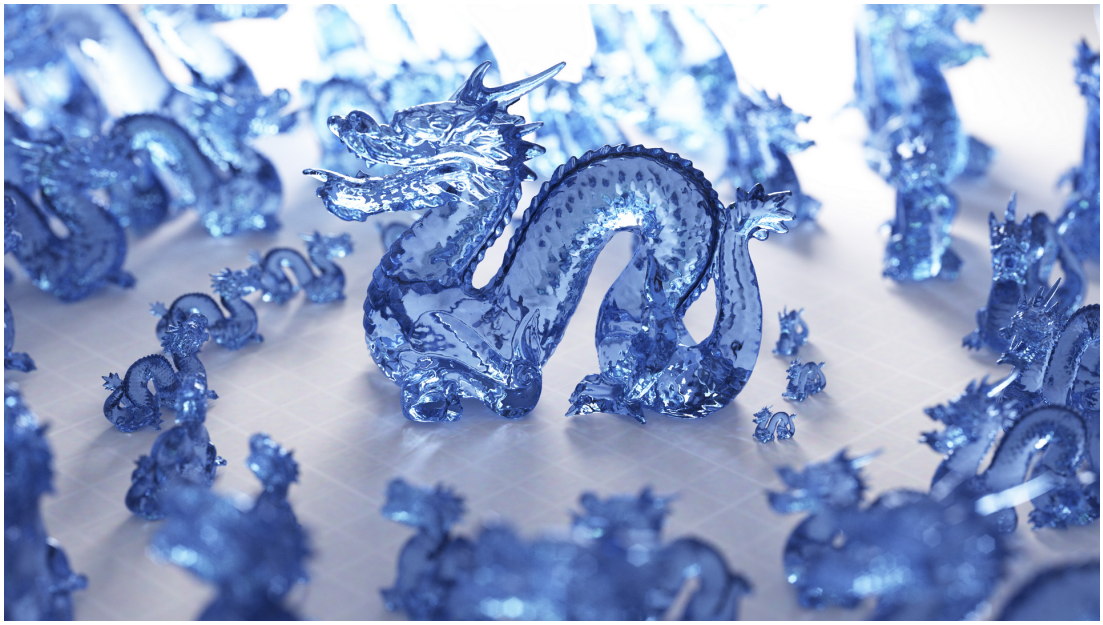


Figure 5.2: A good caption should be sufficient enough to put the figure in context even if the reader has randomly flicked to the current page and looked only at the figure in isolation. All figures should also be referred to directly within the main text of your document. You can use the LaTeX **\ref{key}** command to insert the correct figure number when you refer to it in the main text. By the very logic of this caption, this is a very poor caption because we still don't know why on earth is there an picture of glass dragons here. Image of glass dragons rendered using Path Tracing [31].

5.3.3 Placement that supports and enhances the flow of the document

All figures shown in your document should be displayed in relevant locations, ideally just after that have been alluded to in the main text. Although there are many times where it is best to force a figure to the top or bottom of a nearby page.

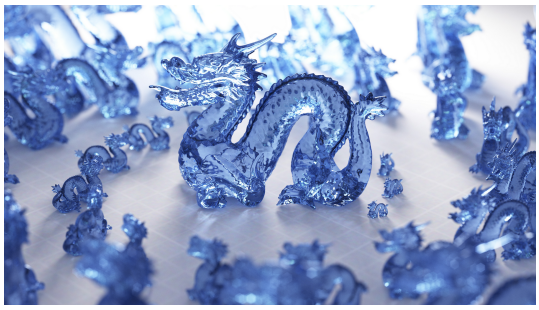
5.3.4 Avoid directly importing other peoples images

You should avoid using other peoples figures whenever possible, and instead create your own figures for visualizing the specific methods and data you are working with in a way directly relevant to your project.

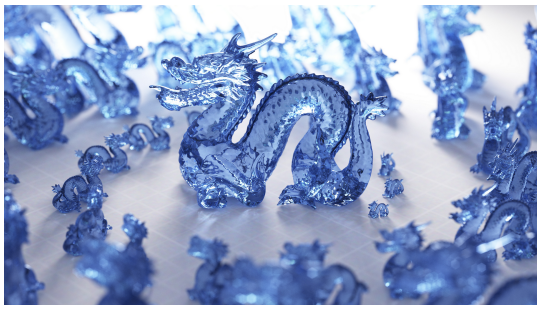
5.3.5 Format sub-figures in LaTeX, not in the image itself

Construct sub-figures from multiple image files in LaTeX not in the image file itself. This allows you to tweak the positioning and layout without having to modify the images. It also

allows for automatic formatting and numbering of captions and sub-captions. Figures 5.3 and 5.4 show examples of side-by-side and quad layouts respectively.

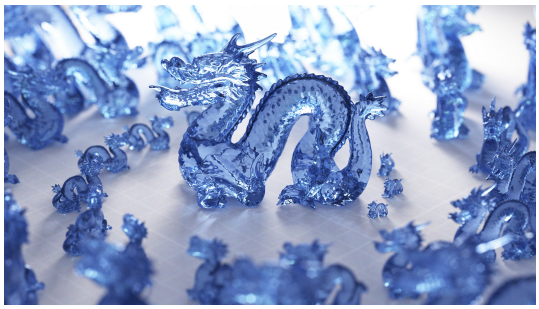


A. Left image sub-caption.

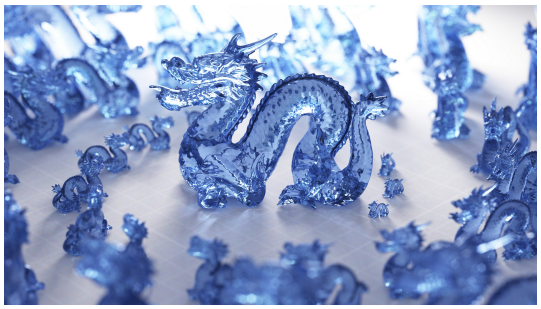


B. Right image sub-caption.

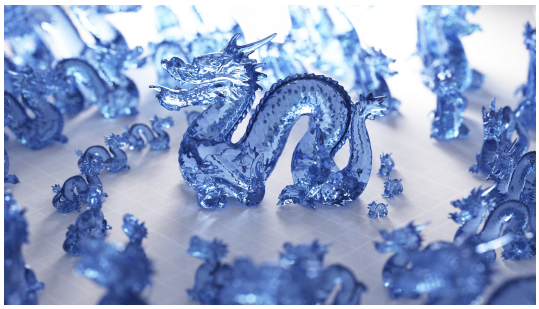
Figure 5.3: Construct sub-figures from multiple image files in LaTeX not in the image file itself. This allows you to tweak the positioning and layout without having to modify the images. It also allows for automatic formatting and numbering of captions and sub-captions. Image of glass dragons rendered using Path Tracing [31].



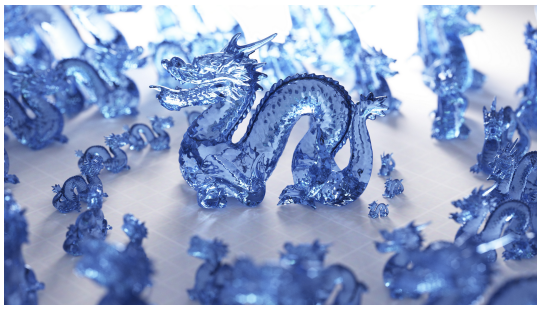
A. Top-Left image sub-caption.



B. Top-Right image sub-caption.



C. Bottom-Left image sub-caption.



D. Bottom-Right image sub-caption.

Figure 5.4: A demonstration of a 2x2 sub-figure layout. Between A-B and C-D we use tilde symbols and between B-C we use a new line. Image of glass dragons rendered using Path Tracing [31].

5.3.6 Robust captions that can stand in isolation

Figures need to be captioned such that they can be viewed in isolation and still be meaningful to the viewer. There will likely be some duplication of information that is written in the main text, but this is intended.

5.3.7 Proper attribution and citation of images

If an image does not belong to you it **must** be cited directly in the figure caption. **It is not correct to put a URL in the figure caption directly.** A URL in isolation is not an accurate or reliable way of directing a future reader to the exact content you are referencing. Instead make a new entry in your `citations.bib` file and then reference that citation in the caption using the `\cite{key}` command. Figures 5.2, 5.3, and 5.4 each include a statement in the caption stating "Image of glass dragons rendered using Path Tracing [31].". When adding the BibTeX entry, try to find the proper information about the original author and source document to strengthen the citation in case the URL changes.

5.4 Code Listings

Code listings should be formatted in the same style as figures and inline equations. It is important to use a monospace font so that characters line up vertically. Syntax highlighting is also extremely important for effectively displaying complicated code segments. To format inline code listings you can use the `\lstinline|the_code|` command[2].

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello world.\n");
  return 0;
}
```

Listing 5.2: An implementation of an important algorithm from our work.

In LaTeX the "Listings" package can be used to properly format code and provide basic syntax highlighting, line numbering, and captioning of embedded code excerpts. Listing 5.3 shows examples of how to properly format code using the listings package.

[2] So meta.

```
% The lstinline command can be used to insert monospace formatted code directly
% inline within the documents main text. You can optionally specify a programming
% language to enable syntax highlighting.
\lstinline|the_code|
\lstinline[language={the_language}]|the_code|

% The lstinputlisting command is used to insert an external file containing
% code into the document formatted in the same manner as a figure or table.
% All stand alone listings should have a label and caption. You can optionally
% specify a programming language to enable syntax highlighting.
\lstinputlisting[label={lst:my_label_name}, caption={The caption.}]{the_file}
\lstinputlisting[language={the_language}, label={lst:the_label}, caption={The
    caption.}]{the_file}

% An example showing how Listing 3.1 is formatted in LaTeX code.
% The C code is stored in its own file as C code, allowing it to be modified
% and prepared separately using a dedicated code IDE to ensure correctness and
% proper formatting.
\lstinputlisting[language={c}, label={lst:c_hello_world}, caption={An
    implementation of an important algorithm from our work.}]{./listings/
    hello_world.c}
```

Listing 5.3: Examples of methods for typesetting code listings within a LaTeX document.

5.5 Tables

Tables are also quite predictably captioned and formatted the same way. It is important to decide on a style for how you will organize your data and apply that style consistently for all of your tables. Table 5.1 shows one possible way of styling your data but is by no means the only way of doing so neatly. Consistency is the key.

Table 5.1: An example of a table formatted with caption.

Some	Relevant	Fields	From	Your	Data
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2

Chapter 6

Conclusions and Future Work

In this document we have demonstrated the use of a LaTeX thesis template which can produce a professional looking academic document.

6.1 Contributions

The main contributions of this work can be summarized as follows:

- **A LaTeX thesis template**

  Modify this document by adding additional top level content chapters. These descriptions should take a more retrospective tone as you include summary of performance or viability.

- **A typesetting guide of useful primitive elements**

  Use the building blocks within this template to typeset each part of your document. Aim to use simple and reusable elements to keep your document neat and consistently styled throughout.

- **A review of how to find and cite external resources**

  We review techniques and resources for finding and properly citing resources from the prior academic literature and from online resources.

6.2 Future Work

Future editions of this template may include additional references to Futurama.

Bibliography

[1] P. Technologies. (2018) Cybersecurity threatscape 2017 trends and forecasts. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Cybersecurity-threatscape-2017-eng.pdf

[2] ——. (2019) Cybersecurity threatscape 2018 trends and forecasts. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Cybersecurity-threatscape-2018-eng.pdf

[3] ——. (2020) Cybersecurity threatscape 2019. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/cybersecurity-threatscape-2019-eng.pdf

[4] ——. (2020) Cybersecurity threatscape q2 2020. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/cybersecurity-threatscape-2020-q2-eng.pdf

[5] Statista.com. (2020) Percentage of mobile device website traffic worldwide from 1st quarter 2015 to 2nd quarter 2020. [Online]. Available: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/

[6] P. Technologies. (2019) Vulnerabilities and threats in mobile applications. [Online]. Available: https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Mobile-Application-Vulnerabilities-and-Threats-2019-eng.pdf

[7] J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Understanding the evolution of android app vulnerabilities," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 212–230, 2019.

[8] H. Wang, H. Li, and Y. Guo, "Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play," in *The World Wide Web Conference*, ser. WWW '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1988–1999. [Online]. Available: https://doi.org/10.1145/3308558.3313611

[9] Srinivas. (2016) Cracking damn insecure and vulnerable app (diva) – part 4. [Online]. Available: https://resources.infosecinstitute.com/topic/cracking-damn-insecure-and-vulnerable-app-diva-part-4/

[10] dineshshetty. (2019) Android-insecurebankv2. [Online]. Available: https://github.com/dineshshetty/Android-InsecureBankv2

[11] Hacktivities. (2020) Android insecurebankv2 walkthrough: Part 2. [Online]. Available: https://infosecwriteups.com/android-insecurebankv2-walkthrough-part-2-429b4ab4a60f

[12] J. Umawing. (2018) Why bad coding habits die hard—and 7 ways to kill them. [Online]. Available: https://blog.malwarebytes.com/101/2018/05/bad-code-wont-die-7-ways-kill/

[13] Developer.android.com. (2021) Application fundamentals. [Online]. Available: https://developer.android.com/guide/components/fundamentals

[14] ——. (2021) What is a service. [Online]. Available: https://developer.android.com/reference/android/app/Service#what-is-a-service

[15] ——. (2021) Services overview. [Online]. Available: https://developer.android.com/guide/components/services

[16] ——. (2021) Broadcasts overview. [Online]. Available: https://developer.android.com/guide/components/broadcasts

[17] ——. (2021) Permissions on android. [Online]. Available: https://developer.android.com/guide/topics/permissions/overview)

[18] ——. (2021) Define a custom app permission. [Online]. Available: https://developer.android.com/guide/topics/permissions/defining

[19] ——. (2021) Intent. [Online]. Available: https://developer.android.com/reference/android/content/Intent

[20] D. Sbîrlea, M. G. Burke, S. Guarnieri, M. Pistoia, and V. Sarkar, "Automatic detection of inter-application permission leaks in android applications," *IBM Journal of Research and Development*, vol. 57, no. 6, pp. 10:1–10:12, 2013.

[21] Developer.android.com. (2021) Intents and intent filters. [Online]. Available: https://developer.android.com/guide/components/intents-filters

[22] E. Chin, A. P. Felt, K. Greenwood, and D. Wagner, "Analyzing inter-application communication in android," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 239–252. [Online]. Available: https://doi.org/10.1145/1999995.2000018

[23] Developer.android.com. (2021) Handling android app links. [Online]. Available: https://developer.android.com/training/app-links

[24] Internet Live Stats. (2020). [Online]. Available: https://www.internetlivestats.com

[25] G. Branwen. (2020) Internet search tips. [Online]. Available: https://www.gwern.net/Search

[26] RELX Group. (2019) Mendeley. [Online]. Available: https://www.mendeley.com

[27] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: ACM, 1986, pp. 143–150.

[28] Overleaf. (2020) Overleaf documentation. [Online]. Available: https://www.overleaf.com/learn

[29] Stack Overflow. (2008) Tex stackexchange. [Online]. Available: https://tex.stackexchange.com

[30] A. Ravikumar. (2019) Texnique. [Online]. Available: https://texnique.xyz

[31] J. Whittle. (2015) Path traced glass dragons.

Appendix A

Implementation of a Relevant Algorithm

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello world.\n");
  return 0;
}
```

Listing A.1: An implementation of an important algorithm from our work.

Appendix B

Supplementary Data

The results of large ablative studies can often take up a lot of space, even with neat visualization and formatting. Consider putting full results in an appendix chapter and showing excerpts of interesting results in your chapters with detailed analysis. You can use labels and references to refer the reader here for the full data.