

# Security Analysis in SMS-based Applications

Individual Project  
CM3203 – 40 Credits



School of Computer Science and Informatics  
Cardiff University 2022

---

Author: Finn Milliner

Supervisor: Neetesh Saxena

Moderator: Bailin Deng

## Contents

|   |    |
|---|----|
| 1. Introduction.....                        | 5  |
| 1.1. Project Scope .....                    | 6  |
| 1.2. Project Approach .....                 | 7  |
| 1.3. Aims & Objectives .....                | 7  |
| 2. Background.....                          | 8  |
| 2.1. SMS as a Service .....                 | 8  |
| 2.2. Structural Security Issues .....       | 9  |
| 2.3. SMS Misuse.....                        | 10 |
| 2.4. Existing Solutions & Constraints ..... | 10 |
| 2.5. Existing Analysis Works .....          | 12 |
| 3. Approach .....                           | 14 |
| 3.1. Development Methodology .....          | 14 |
| 3.2. Project Timeline .....                 | 14 |
| 3.3. Research Approach.....                 | 15 |
| 3.4. Data Collection .....                  | 16 |
| 4. Implementation .....                     | 18 |
| 4.1. Tools Used.....                        | 19 |
| 4.1.1. Platform .....                       | 19 |
| 4.1.2. Python.....                          | 19 |
| 4.2. Risk Assessment .....                  | 21 |
| 4.3. Flowchart and UML.....                 | 22 |
| 4.4. SMS Analysis Tool .....                | 24 |
| 4.4.1. GUI .....                            | 25 |
| 4.4.2. Key Functions.....                   | 26 |
| 4.4.2.1. Get_file Function.....             | 26 |
| 5. Results and Evaluation .....             | 34 |
| 5.1. Exhibits.....                          | 34 |
| 5.1.1. Collected Dataset .....              | 35 |
| 5.1.2. Analysis Tool.....                   | 35 |
| 5.2. Test Cases.....                        | 37 |
| 5.2.1. Complex Cases .....                  | 38 |
| 5.3. Results tables .....                   | 41 |
| 5.3.1. Dataset Analysis Results .....       | 41 |

|  |    |
|--|----|
| 5.3.2. URL Analysis Results .....  | 42 |
| 5.4. Evaluation .....  | 43 |
| 5.4.1. OTP Analysis .....  | 43 |
| 5.4.2. URL Analysis .....  | 44 |
| 5.4.3. Keyword Analysis .....  | 44 |
| 5.4.4. URL Health Check Analysis .....   | 45 |
| 5.5. Recommendations based on results .....  | 45 |
| 6. Future Work .....   | 46 |
| 7. Conclusions .....   | 47 |
| 7.1. Aims & objectives Reflection .....  | 47 |
| 7.1.1 Objective 1: Outline Key Security Flaw or Exploits in SMS Services and Applications .....  | 47 |
| 7.1.2 Objective 2: Identify and Explain Both the Advantages and Limitations of SMS as a Service, Providing Quantitative Data to Show Relevance to Security ..... | 48 |
| 7.1.3 Objective 3: Reflection on SMS Analysis and Results Recommendations .....  | 48 |
| 7.2 What Has Been Learned and Achieved .....   | 48 |
| 8. Reflection .....  | 49 |
| References .....   | 51 |

## Figures Contents

|   |    |
|---|----|
| Figure 1 [19] (SMS ecosystem, its carriers, gateways, and services) ..... | 8  |
| Figure 2 (Project Gantt Chart) .....                                      | 15 |
| Figure 3 (Scrapy spider example) .....                                    | 17 |
| Figure 4 (Received messages from public SMS gateway) .....                | 18 |
| Figure 5 (implementation of regular expressions to find numbers) .....    | 20 |
| Figure 6 (Flowchart for analysis tool) .....                              | 22 |
| Figure 7 (UML diagram for analysis tool) .....                            | 23 |
| Figure 8 (Tkinter text input box) .....                                   | 25 |
| Figure 9 (Tkinter button) .....   | 26 |
| Figure 10 (Tkinter label) .....   | 26 |
| Figure 11 - [36] (Tkinter TopLevel window) .....                          | 26 |
| Figure 12 - [37] (get_file function) .....                                | 26 |
| Figure 13 - [38] (collect_data function) .....                            | 27 |
| Figure 14 (Outdated Version 3 of the multi_finder function) [39] .....    | 27 |
| Figure 15 (else statement eliminating OTP recognition) .....              | 29 |
| Figure 16 (Final Version of multi_finder) .....                           | 30 |
| Figure 17 (regular expression) .....                                      | 30 |
| Figure 18 (nested for loop) .....   | 30 |

|   |    |
|---|----|
| Figure 19 (Get URL Occurrences button command).....             | 30 |
| Figure 20 (collector functions for writing data to files) ..... | 31 |
| Figure 21 (keyword finder function) .....                       | 32 |
| Figure 22 (URL health checker) .....                            | 33 |
| Figure 23 (URL Health Check Result) .....                       | 34 |
| Figure 24 (Dataset Example) .....                               | 35 |
| Figure 25 (GUI Main Body) .....                                 | 36 |
| Figure 26 (Both possible output files).....                     | 36 |
| Figure 27(Results window from URL occurrences).....             | 36 |
| Figure 28 (OTP finder with issue).....                          | 39 |
| Figure 29 (OTP finder with issue fixed).....                    | 39 |
| Figure 30 (Problematic Logic) .....                             | 40 |
| Figure 31(Fixed Logic).....                                     | 40 |

## Tables Contents

|  |    |
|--|----|
| Table 1 - Test Cases .....               | 37 |
| Table 2 - Dataset Analysis Results ..... | 41 |
| Table 3 - URL Analysis Results .....     | 42 |

# Abstract

SMS is one of the most widely used platforms for communication over the world and has been so for many decades now. With such widespread use brings an element of attention to the system, especially from those who seek to abuse it. With an aging infrastructure and lack of information on behalf of users, text messages are an open vector for attack and the actual level of security users have when sending and receiving texts is not something most are aware of. This paper serves as research into how SMS is used, how its flaws are repurposed to cause damage, and as a method to conduct analysis on sample SMS data in an attempt to draw information from its findings which can benefit the security of the stakeholders in the platform. Ultimately, we are looking to provide quantitative evidence on the security of current SMS usage for the benefit of everyday users.

## 1. Introduction

With growing reliance on digital devices in our society, especially smartphones, it is more important than ever to be aware of how secure our data and our devices are.

SMS (short message service) is a critical component of the security infrastructure for many applications and the usage stats for the service are extremely high; recent research found that mobile subscribers in the UK alone sent over 65 billion SMS and MMS (multimedia messaging service) messages in 2019 [1] and over 350 billion text messages are sent each month globally as of 2014 [2].

With its popularity, attention is drawn to exploiting the wide number of users the service can reach for scammers, hackers and more who can use the service in a few different ways to gain data, access or money from unsuspecting victims. "Because the messages are stored on these systems longer than necessary, it increases the window of vulnerability through which the hacker can attack. Rather than having to defend a system for a few seconds to prevent a hacker from stealing a message, it needs to be protected for days, weeks, months." [3] says Wickr CTO Cristopher Howell in reference to the virtual path our SMS messages take between sender and receiver, highlighting the potential weak points in a system that is used by so many but understood by very few.

Analysing just how secure it is to use and finding countermeasures to exploits could potentially save many companies and individuals from having data stolen or compromised, or even just raise awareness to everyday users who are unaware of the associated risks.

This project aims to analyse the nature of security when it comes to the use of SMS and its applications, how and what they are used for and the ways in which the service is used to provide a threat to the key users. Said analysis includes researching how the structure of SMS could be exploited, identifying any known

security flaws, and breaking down the content of example SMS data to contextualise how secure the current use of SMS is.

My approach will be managed by following my Gantt chart detailing key stages of the project and will involve both research and developmental aspects. For my research I will be investigating the structure of SMS; its limitations and advantages, discuss how it is used and present findings on solutions to its issues or alternatives. Development will include the short software development life cycle of a Python based programme used to collect and analyse SMS data as a representation of what is commonly found throughout the SMS network.

Beneficiaries of the work done encompasses the companies which employ SMS messaging as a part of their service; a huge number of companies, with 39% of businesses opting to use text messaging in some way to reach their customers as of 2020 [4]. It also covers the everyday user who owns and uses their mobile phone and its SMS capabilities, as in today's world there is never a guarantee that you are only going to be reached by those in your contact list. Spam is widespread, malicious links hide in fraudulent texts and valuable data is left vulnerable save for the one-time passcodes we use so frequently.

The outcome will reflect research-based advantages and limitations with quantitative information about the security level of SMS as a service and its applications. From this data I will also recommend countermeasures or alternatives to address the existing/found security issues to provide a better understanding of the ways in which we can safely use SMS.

## 1.1. Project Scope

As touched on above, the end product for the project is quantitative data concerning the composition of SMS messages and thus the derived quality of security in the ecosystem. Following these results, I will recommend how issues could be remedied or alternatives could be used. Obstacles experienced could relate to the collection of the data as there are many sources and methods for collecting SMS data but due to project timeline constrictions and ease of access whilst ensuring a varied dataset could mean some methods may be rendered impossible, or data size may be restricted. In continuation of this, I will require resources in the form of a source for the data to be collected from and a computer as a system on which to design and run the algorithm for data collection. Stakeholders will include all users of SMS messaging as well as any company using SMS as a vehicle to deliver information to customers, whether that be advertisements or one-time passcodes. Those who push messages containing spam en masse or utilise the platform in order to maliciously attack others may also be negatively affected by the study.

## 1.2. Project Approach

For the development of the analytical tool, an agile methodology could suit the requirements very well; the tool to be developed is one of no certain definitions other than collecting and analysing SMS data, so there is a level of flexibility in the features to be programmed. Furthermore, the time to deadline is accelerated, the focus is on producing a project rather than planning due to the small project size in this regard, although one consideration is despite using agile as a template, I will be using documentation as it is crucial in the characteristics of the project itself [5]. Although a waterfall methodology does fit the fixed timeline delivery criteria, the increased creativity allowed from agile is something I think could provide a valuable opportunity for the outcome.

In terms of the approach to research, I will be delving into the structure of SMS and breaking down the issues present in the system, how the structure is abused and breaking down existing solutions and their constraints. SMS related works, especially those which are related directly to security concerns or SMS content will be critically reviewed and used as inspiration in the work I plan to do and the conclusions to be drawn.

## 1.3. Aims & Objectives

### ***1. Outline key security flaws or exploits in SMS services and applications.***

- a. Identify and detail the structure of the SMS System*
- b. Detail the use and structure of SMS based applications and services*
- c. Research into any large-scale events of security compromise relating to SMS*
- d. Outline related security flaws and exploits.*

### ***2. Identify and explain both the advantages and limitations of using SMS in providing a service, providing quantitative data to show relevance to security.***

- a. Research limitations and advantages of SMS as a service and evaluate existing solutions.*
- b. Provide SMS dataset ready for analysis*
- c. Analyse and document the provided data and how it is applied to real world context.*
- d. Present findings as quantitative data on the use of SMS with relevance to security.*

### ***3. Reflection on SMS analysis and results recommendations.***

- a. Reflect on whether SMS is indeed secure enough for use or if an alternative must be found with justification.*
- b. Offer recommendations to address results from my analysis*





airlines, retail for digitised receipts and finance for correspondence with banks among many others [7]. For all these ventures, security is surely a primary concern.

## 2.2. Structural Security Issues

With this SMS traffic spending most of its existence in the SMS system unencrypted, there is surely a massive risk to all the data contained in the messages. Further adding to this, ESMEs communicating with SMSCs using the Short Message Peer to Peer Protocol (SMPP), which does not encrypt its content meaning SMS messages originating from ESMEs are susceptible to a man in the middle attack [8], and the Signalling System No. 7, (SS7) standard which facilitates SMS sending and receiving as well as other mobile functionalities has a huge vulnerability where malicious actors can use a Linux PC with a SS7 SDK to emulate a Mobile Switching Centre (MSC), gaining access to all messages travelling through it [9].

Furthermore, the GSM System has similar flaws of its own; as detailed in a paper on the ability to perform a similar attack as detailed above, impersonating a base station whereby the malicious actor can take control of communications and abuse the one-way authentication and key agreement protocol as standard in GSM to disable encryption for devices interacting with the “hijacked” station [10]. Fortunately, these attacks are unlikely to reach large numbers of people, but when they do, they are devastating with a full data compromise taking place, for example, any OTP the user might happen to receive is instantly accessible to the attacker.

A study in 2012 investigated another exploit of SMS structure via the setup and delivery of ‘Silent SMS or Stealth SMS’, whereby, in exploiting the SS7 protocol as above, SMS messages can be delivered to a recipient without their notification and without any actual message being readable. However, because on a network level the device has still technically received a message, subscriber location information is forced to update and can be used to track the victim. Not only this, but the silent messages can be spammed en masse to create a denial-of-service attack on the user’s device [11]. An attack such as this poses a clear threat to targeted individuals, most likely victims being politicians, business executives, celebrities etc.

From these examples it is evident there are several ways in which the security of the SMS system can be compromised by hackers or criminals and there are several points of weakness in the journey that each message takes from sender to receiver, leaving many stakeholders in the operation vulnerable.

## 2.3. SMS Misuse

Aside from the ways the structural integrity of the SMS system is abused to compromise security and take advantage of the stakeholders in the operation, SMS misuse is far more likely to occur and far more relevant to the everyday user. This encompasses activities such as smishing, a form of phishing exclusively used over SMS [12], using links to sites housing malware, general spam and 2FA (two-factor authentication) code theft among many others.

There are several instances of high profile incidents occurring worldwide which fall under this misuse umbrella such as the 2020 EMOTET case where cybercriminals tricked customers into credential theft using malware infection via an SMS posing as trusted US banks in urgent sounding text messages [13], as well as the September 2020 “Rampant Kitten” operation whereby victims were social engineered by criminals to install a specific app which required permission to read SMS content, allowing them to read and use the 2FA codes received by the victim.

Links embedded in SMS messages are often a sign of malicious activity. In the ‘Filecoder’ case, a piece of ransomware was disguised in a link to an image or document which was sent to victims posing as something alluring. Once the file was downloaded, the malware infects the phone, encrypting all files and locking it, presenting only a screen informing the user that their data has been encrypted and the only way to decrypt it was to pay a certain amount to a bitcoin address. However, it doesn’t end here; when the device is initially infected and locked from use, the malware sends the same messages with the malicious link to every contact in the victim’s phone.

Plenty more incidents have occurred, especially in the last few years where OTPs (one-time passcodes) have become the default in second-line defence for account security. Although the objectives of cyber criminals in the acts above lie beyond simply gaining access to victim’s OTPs, it is a common theme throughout most SMS related crime seen in recent years and present times due to the sheer popularity of the infrastructure choice, causing trouble for both the companies which employ the technique and the customers whose accounts are vulnerable.

## 2.4. Existing Solutions & Constraints

SMS phishing or ‘smishing’ is a prevalent form of misuse as previously mentioned and being able to identify exactly what falls under that category could be very beneficial for improving automatic phishing recognition systems. DSmishSMS is a system prototyped by Sandhya Mishra et al. [14], which detects Smishing SMS by

evaluating the legitimacy of the URL (uniform resource locator) in given messages using two phases; domain checking phase and SMS classification phase, which could be key in preventing the loss of user personal data and accounts which contributes to so much of the crime committed. Results from this development achieved an accuracy of 97.3%, proof of an efficient method and something which I could build upon in my project in conjunction with a wider identification of SMS content in relation to other possible methods aside from smishing which could compromise security when SMS is used.

As described earlier in my description of SMS as a system, I talked about the lack of encryption of text messages past the initial routing from the sender's device to the base station. After this, the message is completely unencrypted and can remain stored in the system for far longer than is necessary, leaving the data completely open to an attack. In a 2015 paper written by Ali M Sagheer et al. [15], a hybrid cryptographic scheme for SMS which combines both AES (advanced encryption standard) [16] and RC4 (Rivest cipher 4) [17] algorithms to achieve more reliable security for the function of SMS. Their findings discovered the algorithm was feasibly run on multiple tested mobile phones, smartphones, and non-smartphones alike; examples include the Nokia 5233 which encrypted one whole block in 34 milliseconds and decrypted the same block in 9 milliseconds. Unfortunately, the mobile device using the algorithm must be java compatible and have equal or more RAM (random access memory) than the tested phones, therefore ruling out iPhones which do not have a compatible JVM (Java Virtual Machine) to run Java programs. However, it is proof that there are a large number of mobile phone models which could feasibly use encryption in everyday texting, at least in theory.

Another version of SMS cryptographic security was theorised in the research conducted by Ikechukwu Ibekwe et al. [18], whereby end-to-end encryption was used with a single point of decryption at the receiving node. The method would implement both symmetric and asymmetric encryption; symmetric for the actual encryption and decryption of the data, and asymmetric methods to distribute the keys, deploying the use of a pair of public and private keys for each subscriber to the interaction. They also theorise that message integrity could be verified by using a hash in the message data itself. Regrettably, the theory remained just that and was never implemented by the team so the practicality of such an endeavour cannot be fully analysed. It is however noted that key distribution would be an issue with this method because the number of keys needed for communication is approximate to the square of the number of senders, so it is a further challenge to provide efficient and lightweight encryption as well as distributing these keys efficiently.

In the paper written by Mohammed Shafiul Alam Khan and Chris J Mitchell [19], the structure of the SMS network is discussed as well as the vulnerability of one-way

authentication between the mobile device and network. Building on from their discussion of the security weakness they propose the utilisation of RAND hijacking to enable the network-to-phone authentication within the GSM system in a way that is compatible with the current network infrastructure. Also covered is the modern relevance of such a solution, as despite the use of 3G (UMTS) and 4G (LTE) systems which do provide mutual authentication, GSM is still used so widely that the security flaw has not become irrelevant at all. Findings concluded that it was indeed an improvement over non-mutual authentication and does not affect the SIM-to-network authentication negatively. One stipulation is that the mobile device would need to support 'class e' STK (SIM application toolkit) commands.

From what I have covered it is clear that there are plenty of solutions which aim to combat SMS security issues; mainly in adding additional steps, actions, or procedures to the communication infrastructure in place.

## 2.5. Existing Analysis Works

So, it becomes clear from the background surrounding the security of SMS that there needs to be some clarification for the sake of both the everyday SMS user and the big businesses which use it. Arguments exist internet wide on whether SMS is secure, if OTP is enough protection for your account and how to avoid smishing attempts and more than a few scientific papers have approached the problem from several perspectives and attempted provide answers but there is nothing precisely covering the precise insecurities of SMS and how it is used in a way that can present a threat to users.

One of the similar works of note would be the two successive papers from Bradley Reeves et al. on *"Characterising the Security of the SMS ecosystem with Public Gateways"* [6], two successive studies which investigated the messaging infrastructure as a whole, by collecting SMS data from public gateways; one of the most effective methods of data collection as I will discuss later. In their two papers, both with the same goals and the latter with a larger dataset, the researchers identified the content of said messages in an attempt to label the use of this ecosystem and provide material to combat phone-verified account fraud. Similarly to my study, the aims included attempting to identify the malicious behaviour occurring within the present using SMS messages.

Another project which was also written by Bradley Reeves, Logan Blue, Dave Tian and more, described the detection and analysis of spam traffic in the SMS system with a focus on the use and performance of specifically purposed spam detectors and their classification success rate [19]. Again, similarly to my own work, the part

that spam plays in the current SMS system is in focus and cannot be ignored and reading their paper it is clear that the role spam plays in its composition as part of the ecosystem and therefore the way it can permeate a feeling of uncertainty and fear within users is vital as this creates a knock on effect as to how these same users interact with other truthfully malicious texts.

With spam being such an issue in being so abundant as a message type in the SMS ecosystem, having a way to easily recognise and automatically filter out spam text messages could help many everyday users to avoid potentially harmful messages, allow for better recognition of genuinely malicious messages and waste less time worrying about what is real and what is not. M. Hassan Shirali-Shahreza et al. wrote their paper discussing the idea of using CAPTCHA to recognise and filter these messages [20]. The idea described how a CAPTCHA test would be sent in receipt of an SMS message. This test was in the form of an SMS message including a small image of an object, and 3 named options. If the correct name to identify the object is selected and sent back in a text, then the CAPTCHA is passed, and the initial sender is marked as non-spam. In terms of the viability of the solution, the method can be used on any SMS capable device as the image quality, size and colour befits those capabilities and it is also usable on other SMS capable devices. On the other hand, the logistics of applying such a method is not entirely efficient, given that in order for the test to be conducted there needs to be a whole sequence of messages, consuming both time and resources as sending each message has a cost.

End to end asymmetric encryption is another approach which could highly improve the security of SMS. It is one of the greatest benefits of some alternatives to SMS, services such as WhatsApp and Facebook Messenger, and make the risk of using OTPs far less daunting for users. Mary Agoyi and Devrim Seral explored the possibility of implementing this type of encryption using several methods, documenting their test results. Out of the tested RSA, ELGamal and Elliptic Curve algorithms, they concluded that Elliptic Curve was the most ideal due to its smaller key size when used in a limited resource device such as a mobile phone. Another key point which was noted is that encrypted messages usually end up larger than the original messages which can lead to additional charges when sending the message, but they believe data compression could be used to reduce the data size and remove this downside.

In the approach to solve this problem, the underlying objective of my work will be to provide clarity, quantitative evidence, and structured reasoning to exhibit and explain exactly what SMS is currently being used for and what exactly the everyday users must watch out for regarding safely using the platform.

### 3. Approach

The problem I need to solve is making clear the composition of current SMS content, analysing how secure this content in circulation is and providing a solution in the form of presenting statistics, analysis, and conclusions, reflecting on how safe the platform is for the average user. This solution will be provided in the form of the collection of a series of SMS messages, aiming to be representative of the type of SMS content present within the system, and the analysis of this data via a developed tool which breaks down specific characteristics within the dataset. This analysis will then be summarised and concluded with a documentation of results and how this affects the security of such a system.

#### 3.1. Development Methodology

For the development of the analytical tool, an agile methodology could suit the requirements very well; the tool to be developed is one of no certain definitions other than collecting and analysing SMS data, so there is a level of flexibility in the features to be programmed. Furthermore, the time to deadline is accelerated, the focus is on producing a project rather than planning due to the small project size in this regard, although one consideration is despite using agile as a template, I will be using documentation as it is crucial in the characteristics of the project itself [21].

Although a waterfall methodology does fit the fixed timeline delivery criteria, the increased creativity allowed from agile is something I think could provide a valuable opportunity for the outcome. From a 2012 paper by Gaurav Kumar and Pradeep Kumar Bhatia, there were conclusions drawn which included the following:

“Refactoring leads to higher code reuse and better quality. All aspects of software are improved, from design and architecture to performance of the products of each sprint.” [22] as well as the recognition of agile not being very well fitted to a development with many teams or a large-scale project, both characteristics not seen in this work and reinforcing my decision to follow it.

#### 3.2. Project Timeline

To guide the completion of the project and map critical steps to be taken and their ideal completion dates a Gantt chart was used. This was mapped against the aims and objectives outlined in the introduction so I could segment the work against a

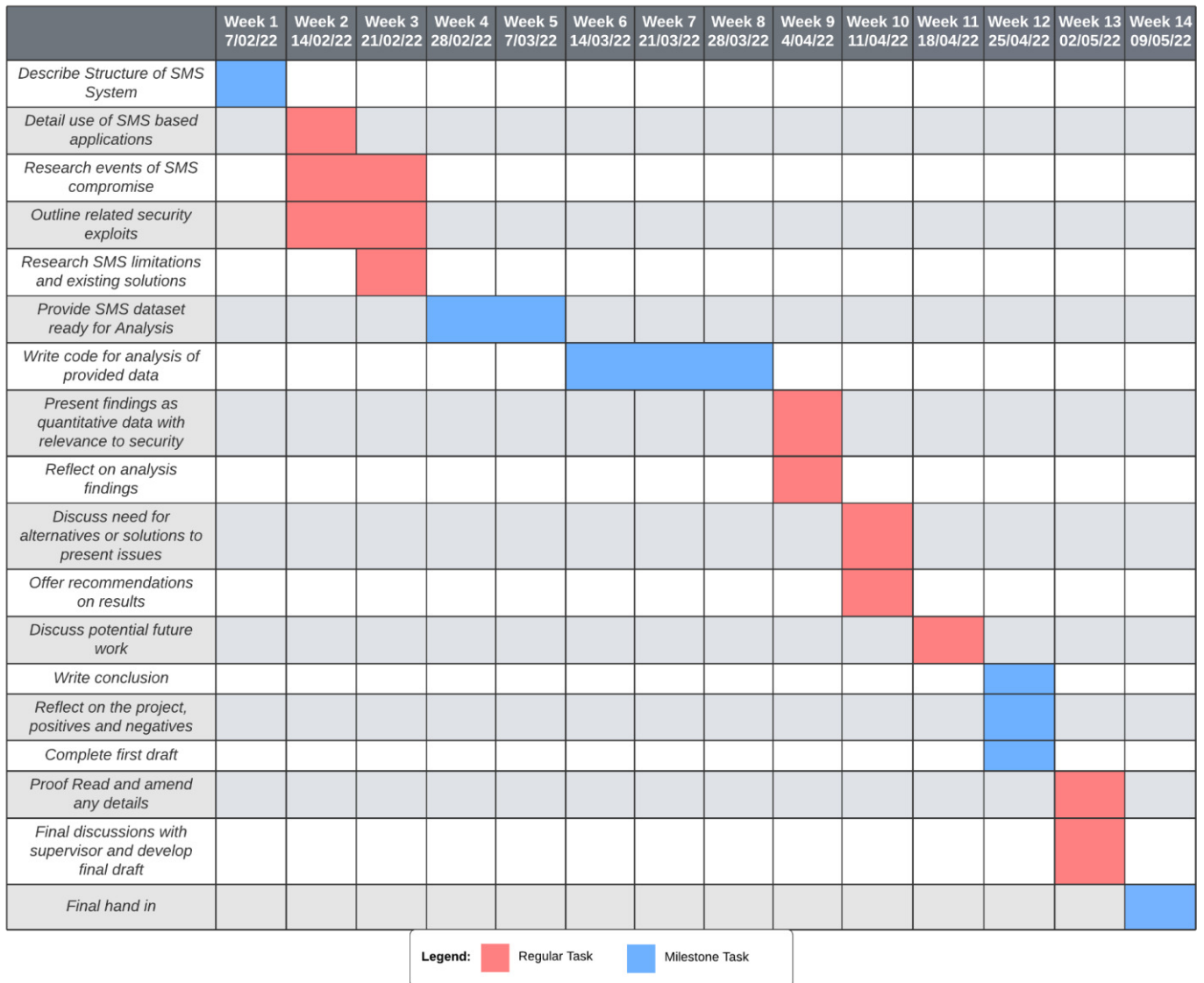


Figure 2 (Project Gantt Chart)

weekly schedule, giving me an idea of when tasks should aim to be completed by, to manage the project effectively. The Gantt chart can be seen below in fig. 2.

### 3.3. Research Approach

In terms of the approach to research, I will be delving into the structure of SMS and breaking down the issues present in the system, how the structure is abused and breaking down existing solutions and their constraints. SMS related works, especially those which are related directly to security concerns or SMS content will be critically reviewed and used as inspiration in the work I plan to do and the conclusions to be drawn. A quantitative research methodology is what I will be following for the scope of the entire project; due to the collection and statistical analysis of concrete data in the form of SMS content in order to define how the SMS system is being used and the inferred security of that use. This data will be

presented in the form of numbers, tables, and potentially graphs as is conventional to qualitative work, in aid of measuring the aforementioned security [23]. Another conventional feature of this kind of research is testing theories or hypotheses and in this project that will take the form of the question how secure the current use of SMS is, the answer to which will be provided with the completion of the SMS analysis tool and the following results when used on the SMS dataset, which is part of the final objectives of the work.

### 3.4. Data Collection

Discuss how the data was collected, talk about using the web scrapers, the public gateways, the issues I had with connections and then manually collecting a sizable dataset with enough variation to prove useful in the study. Talk about if I had more time then I may be able to find a workaround for the web scraper issues and collect data automatically and in much larger quantities but for this project a manual dataset of around 300 messages should be just fine.

As evident from my background research, one of the best ways of collecting a diverse and well-represented SMS dataset is by using the data fed through public SMS gateways [6] as it contains a variety of message types from unique phone numbers, making the figures from the product of analysis far more meaningful to the outcome of the project. It is also by far the most efficient way of collecting data; picking several individuals and scraping their personal text data would both bring ethical considerations into play and create a far less interesting and well featured dataset. By using the public gateways, we can filter through a lot of the personal texts you might receive such as from friend to friend, the contents of which we are not concerned about.

In terms of collecting the dataset from the public gateways, there are plenty of options to choose from, including Octoparse, a cloud-based web scraper [24], Parsehub, an advanced scraping tool [25] and Scrapy, a Python based library allowing for scalable automated web crawlers [26]. Scrapy was my tool of choice as it allows the most control and the deployment is simple and reliable, being implemented with Python which is a language I am very familiar with. After reading through the scrapy documentation and gaining a basic understanding of the concepts such as spiders, selectors and the scrapy shell. Using the documentation material as guidance I managed to build a basic spider which would scrape just one or two pages of material from their provided example website, allowing you to test simple text collection from a basic formatted layout. The result can be seen below in fig.3



```

import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'https://quotes.toscrape.com/page/1/',
        'https://quotes.toscrape.com/page/2/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').get()
            }

```

Figure 3 (Scrapy spider example)

Essentially, this spider would be run from the command line tool and search the given URLs in the list “start\_urls” for the data within the “<quote>” HTML elements. To make this applicable to the data I wanted to collect I first had to find public gateways which had reliable connections and plenty of data to source. I referred to one of the papers I read whilst researching the background of the project [6] where Bradley Reaves et al. compiled a list of all the gateways they were able to collect data from. Given the study being relatively recent I hoped some of the gateways might still be available. Investigation revealed that there were two websites hosting these public gateways which were still functional and active:

1. <https://www.receive-smsonline.net>
2. <https://receive-sms-online.info>

The way the sites work is by organising collected messages via each gateway number; each site has several numbers for varying country codes, and you can sort via each number and see the contents of the message and when it was logged through the gateway as seen below.

| From        | SMS Messages   | Added          |
|-------------|--|----------------|
| 6XXXX       | PayPal: 517642 is your security code. It expires in 10 minutes. Don't share this code with anyone.   | 1 minute ago   |
| Binance     | [Binance] Verification code: 206392. You are trying to log in with a new device. Beware of scam calls and SMS phishing.  | 2 minutes ago  |
| Num2        | <#> Your activation code is: 4022 Enter the code if it did not update automatically. 66G+7YeHkbA   | 2 minutes ago  |
| Binance     | [Binance] Verification code: 667496. If this was not you, please inform Support. Beware of scam calls and SMS phishing. Verify sources with Binance Verify.                                    | 7 minutes ago  |
| Casino 2020 | /creat   | 15 minutes ago |
| Casino 2020 | Thank you for requesting a new password. Your pass code is 0776 Alternatively, click this link to reset your password: <a href="https://www.casino2020.co.uk">https://www.casino2020.co.uk</a> | 15 minutes ago |
| +1855929XXX | 769082 este codul dvs. de conectare Lime. F9CoToz+FUJ  | 36 minutes ago |

Figure 4 (Received messages from public SMS gateway)

With the webpage data now accessible we can go ahead and fit the specific URL and containers needed into my web spider algorithm. Unfortunately, straight away with gateway 1, I encountered issues; where I was usually getting a quick connection to the desired site and the desired data returned, when I used my SMS gateway URL, I was getting a 403-connection error every time, meaning the website was rejecting my connection based on the request headers. After doing some research I found a potential solution in changing my request headers to match a common user agent string [27]. Although this did not fix the 403-error encountered on gateway site number 1, number 2 allowed the connection, but it was temperamental, restarting the spider when testing different HTML element tags to try and capture the correct data caused the website to react badly and start refusing my connection. Given the formatting of the website was extremely confusing and the required data being held within multi-layered tags meaning it was extremely difficult to capture this data using scrapy. Given that these endeavours were only to capture the data on which the analysis was to be performed, and the construction of the actual analysis tool was yet to come, the most sensible decision seemed to be collecting some data manually, so I had something for my tool to be tested on throughout the project. With this manual method, a dataset of 200 messages from both gateway sites, using just UK based numbers was collected and stored line by line into a text file.

## 4. Implementation

SMS analysis describes the process of going through the dataset which I have collected and verifying the type of content that can be found in it as well as picking out a specific criterion such as keywords denoting a specific type of SMS usage, vulnerabilities that could be exploited, or sensitive information such as one-time

passcodes which could lead to account compromise. Breaking down the statistics of this type of content is key in explaining how SMS is being used in the current day and how safe this use is; whether there are precautions or changes that need to take place for the system to be used safely. This is the source of the quantitative data which is the crux of the project and will be able to provide the required insight on how the results affect SMS use in the real world.

## 4.1. Tools Used

During this chapter the planning, platform and implementation of the project development will be detailed, including screenshots of code, flowcharts, and a UML diagram to represent code structure.

### 4.1.1. Platform

For all of the development process the same hardware, software and OS were used. Hardware required to develop the analysis tool, collect the dataset, and run all the code will remain the same throughout and be able to run on a variety of systems. Specifications used can be seen below:

- Processor: i5-8300H @ 2.30GHz 8 core
- Memory - 16GB 2800 MHz DDR4
- Operating System - Windows 10 Home 64-bit
- Graphics Device: NVIDIA GeForce GTX 1060 6GB

It is worth noting that despite these specifications, a machine with far less memory and processor speeds could easily handle the development and a variety of different operating systems are also capable of running python.

### 4.1.2. Python

Python is the most fitting language for use in the entirety of the project for several reasons. Firstly, the degree of personal experience is extremely beneficial as it allows for a smoother development process and ease of use when it comes to more complex libraries for example. Coincidentally, the python libraries are also another great reason to pick the language, there is a huge array of different libraries all helping to perform specific functions as will be covered later in this chapter [28]. Furthermore, for a smaller tool such as my SMS analysis program, using object-oriented programming is not as beneficial because there is far less utilisation of classes, inheritance and other OOP properties; the ease-of-use python presents fits well with the project timeline and the scale of the project as a whole.

#### 4.1.2.1. Python Libraries

##### ***Tkinter***

The output data from the SMS analysis needed to be presented to the user in one way or another. One of the most straightforward ways of doing this, which merges well with user interaction is developing a user interface to act as the medium between user and tool. Tkinter is a standard GUI library for python which allows developers to easily build a GUI for their programs. Although there are several python libraries which facilitate building a graphical user interface, my personal familiarity with the library as well as its detailed documentation made it a clear choice for use in the project.

##### ***Re***

Re is the python library which provides access to regular expression matching. For earlier versions of my project this saw a lot more use, but this changed due to circumstances explained later in this chapter [30].

```
nums = re.findall("\d+", message)
```

*Figure 5 (implementation of regular expressions to find numbers)*

Above is pictured one of the uses of the 're' library, where it is used to find any sequence of numbers when located within a message. In terms of finding substrings within strings in python there are a few different techniques which can be used but regular expression matching is the most straightforward and reliable method for capturing OTPs as there is such a variation between formatting and content which methods such as the native python 'in' operator could not handle as well.

##### ***Requests***

Part of the SMS analysis is connecting to a third-party site which can perform a full URL health check. For this to be possible, we need the requests library to allow the tool to send HTTP requests, sourcing the necessary information for URL details [31].

##### ***Urllib***

Similarly to the requests package, the urllib package is one made of several models designed to assist working with URLs [32]. Specifically, the parsing module is the one which is required to take the specific section of the URL we want to check with the HTTP request in order for it to be valid. Again, this is a simple to implement and use library.

##### ***Json***

The data returned from the HTTP needs to be formatted correctly and neatly in the command line and as the data is being taken from an HTTP response the JSON

encoder and decoder acts as an interchange format to allow for this to happen. It is used in conjunction with requests and urllib.

## 4.2. Risk Assessment

To maintain the development process as well as the project journey from collecting the dataset to providing the analysis results, risk assessments on various aspects of the system and project process had to be considered and accounted for.

### ***Issue - 1***

Although Python is an extremely well documented language and very popular, there are some niche issues encountered which will not have online solutions.

### ***Resolution - 1***

This issue is not entirely solvable but by making sure to read official documentation of used libraries and searching extensively for problems encountered it is likely to be avoided. In the case where this is not possible, changing the approach may be ideal.

### ***Issue - 2***

Some of the libraries used in development will be unfamiliar during development which may lengthen the developmental section of the project planning and take more time than expected.

### ***Resolution - 2***

Read at least basic documentation before reading, if necessary, push back the time schedule.

### ***Issue - 3***

Additionally, using several complex libraries in conjunction with one another has the potential to create unexpected issues where they do not interact as planned.

### ***Resolution - 3***

Research beforehand to ensure that libraries do not have conflicts, make sure to be aware of common issues or flaws found by other users.

### ***Issue - 4***

There is a risk for loss of progress if the device being used for development, a single windows laptop, encounters issues with data corruption in storage.

### ***Resolution - 4***

To combat this, copies of all project contents including the python files and dataset were stored in the cloud using google drive, with regular updates occurring to ensure minimal data loss given a storage failure.

### 4.3. Flowchart and UML

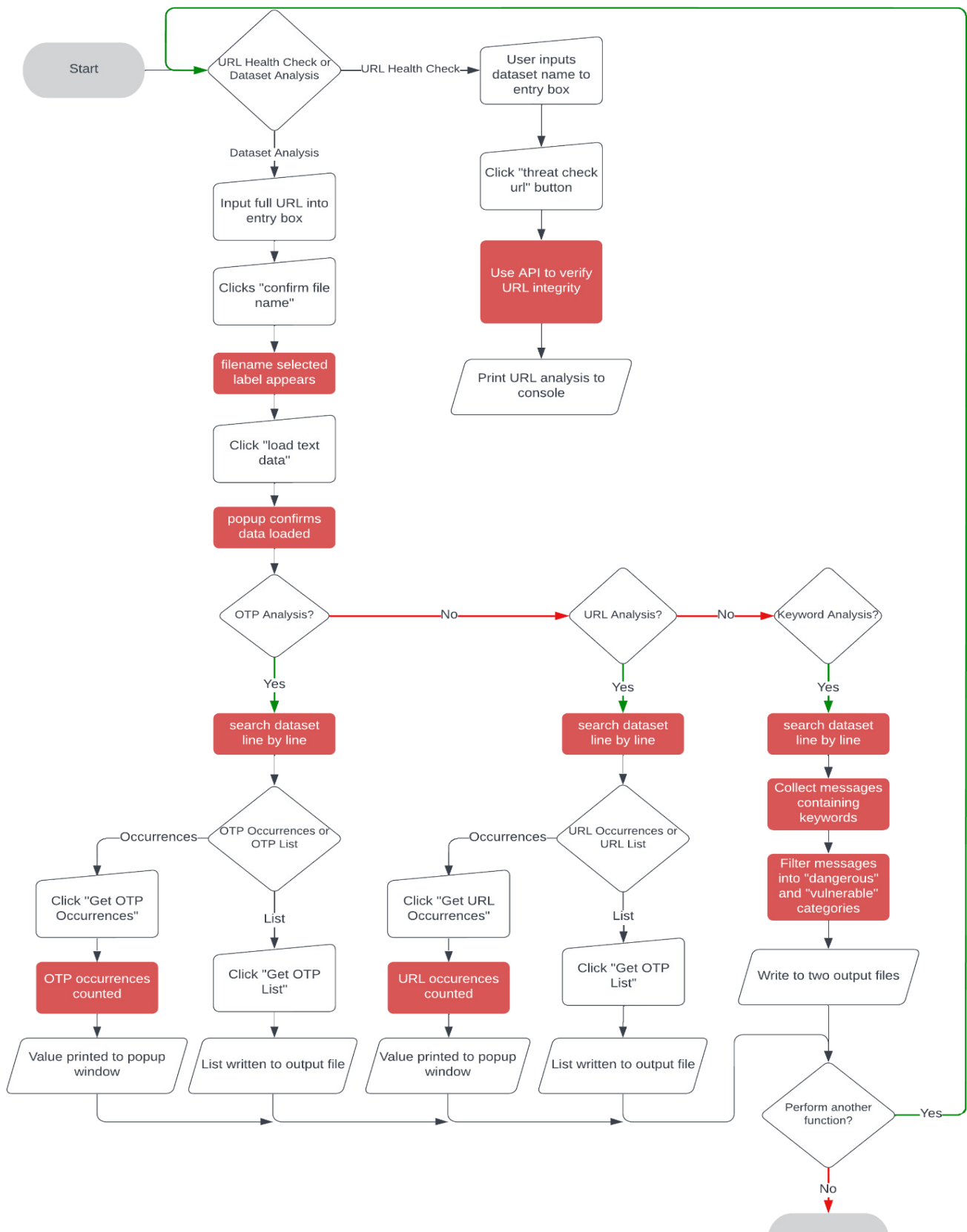


Figure 6 (Flowchart for analysis tool)

Above in figure 6 is the flowchart designed to map out all the possible user interactions made by a user based on the type of analysis we were planning to perform. This was based on some basic planning which included the type of analysis to perform and how that can be implemented into the python tool. It is representative of the flow of the final product which evolved gradually throughout the development stage, so was used for the template of how the program would work in its final iteration. The development stages and changes of the tool itself will be covered later in this chapter.

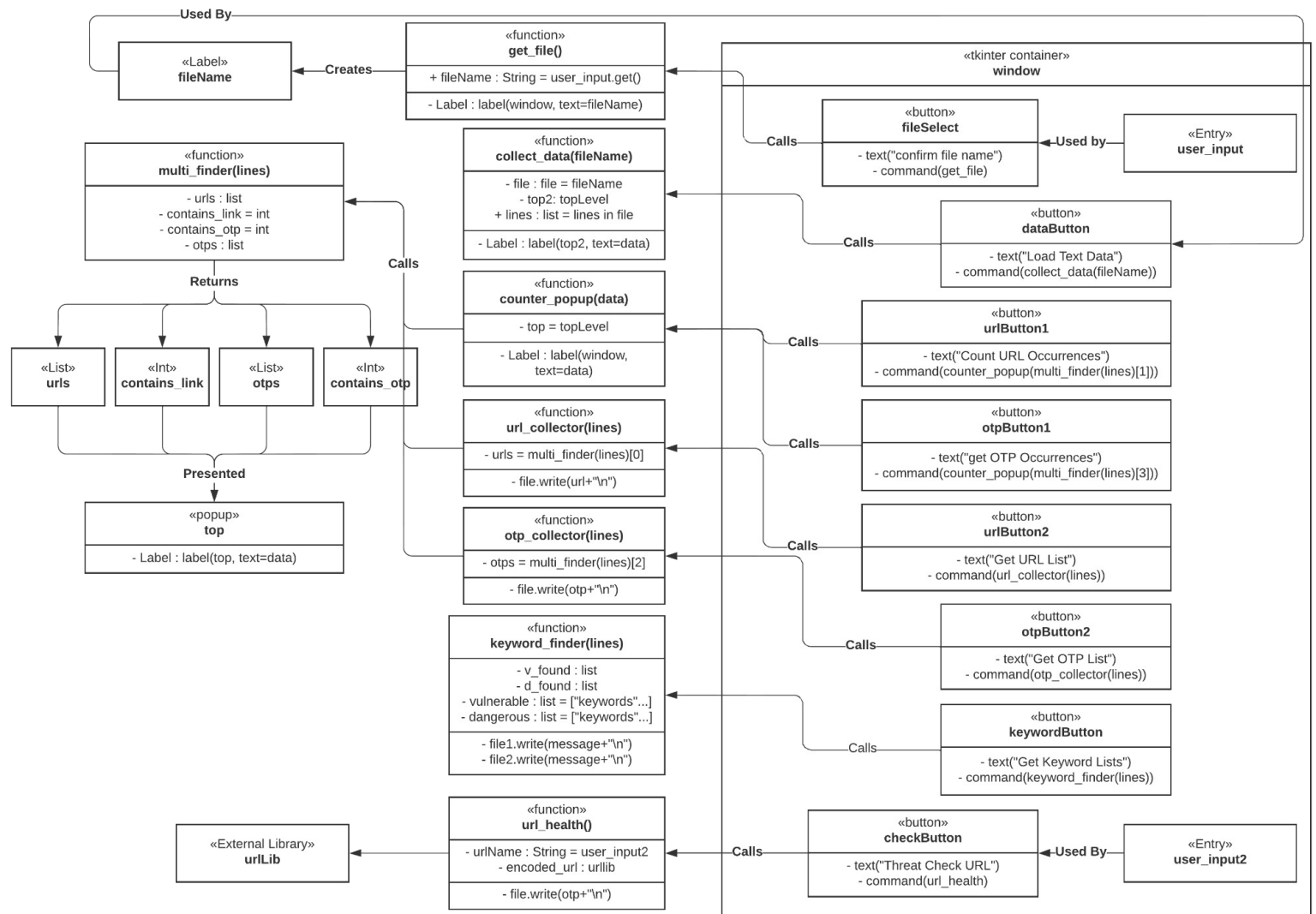


Figure 7 (UML diagram for analysis tool)

Creating a visualisation of my analysis tool is a crucial step in design and lays out the structure and requirements of the program in a relatively standardised way allowing the smooth transfer from idea to code. As my tool is a relatively small project using Python as the chosen programming language, I did not use a class-based UML diagram but a more generalised one based on the main concepts of the tool which includes the interface, and the analysis functions as can be seen above.

Whilst it is possible to use Python as an object-oriented language and implement user defined classes, for my smaller scale project I deemed it unnecessary. Usually for object-oriented programming to be justified, the program would need to be larger, and classes would be useful as they make the code reusable [33].

In the UML diagram I made sure the scope and model of the GUI would be captured correctly, helping to map out how the user interacts with the interface and how that connects on the back end to the functions. The plan is for the interface to remain simple and readable, presenting a series of buttons and a couple of text boxes for the user to interact with to analyse their dataset. Each operation that I am aiming for as a part of my analysis is atomised into individual buttons which are assigned commands to call a function when pressed by the user, as seen by the “calls” keyword outgoing from each button in the diagram.

All elements which the user can interact directly with are within the “window” container, representing the GUI the user must navigate to analyse their dataset. To run the main content analysis functions, they would first need to use one of the text entry boxes, confirming the file name which is then used by the rest of the buttons to make sure any called functions are acting upon the correct filename. Once the user has done this, they have free choice to select the type of analysis they want, all selected via the buttons on the interface which call the functions required. Where possible, functions will be reused to make the code less complex and more readable. For example, if we look at “urlButton1” and “otpButton1” in the diagram, we can see they both call the same function, but the parameters can change so the popup produced by said function outputs the correct data. Furthermore, we can see that there are four total buttons which lead towards the same function “multi-finder(lines)” but return different results.

“urlButton1” and “urlButton2” are the two descriptors for the buttons which allow users to collect URL specific analyses from the dataset, their OTP counterparts being “otpButton1” and “otpButton2”, each having their own command for when pressed. Both “otpButton2” and “urlButton2” directly call an intermediary function which acts as a middleman between the “multi\_finder” function, allowing for the correct data being retrieved.

## 4.4. SMS Analysis Tool

Of all the elements which comprise this project, the largest individual element is the development of the SMS analysis tool. To achieve the objective of providing quantitative data in relation to the level of SMS security, we need this tool to go over the collected dataset and produce a variety of values and statistics which can explain exactly what is contained in current day messages and what users should be aware of.



#### 4.4.1. GUI

As represented by the UML diagram in figure 6, a large portion of the analysis tool is the GUI. In terms of user interaction, a graphical user interface is one of the most ideal options as it provides an intuitive and simple to grasp understanding of system use in most applications, when designed correctly [34].

Not overcomplicating things is very important in this position; the focus still needs to be on the code working behind the scenes, so the aim is to build up a simple but effective graphical user interface that makes getting the required outputs as easy as possible. Therefore, I decided on a window of short width and longer length in order to display the required buttons in a vertical arrangement. To make sure I was designing an efficient user interface, I made sure to adhere to Jakob Nielsen's 10 usability heuristics for UI design [35]. Principles such as number 3, user control and freedom, is matched by the ability to exit the application at any time through use of the commonly seen 'X' button on each interface window, as well as the fact that at no time is the user prevented from interacting with the window. Furthermore, principle number 8, aesthetic and minimalist design was kept in mind when arranging the layout of the user interface; it is why I used colour coded text for each analysis feature and used simple button designs arranged in a vertical layout as to not confuse the user or add unnecessary elements to the window.

It is however important to note that some design decisions are limited by the capabilities of the tkinter library.

##### ***Text Input Box***

One of the first things that needs to happen is for the user to select the file they want to read text data from, in this case the pre-collected data is stored in a text file. The simplest way of the user selecting this data, whilst also having the option to select different text files in anticipation of future application, is a text box entry widget from the tkinter module. There was another option, where the system could be simplified by having the dataset collected during the project hard coded into the program but making the code more reusable seemed to be the best way to take things.

```
user_input = Entry(window)
user_input.place(x=100, y=20)
```

*Figure 8 (Tkinter text input box)*

##### ***Button***

This user input is then confirmed by pressing a button placed below the entry box as shown in fig. 9. Tkinter buttons work as an object which can be assigned properties such as text and command, where command allows us to assign a function to the action of the user pressing the button. In the given example, the 'fileSelect' button calls the "get\_file" function.

```
fileSelect = Button(window, text="Confirm File Name", fg='black', command=get_file)
fileSelect.place(x=100, y=60)
```

*Figure 9 (Tkinter button)*

### ***Label***

The label is a convenient and simple way of presenting pre-defined text towards the user, in this case as shown in figure 10, directing the user on how to interact with the text entry box. This is used several times throughout to inform the user using short and important statements.

```
labell=Label(window, text="Enter URL Here", fg='green', font=("Helvetica", 12))
labell.place(x=100, y=340)
```

*Figure 10 (Tkinter label)*

### ***Popup Window (Toplevel)***

Below is an example of another way the program delivers output to the user, via the 'Toplevel' popup window. The code from figure 11 is inside a function which is called upon a specific button being pressed. Once the function has calculated what it needs to, output data is shown to the user like this in a way that grabs their attention and makes sure they are aware of what exactly has just happened.

```
top2=Toplevel(window)
top2.geometry("180x80")
top2.title("Information Window")
Label(top2, text="Data Collected Successfully").place(x=20,y=20)
```

*Figure 11 - [36] (Tkinter TopLevel window)*

## **4.4.2. Key Functions**

### **4.4.2.1. Get\_file Function**

```
def get_file():
    global fileName
    fileName = user_input.get()
    Label(window, text=f"selected filename : {fileName}").place(x=80, y=40)
    return
```

*Figure 12 - [37] (get\_file function)*

Figure 12 shows the function responsible for grabbing the user input from the text box and assigning that as the value for the file to draw data from. The variable "fileName" must be declared as a global variable to allow for the local variable to be used outside the function as well and the value to remain. Before making this change, I struggled with a file not found error, as the value of the "fileName" was being reset. This is a section of code which came later in development as hard coding the dataset and figuring out exact analysis techniques was my priority.

#### 4.4.2.2. Collect\_data Function

```
def collect_data(fileName):
    try:
        with open(fileName, encoding="utf8") as file:
            global lines
            lines = [line.rstrip() for line in file]
            top2=Toplevel(window)
            top2.geometry("180x80")
            top2.title("Information Window")
            Label(top2, text="Data Collected Successfully").place(x=20,y=20)
    except OSError as e:
        print("File not found")
    return
```

Figure 13 - [38] (collect\_data function)

After the file name has been confirmed it can be used as a parameter for the collect\_data function which opens and reads the specific file into a local list “lines” which is used to house the message data for the rest of the analysis. In order to handle the use case where an incorrect file name is entered, we can use “except OSError” which means we can catch the exception where the file name is not recognised within the assigned directory, and inform the user of this error and let them enter a new file name.

#### 4.4.2.3. Multi\_finder Function

```
def multi_finder(lines)
    urls = [] # List holding all extracted URLs
    contains_link = 0 # Counter for URL links found
    contains_otp = 0 # Counter for the number of OTPs found
    otps = [] # List holding all the extracted OTPs
    for message in lines:
        # findall incase more than one URL in the message
        url_matches = re.findall("http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+", message)
        if url_matches:
            contains_link += 1
            for match in url_matches: # iterate through the list of URLs
                urls.append(match)
        else: # We only need to check for OTPs if there was no URL in the message
            nums = re.findall("\\d+", message) # Finding OTP in the form of any integers within the string
            if nums:
                for j in nums:
                    # The length of OTP usually falls within this number range so limit what we return
                    if len(j) in (3,4,5,6):
                        otps.append(j) # adding the OTP to the OTP list
                        # If adding statement below the otps list becomes a list of messages which contain an OTP instead
                        #otps.append(message)
                        contains_otp += 1
    return urls, contains_link, otps, contains_otp
```

Figure 14 (Outdated Version 3 of the multi\_finder function) [39]

The centrepiece of the tool is the function shown above in figure 14, “multi\_finder”. Throughout development, the exact details of analysis changed into many different iterations. Initially, I investigated the content of the dataset I was using to get an idea of what would return valuable statistics and made a list of these key elements that were featured. The first two of these were finding URLs and OTPs. Frequently I could see these were appearing in the dataset and they are very relevant to security as URLs in messages are frequently used to perform phishing or hide malware and OTPs are the final line of defence for so many users in securing their account and if

this gets compromised there is a possibility for account compromise and personal data being leaked.

Aside from collecting the data itself, counting the occurrences within the dataset gives us a simple figure which can tell us about the overall context of how secure the messages from the dataset are likely to be. A higher number of URL occurrences obviously means that a message from the dataset is more likely to, on average, contain a malicious URL or a phishing link, before we even take into consideration the ability of the tool to analyse individual URLs. "contains\_url" as seen above is a simple integer counter which is used to keep track of this value, where the counter value is increased by one after every URL match is confirmed. Its passcode counterpart can also be seen in both figure 14 and 16, "contains\_otp".

Finding OTPs and URLs in the dataset is a difficult task meaning that there were several options for how the problem could be addressed, with confusion stemming from the idea that I ultimately wanted to keep both URL and OTP analysis within the same function. In figure 14, the third iteration of an approach can be seen, where regular expressions were used to identify both data types, however this caused some issues with repeat values and numbers within URLs being recognised as OTPs, as will be detailed below.

As for OTP values, much the same steps were taken as for the URLs, except a much simpler regular expression was used to find any digits in the messages. Only when these digits were within the required length of 3 to 6 digits are the numbers validated, stored, and added to the tally of total OTP occurrences. This is because in my research looking through hundreds and hundreds of messages, consecutive numbers between these lengths were the most used in OTPs by far. Straying outside this range has the possibility of picking up various other digits in messages which are unwanted, such as mobile numbers.

Several versions were developed, with the task of counting and collecting OTPs and URLs taking many different shapes. Initially without the use of functions, there was a single for loop iterating over each message, and for OTPs and URLs an individual if statement which checked for their respective values and matches. Of course, running the program procedurally like this does not work when we need to adapt to a different dataset or map specific code to a GUI as required, hence the use of functions.

In my second version, I had implemented functions, but struggled with returning the correct values and making sure that there was no redundant code for each step of the analysis. For this, I had two separate functions, one for handling OTPs and one for handling URLs as I was of the mindset that they were not mutually exclusive. This

is not entirely wrong as there are cases where an OTP is in a message alongside a URL, but these are a real minority. However, to make these analyses more reusable, I combined them into an individual function.

This led to the opposite problem where if the tool was analysing a message which contained both a OTP and a URL, the URL would be recognised and collected but the OTP would not, due to the else statement being used, as we can see in figure 15.

```
if url_matches:
    contains_link += 1
    for match in url_matches:
        urls.append(match)
else:
    # We only need to check for OTPs if there was no URL in the message
    nums = re.findall("\d+", message)
    if nums:
```

Figure 15 (else statement eliminating OTP recognition)

In order to remedy this the way in which we check for OTPs had to change. Whilst regular expressions were very compact code-wise, splitting the message string into a list of strings using the “.split()” method would allow for a more accurate reading of message contents and identification of properties. With this there is no risk of picking up a sequence of numbers which are within criteria length but are part of a URL for example.

Along with this there are a few edge cases where a single OTP is split with a whitespace between. To pick up these cases in the occurrence counter, we can check the list of substrings in the message and if two integers are adjacent, it could denote a single OTP.

Whilst amending these details it also came to my attention that when looking at the collected OTP data, the message context is important. Instead of collecting the OTP itself and storing this in a text file for the user, the entire message which contains the OTP can be filtered into the file instead, allowing for more understanding of where the OTP has come from and general details.

Both regular expressions and “split()” have drawbacks and inaccuracies which are due in part to the random formatting of the text data, but ultimately regular expressions proved most accurate when capturing the numerical passcodes. The formatting of the message data is so erratic that capturing all of the OTPs using the “split()” and “isnumeric()” methods is impossible. After experimenting with several different techniques, I found the most effective to be using the regular expression to search for a number of length 3, 4, 5 or 6 but also one that is not found within a URL which had already been identified. It is worth noting that there are a very small number of edge cases where a single OTP is split into 2 numbers separated by whitespace. Due to time constraints and low impact, I did not spend the time handling these.

```

def multi_finder(lines):
    urls = []          # List holding all extracted URLs
    contains_link = 0   # Counter for URL links found
    contains_otp = 0    # Counter for the number of OTPs found
    otps = []          # List holding all the extracted messages containing OTP(s)
    for message in lines:
        strings = message.split() # Split the message into individual strings
        nums = re.findall("\d+", message)
        for element in strings:    # Iterate through the strings
            current_url = ""       # Grab URL to make sure we dont pick up OTPs
            if "http" in element:  # Check for URLs
                contains_link += 1
                urls.append(element)
                current_url = element
        if nums:
            for j in nums:
                if len(j) in (3,4,5,6) and j not in current_url:
                    otps.append(message) # Filter messages containing OTPs
                    contains_otp+=1
    return urls, contains_link, otps, contains_otp

```

Figure 16 (Final Version of multi\_finder)

Furthermore, whilst changing how passcodes were collected, I experimented the time to completion for using both the “split()” and “in” methods compared to regular expression for finding URLs. The results as seen below show that even if time efficiency was a key concern, there is not much difference between the methods so either are acceptable.

Elapsed time : 0.5616556

Figure 17 (regular expression)

Elapsed time : 0.6007957000000002

Figure 18 (nested for loop)

Outputs from this function include a list of messages which contain at least one OTP, a list of URLs, and individual counters for the occurrences of OTPs and URLs. As seen below in figure 19, the “collectors”, or the lists which collect URLs and messages containing OTPs are called from within these functions. On the other hand, the “contains\_link” and “contains\_otp” occurrence values are returned from calling the “multi\_finder” function when it is used directly in the command parameter of the specific button in the GUI, shown below.

```
command=lambda: counter_popup(multi_finder(lines)[1])
```

Figure 19 (Get URL Occurrences button command)

#### 4.4.2.4. Url\_collector and otp\_collector functions

```
def url_collector(lines):
    # Access the return value that is the list holding all the extracted URLs
    urls = multi_finder(lines)[0]
    with open('url_list.txt', 'w', encoding="utf-8") as file:
        for url in urls:
            file.write(url+"\n")
    return

def otp_collector(lines):
    # Access the return value that is the list holding all the extracted OTPs
    otps = multi_finder(lines)[2]
    with open('otp_list.txt', 'w', encoding="utf-8") as file:
        for otp in otps:
            file.write(otp+"\n")
    return
```

*Figure 20 (collector functions for writing data to files)*

In the explanation above I covered how the calculations for both OTP and URL finders work and we can see given figure 16 the values the function returns. In order to actually make use of these returned values, there are separate functions “url\_collector” and “otp\_collector”, are each assigned to a graphical interface button and call the “multi\_finder” function with the chosen return value being the relevant list. Once this data is fetched they can write to their assigned output files in order to present the data neatly for the user. As touched on earlier, for the “otp\_collector” function, the entire message which contains an OTP is written to the file in order to allow for message context.

#### 4.4.2.5. Keyword\_finder function

Another analysis function which was a desired objective was identifying keywords which could connote specific behaviour or likely malicious intent. In order to do this there had to be a dictionary of relevant keywords which could point us towards specific circumstances or type of text message use. I split these into two categories, “vulnerable” and “dangerous”, where vulnerable keywords point toward SMS messages which potentially contain very sensitive personal data and dangerous keywords are likely to be found in malicious messages.

```

def keyword_finder(lines):
    v_found = [] # List of messages containing a vulnerable keyword
    d_found = [] # List of messages containing a dangerous keyword
    vulnerable = ["password", "cvv2", "credit", "address", "debit", "card"]
    dangerous = ["free", "paypal", "account", "secure", "suspend", "protect",
    for message in lines:
        v = [ele for ele in vulnerable if(ele in message.lower())]
        d = [ele for ele in dangerous if(ele in message.lower())]
        if v:
            v_found.append(message)
        elif d:
            d_found.append(message)
    with open('vulnerable_list.txt', 'w', encoding="utf-8") as file1:
        for message in v_found:
            file1.write(message+"\n")
    with open('dangerous_list.txt', 'w', encoding="utf-8") as file2:
        for message in d_found:
            file2.write(message+"\n")
    return

```

Figure 21 (keyword finder function)

Similarly to the “multi\_finder” function, the keyword finder setups up the capture variables of “v\_found” and “d\_found” which hold messages containing an associated keyword, before going through the dataset line by line and using the “in” operator again to check if each of the words in the keyword lists can be found in the message. Messages meeting the criteria are then written to a new text file, allowing the user to look at the messages as a whole and see why they have been flagged. This proves very useful as when used in conjunction with the URL health checker, we can pick up on messages which may be attempts at phishing or spreading malware without having to do a case-by-case manual analysis. Furthermore, additional keywords can easily be added to the list, with more experience using the analysis tool and looking through messages, a better idea of what keywords are best for finding malicious behaviours can be built up. For collecting initial keywords research was conducted into the most frequently used words for phishing and malicious messaging, with a paper by Gaston L’Huillier et al. proving very useful in showing proven examples [38].

#### 4.4.2.6. URL Health Check

The final quantifiable analysis performed on the dataset is the analysis of the URLs themselves. Whilst there are other types of analyses that could be performed, most are demanding time wise and require a far more in-depth analysis. This includes such aspects as sentiment analysis, where the exact contents of each message could be broken down word by word and the presumed sentiment of the message derived into a series of pre-selected categories to define exactly what the message is being used for. This unfortunately is something which lies outside the scope of this project.

However, verifying what exactly is behind each URL contained within a message can provide great insight into how likely an unknown link received in a phone inbox is to be malicious in any way. Malactors are continually using links to try and deceive unsuspecting victims, as their very nature means to have any clue of what is behind



it, you must click. Here lies the issue, as in many cases, once that is done, it is already too late. Following a link can start a download directly onto your handheld device, provide its owner with the device information of anyone who is accessing it or simply be acting as a familiar website in a ruse to collect sensitive information such as names, addresses and passwords. To combat this, my application has the ability to use an API to access a third-party URL checker which provides detailed information on given URLs based on blacklist data and deep machine learning by IPQS [40]. This ensures an accurate analysis as the scanning algorithms utilise latest threat data and content analysis whilst minimising false positives. Below is the implementation of this in my code.

```
def url_health():
    urlName = user_input2.get()
    print(urlName)
    encoded_url = urllib.parse.quote(urlName, safe='')
    api_url = "https://ipqualityscore.com/api/json/url/LF4oqyoU6khP49mpKCB3MKFMBM8SdE2T/"
    data = requests.get(api_url + encoded_url)
    # Printing report to console instead of to file to keep things
    # simple, also means there is no overwriting
    print(json.dumps(data.json(), indent=4))
    return
```

*Figure 22 (URL health checker)*

Starting off with fetching the given URL input into the text box by the user, the function then proceeds to format the URL into a valid URL string ready to be combined with the “api\_url” for the request action. The variable “api\_url” also contains a specific key as part of the request, the long string of characters at the end of the link. This is generated by creating an account on the site, in this case I signed up for a free account which allows for 5000 uses of the API per month, more than enough than required for this project, especially given this specific analysis is performed manually on an individual basis. “Data” is the returned analysis results which are printed, after another round of formatting, to the console. Below in figure 23 can be seen the results after an example analysis using a URL extracted from the dataset.

```

https://www.hooyu.com/s/oK6t
{
  "message": "Success.",
  "success": true,
  "unsafe": false,
  "domain": "hooyu.com",
  "ip_address": "52.49.247.91",
  "server": "N/A",
  "content_type": "text/html;charset=UTF-8",
  "status_code": 404,
  "page_size": 1087,
  "domain_rank": 0,
  "dns_valid": true,
  "parking": false,
  "spamming": false,
  "malware": false,
  "phishing": false,
  "suspicious": true,
  "adult": false,
  "risk_score": 75,
  "category": "Business",
  "domain_age": {
    "human": "9 years ago",
    "timestamp": 1383071451,
    "iso": "2013-10-29T14:30:51-04:00"
  },
  "request_id": "4AkCgnNNH7"
}

```

*Figure 23 (URL Health Check Result)*

## 5. Results and Evaluation

Overall, the project was a success, with the aims and objectives being met at least to a certain degree. Research into the security flaws and exploits currently seen in the SMS ecosystem was conducted and successful, with the analysis of previous works also being carried out well. These critical reviews covered previous attempts at remedying security issues and any other research done relating to the content or security of SMS use. Although the dataset was not collected remotely as anticipated and a larger dataset could have been more beneficial in providing a varied dataset, manual collection for a dataset of 200 messages proved to be adequate in showing the application of the analysis tool itself. Another success, the tool does provide quantitative data based on the security of SMS messages by analysing several features within those messages accurately and efficiently. This section will cover a more in-depth evaluation of the project outcome.

### 5.1. Exhibits

Two of the quantifiable outputs from this project have been the dataset and the analysis tool. Although the dataset is not entirely the focus, it was a crucial step in the specification and design of the project and required a large amount of time to collect, especially due to attempted methods. In the end the dataset was 200 messages large, formatted into a text file with messages being separated by a new line. Aside from this the entirety of the analysis tool was able to be developed and fit the project objectives, alongside a GUI which was somewhat beyond initial aims.

### 5.1.1. Collected Dataset

```
1 [KoA]222498(FunPlus verification code. Please DO NOT tell others).  
2 Use 593 187 to verify your Instagram account.  
3 Твоят код за Tinder е 283345 Не го споделяй  
4 Hello, where are you from ?  
5 Votre code de connexionLime est le250276.  
6 Use 241 879 para verificar a conta do Instagram.  
7 [Potato]your verification code is: 78019.  
8 Use 661181 as your login code for Tinder. (Account Kit by Facebook)
```

*Figure 24 (Dataset Example)*

Collecting the dataset from a public SMS gateway ensured there was a variety of different types of texts, from one-time passcodes to plain text messages to phishing links. This is perfect for use with my analysis tool as it exhibits all the different functions and produces quantitative results which help answer the ultimate question posed by this project; how secure is the use of SMS?

### 5.1.2. Analysis Tool

Initially there was not a concrete idea for the form the analysis tool needed to take, but after developing the basic structures for dataset analysis within python it became apparent that running a command line program would only complicate the use of the application and make it more difficult to present the actual findings to the user. Hence, during the development journey, a GUI approach was adopted and eventually integrated into the code which was written and reformatted in order to map directly to the features of this user interface, proving for a simple and seamless solution which output results cleanly and efficiently.

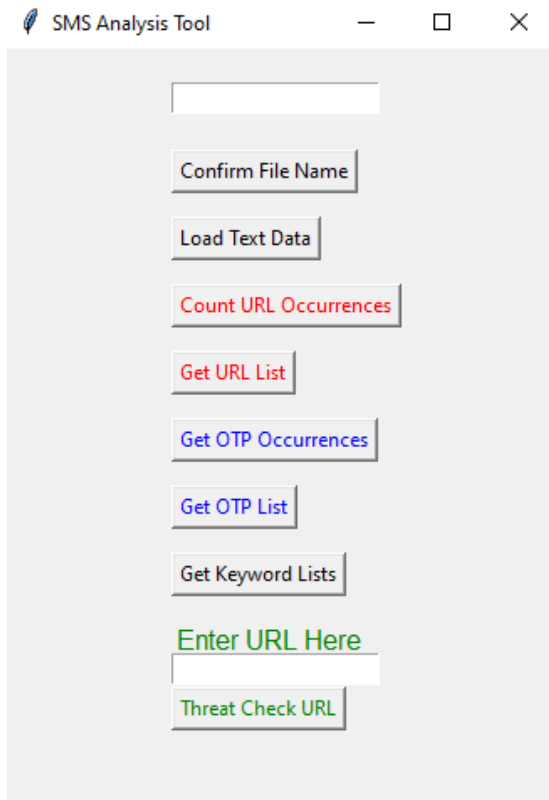


Figure 25 (GUI Main Body)



| Name   | Date modified    | Type          | Size  |
|--|------------------|---------------|-------|
|  otp_list.txt | 24/05/2022 21:55 | Text Document | 13 KB |
|  url_list.txt | 24/05/2022 21:58 | Text Document | 2 KB  |

Figure 26 (Both possible output files)

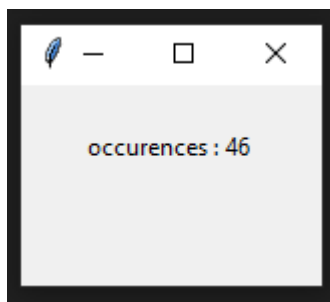


Figure 27(Results window from URL occurrences)

All shown figures are representative of the project results, alongside fig. 23 which can be seen in section **4.4.2.6** showing the results of a URL health check. Included in the URL health check are such categories as “unsafe” which is an overall assessment judging whether the URL is malicious or not. Other returned values include whether the site performs any phishing activity, spamming, or adult activity as well as providing a “risk score” which is an integer from 0 to 100 based on the

overall assessment of how likely the linked site is to cause any damage or malicious behaviour.

## 5.2. Test Cases

To ensure the working condition and results from use of the application, use cases have been designed to cover various aspects of use and types of interaction.

*Table 1 - Test Cases*

| Test Case Number | Test Title  | Description   | Expected Outcome  | Actual Outcome           | Comments  | Status |
|------------------|---|---|---|--------------------------|---|--------|
| 1                | Attempt to load text data from non-existent file. | When entering a filename to draw SMS data from, enter the name of a file that does not exist then confirm file name and load text data to see how the program handles it. | "File not found" printed to console, program still runs allows for another attempt                      | Matched expected outcome | Although loading text data does not work, confirming the file name does even if the file does not exist | Pass   |
| 2                | Load data successfully from a file                | Input a legitimate file name with data in and confirm file name and load text data  | Program will load the data and output popup confirming data was collected successfully.                 | Matched expected outcome | N/A   | Pass   |
| 3                | Count URL occurrences                             | From a dataset with exactly 5 URLs make sure the program also detects 5 URLs  | Program will find and declare all 5 URLs found  | Matched expected outcome | N/A   | Pass   |
| 4                | Count OTP occurrences                             | From a dataset with exactly 5 OTPs make sure the program also detects 5 OTPs  | Program will find and declare all 5 OTPs  | Matched expected outcome | N/A   | Pass   |
| 5                | Collect the messages containing OTPs              | Collect all 5 messages which contain a OTP from a reduced dataset consisting of total 5 messages all containing a OTP   | Program will find all 5 OTPs and write the associated messages to an output file correctly              | Matched expected outcome | N/A   | Pass   |
| 6                | Collect the URLs from messages                    | Collect all 5 URLs from a reduced dataset consisting of 5 total messages all containing a URL   | Program will find all 5 URLs and write the URLs to an output file correctly                             | Matched expected outcome | N/A   | Pass   |
| 7                | Check OTP collection range                        | From a reduced dataset consisting of messages containing various length numerical codes all outside the OTP pickup range check whether they are ignored or not            | Program will not recognise any of the codes not fitting the correct criteria and return a counter of 0. | Matched expected outcome | N/A   | Pass   |

|    |  |   |  |   |   |      |
|----|--|---|--|---|---|------|
| 8  | Test URL Health Check                            | From three unique extracted URLs perform a URL health check in the analysis tool and ensure the analysis output is as expected.   | All three URLs will have a detailed and complete analysis  | Matched expected outcome                              | All 3 URLs returned a 200 code so analysis was successful, and details were full and complete   | Pass |
| 9  | Test URL Health Check for Known Safe URL         | Using a reliable and safe URL:<br><a href="https://www.amazon.co.uk/">https://www.amazon.co.uk/</a><br>Check that the URL analysis is accurate.   | The analysis will be complete and show that the URL is very safe and not showing flags for any malicious activities. | Matched expected outcome                              | Risk score was zero, no negative flags were shown.  | Pass |
| 10 | Test URL Health Check for known Unsafe URL       | Using a URL from a fake site database [41], perform a URL health check and make sure it is flagged as unsafe.   | The analysis will have a high risk score above 80 and will flag as unsafe overall                                    | Matched expected outcome                              | Risk score = 97 and the site was flagged as overall unsafe  | Pass |
| 11 | Test for Messages Containing Vulnerable Keywords | Using a reduced dataset with 5 messages all containing at least one "vulnerable" keyword, check the keyword finder picks up all 5 messages  | All 5 keywords will be identified, and their related messages written to a file                                      | Matched expected outcome                              | N/A   | Pass |
| 12 | Test for Messages Containing Dangerous Keywords  | Using a reduced dataset with 5 messages all containing at least one "dangerous" keyword, check the keyword finder picks up all 5 messages   | All 5 keywords will be identified, and their related messages written to a file                                      | Only 4 keywords were identified                       | Refer to " <i>Complex Test Case 3</i> "   | Fail |
| 13 | Test for OTP edge cases                          | Using a reduced dataset of 5 messages all containing numbers in various formats which are not OTPs but match some of the OTP recognition criteria see how many are detected as a false positive | At least 3 out of 5 of the messages will be flagged for containing an OTP  | 4 messages out of 5 were flagged as containing an OTP | There is no way for the tool to differentiate from phone numbers, dates and other numerical data as well as OTPs, hence the false positives                     | Fail |
| 14 | Test for URL edge cases                          | Using a reduced dataset of incomplete URLs, see if they are flagged as URLs   | All 5 URLs will be identified if they contain "http"   | All 5 messages were flagged as URLs                   | Although result is as expected, this is a negative outcome. Adding the ability to confirm a URL is complete before flagging would be ideal but is quite complex | Pass |

### 5.2.1. Complex Cases

#### *Complex Test Case 1*

When recognising elements within messages, especially OTPs, there is a difficulty due to such varied formatting. Some messages contain OTPs within brackets, others split one number, so it is separated by whitespace and others are just formatted poorly with no whitespace being left between a passcode and the next word in the message. To test this, I built a specific use case where I created a test dataset consisting of a small number of messages containing all of these examples so the accuracy of OTP recognition could be tested on a measured scale.

**Title:** OTP Recognition Fringe Case 1

**Description:** 10 messages with varying types of formatting and structure are the test dataset to see if the analysis tool can correctly identify how many OTPs are contained within them.

**Expected Outcome:** Manually identified OTPs match the number of tool-identified OTPs, which is a total of 9.

**Actual Outcome:** 11 OTPs recognised

**Comments:** The analysis tool is picking up two additional OTPs incorrectly. This is due to two reasons. Firstly, one of the messages contains a single OTP split into two separate numbers. Secondly, another of the messages contains some incoherent characters in the message alongside a legitimate OTP. Within these characters is a 3-digit number which is being incorrectly identified. Both these cases could be easily remedied by limiting the number of OTPs found per message to 1 by amending the following section of code in the “multi\_finder” function:

```
if nums:
    for j in nums:
        if len(j) in (3,4,5,6) and j not in current_url:
            otps.append(message) # Filter messages containing OTPs
            contains_otp+=1
```

Figure 28 (OTP finder with issue)

```
if nums:
    not_found = 1
    for j in nums:
        if len(j) in (3,4,5,6) and j not in current_url and not_found==1:
            not_found = 0
            otps.append(message) # Filter messages containing OTPs
            contains_otp+=1
```

Figure 29 (OTP finder with issue fixed)

This solution is applicable because there is only ever going to be one total legitimate OTP per message.

**Status:** Failed (Fix applied)

### Complex Test Case 2

In various tests where the keyword analysis is used, many more messages than expected are flagged as “dangerous”. This test case is an attempt to identify why.

**Title:** Dangerous keyword oversaturation test

**Description:** 50 messages picked at random from the dataset will be tested for the dangerous keyword analysis to check the rate of detection

**Expected Outcome:** 10 or less messages will be confirmed as dangerous

**Actual Outcome:** 11 total messages recognised and flagged as dangerous

**Comments:** In this test all 11 messages were flagged as dangerous due to the keyword “account”. The “Dangerous” keywords list contains “account”, which is a frequently used message found in many messages which are not malicious. Although the word itself is also used in some malicious messages, the number of false positives mean that ultimately the word should be removed from the dataset as it is doing more harm than good to the analysis results, and it is a very quick fix. After the fix has been completed, only 2 out of 50 messages are flagged as dangerous, a far more realistic statistic.

**Status:** Failed (Fix applied)

### **Complex Test Case 3**

As seen in test case 12, there was an unexpected result when dealing with testing the effectiveness of the “dangerous” keywords. A message containing the dangerous keyword “card” was not recognised and this test case is designed to find out why and how this can be fixed.

**Title:** Dangerous keyword missed identification test

**Description:** Using a dataset which contains use of every single dangerous keyword exactly once and performing a keyword analysis should show us where the issue lies

**Expected Outcome:** No messages will be flagged as dangerous because of flawed logic

**Actual Outcome:** 11 total messages recognised and flagged as dangerous

**Comments:** The operation logic prioritises the words in the “vulnerable” keyword list incorrectly, this will be corrected by changing the “elif” statement to an “if” statement.

```
if v:
    v_found.append(message)
elif d:
    d_found.append(message)
```

Figure 30 (Problematic Logic)

```
if v:
    v_found.append(message)
if d:
    d_found.append(message)
```

Figure 31 (Fixed Logic)

**Status:** Failed (Fix applied)



## 5.3. Results tables

In this section the overall results of a full analysis performed on the 200-message dataset collected during the project timeline will be documented with comments on accuracy and overall effectiveness in relation to the project aims.

### 5.3.1. Dataset Analysis Results

*Table 2 - Dataset Analysis Results*

| Analysis Type         | Results (out of 200) | Comments   |
|-----------------------|----------------------|--|
| OTP Occurrences       | 132 OTPs identified  | OTPs do make up a large quantity of the texts available through public SMS gateways, so this is to be expected, but given the popularity of OTP use regarding account security it is not entirely a surprise.  |
| URL Occurrences       | 46 URLs identified   | Every single URL was correctly identified and collected from the dataset, there is an interesting variety of types of links here. There are 7 links which redirect to PayPal settings, 10 links which redirect to gambling websites of various varieties and plenty of other types besides.  |
| "Dangerous" Messages  | 30 Messages          | A good number of messages are flagged correctly for containing buzzwords like "free" and in these cases even if the message itself is not inherently harmful or malicious, the type of behaviour is, and users associating this vocabulary with messages they perceive to be safe is part of the problem why SMS is so widely used as an attack vector. On the other hand, due to PayPal being used so widely to make victims feel comfortable clicking a link, there are several legitimate messages from PayPal which are being wrongfully flagged as dangerous. |
| "Vulnerable" Messages | 26 Messages          | For this case, most captured cases are correctly identified as most of them are messages containing a direct link to reset a password. Having this message be interrupted can instantly mean the loss of account and personal data to a malicious actor.   |

#### *Extracted OTP Message Examples*

1. "WeChat code (396018) may only be used once for account recovery."
2. "Your Proton verification code is: 341777"

#### *Extracted URL Examples*

1. "https://www.hooyu.com/s/oK6t"
2. "https://verification.sandbox.meonly.co/verification/46b8c135-0866-403d-ad70-bbe44e3d6719 "

#### *Extracted Dangerous Message Examples*

1. "Congratulations! Your Tide account is now up and running. Your card will be delivered to the company trading address you provided. Welcome to Tide!"
2. "Thank you for ordering with Sit Web 3 UK -Papa's Coupon for £18.10. You can also pay quickly and securely by card using the below link. <https://sit-web-3>"

#### ***Extracted Vulnerable Message Examples:***

1. "Thank you for requesting a new password. Your pass code is 6883 Alternatively, click this link to reset your password: <https://www.casino2020.co.uk>"
2. "Tap to reset your Instagram password: <https://ig.me/27pnMJGWLmlrw>"

### 5.3.2. URL Analysis Results

Due to the sophisticated nature of the URL analysis within the tool, to demonstrate the results, ten total URLs will be extracted from the pre-collected dataset and a full analysis report will be run, the results documented in this table. Because of the nature of this analysis being run on a case by case basis, testing all URLs present within the dataset is relatively redundant, because although it would provide complete insight as to whether any truly malicious URLs are found and the average makeup of a URL found within SMS messages, the point of the URL health analysis is to be able to use it on a specific URL of your choice to verify it's safety. Using 10 URLs for these test results means we can get an idea of what the results are likely to be like for most cases without having to perform the analysis on every single URL from the dataset.

*Table 3 - URL Analysis Results*

| URL Number | Full URL  | Analysis Success | Unsafe | Spam  | Malware | Phishing | Suspicious | Adult | Risk Score |
|------------|---|------------------|--------|-------|---------|----------|------------|-------|------------|
| 1          | <a href="https://bbh.dashadmit123.com">https://bbh.dashadmit123.com</a>                     | True             | False  | False | False   | False    | True       | False | 61         |
| 2          | <a href="http://jmmky.com/39fa8">http://jmmky.com/39fa8</a>                                 | True             | False  | False | False   | False    | False      | False | 0          |
| 3          | <a href="https://py.pl/895gdsEq2nb">https://py.pl/895gdsEq2nb</a>                           | True             | False  | False | False   | False    | False      | False | 0          |
| 4          | <a href="https://www.staytouch.com/how-to-stayt">https://www.staytouch.com/how-to-stayt</a> | True             | False  | False | False   | False    | True       | False | 75         |
| 5          | <a href="http://mrspin.co.uk/c/a/A3HgBd">http://mrspin.co.uk/c/a/A3HgBd</a>                 | True             | False  | False | False   | False    | False      | False | 0          |
| 6          | <a href="http://bb0ss.uk/ee8">http://bb0ss.uk/ee8</a>                                       | True             | False  | False | False   | False    | True       | False | 77         |
| 7          | <a href="http://drslot.mobi/2a81c">http://drslot.mobi/2a81c</a>                             | True             | False  | False | False   | False    | False      | False | 0          |

|    |   |      |       |       |       |       |      |       |    |
|----|---|------|-------|-------|-------|-------|------|-------|----|
| 8  | <a href="https://cv2dev.med">https://cv2dev.med</a>                     | True | False | True  | False | False | True | False | 92 |
| 9  | <a href="https://www.hooyu.com/s/oK6t">https://www.hooyu.com/s/oK6t</a> | True | False | False | False | False | True | False | 75 |
| 10 | <a href="https://rwdzulk.com/3a308b81">https://rwdzulk.com/3a308b81</a> | True | False | False | False | False | True | False | 55 |

## 5.4. Evaluation

This section will detail a critical evaluation of the above test cases and results tables, an explanation of how suitable the program is in fitting the requirements for the project and the reasoning behind the tests conducted.

### 5.4.1. OTP Analysis

The OTP analysis is a crucial section of the analysis tool's implementation given how common OTPs are within the SMS ecosystem. I knew from the start when collecting the data how common they were and knew their specific analysis aspect would have to be accurate and sophisticated to well represent the true frequency. Overall, I feel like the implementation worked very well and is as very accurate. After encountering an issue like the one detailed in the complex test case 1, the small problems with accuracy were totally ironed out and in the final test carried out on the entire 200-word dataset, the results align and are exactly as expected. From the 200 total messages, 132 messages were identified as OTP messages, and this is confirmed by the outputted file containing all messages containing an OTP as this is also a total of 132 messages.

However, there are a handful of anomalies, whereby false positives are being flagged, in one case due to a message containing a business telephone number and in another case a date in the format DD/MM/YYYY was flagged as the year was identified as an OTP. Despite these edge cases, most are correctly identified and flagged which is important given the amount of use OTP sees in the dataset and the overall use of SMS messaging, and this aspect fits the requirements well as it gives us a plain figure telling us exactly how many messages OTPs are, a technique used to secure accounts. With many messages containing a passcode used to change passwords, secure accounts, or gain access to a login, the risk of security being compromised is clearly high; if one of the messages from the dataset was intercepted at random, statistics tell us that it is likely an OTP message meaning the listener could theoretically gain access instead. Ensuring users are aware of this risk is part of the desired project outcome.

There is room for improvement in the accuracy of the detection; the code detection is slightly broad hence the pickup of such data as phone numbers, so being able to remedy this by categorising numerical data which is found would be beneficial.

#### 5.4.2. URL Analysis

Every time a URL is part of a text message it is an instant risk in a few different ways. Whether it is receiving one from an unknown number, as part of a push notification, or a link to reset your password it is an inherent risk due to the inherent quality of the URL being that the only way for most people to verify what it is, they must follow it. Doing so can instantly put a device at risk from malicious software being downloaded onto a device or it can mean an unsuspecting victim handing over account access through a cleverly disguised phishing site.

Given the dataset results showing that 46 unique URLs were identified, we can determine there is a roughly 25% chance of a URL being contained within a received text. This statistic is worryingly high, as although through testing none of the randomly selected URLs were flagged as specifically unsafe, each individual URL provides a new risk opportunity. Furthermore, most of the URL types found were password reset links, an extreme risk as it provides direct access to an account if intercepted by a malicious actor. Hence, in terms of providing insight into how secure SMS is, this project tool has shown there is a surprisingly high risk when dealing with messages containing URLs and that users should be very conscious of only following links they are expecting to receive when they can verify the identity of the sender is as expected. This aspect of the program also fit the requirements well as there were no URLs which were missed in identification and the quantitative output from testing shows there is a risk to security.

Accuracy with picking up URLs in the data is on point, every single possible link was found and extracted neatly into an output file ready to be used in the health check analysis, exactly in line with the aims of this aspect of analysis. Using third party resources to allow for this check is extremely useful as providing a thorough report like this requires sophisticated and widely tested algorithms.

#### 5.4.3. Keyword Analysis

One of the trickier aspects of the tool, keyword analysis still worked relatively well in terms of identifying likely dangerous or vulnerable messages. The trouble with accuracy is to be expected when the entire sentiment of the message is not considered, so the context cannot be figured out meaning there are always going to be false positive results. However, it does provide valuable information because out of the 30 dangerous and 26 vulnerable declared messages, users would be able to see the type of behaviours and word usage in genuinely threatening messages.

Using this as a teaching mechanism would be greatly beneficial and work towards project aims of informing potential victims of threatening behaviour patterns meaning these scenarios can be stopped before ever encountered.

The dictionary of words used to identify the dangerous and vulnerable keywords would benefit from further research, due to the current lists not being extensive or exhaustive.

#### 5.4.4. URL Health Check Analysis

The URL Health Check analysis utilising a third-party API is one of the most robust sections of the developed program. Providing a reliable and thorough result output assessing multiple aspects of URL safety means this element is extremely helpful to users. It completely removes the risk of receiving a message containing a URL as it could be copied directly into this tool and a full check can either warn of a potential danger or reassure users.

In terms of what it tells us of the collected data, from the ten sampled URLs we can see six of the ten showed risk ratings of above 50, but none were ruled as overall unsafe. There were no analyses which failed and again six URLs were flagged as suspicious showing that although there was no proven malicious activity picked up there is reason to be cautious with these websites. These statistics indicate that links found in text messages should be treated with precaution. Of the tested links, none showed positive for malware or phishing directly, but this does not mean they are not capable of such behaviours. Relying on AI algorithms and proprietary data [40] means that such an in-depth analysis would not be possible within this project scope, so the service is extremely beneficial in reaching the project goal of presenting quantitative data and showing how secure SMS is.

### 5.5. Recommendations based on results

Due to the aim of this project being proving what SMS is currently being used for and what exactly the everyday users must watch out for regarding safely using the platform, recommendations will be provided based on the results and evaluation as to what users should be aware of and how to remain secure whilst still using SMS.

As shown by the URL collection and count, there is a relatively high chance (around 25%) that they will be encountered in everyday text messages. The associated security risk proved to be surprisingly high with over 50% of a small test sample proving to be deemed suspicious so recommendations should be to click a link only ever when you know the sender and are expecting the message to contain a link. However, the other element of risk is using services which provide a password reset

link over SMS; enabling these services puts your data and accounts at clear risk of compromise and should be avoided whenever possible.

In terms of one-time passcodes, these are proven as a very popular option all round for securing and accessing account logins and setups. Although this is a far better solution than the password reset links for example, a malicious actor with enough information and access could feasibly attempt to access an account and eavesdrop text messages to receive and use the passcode for their own access. Using an authentication app such as Microsoft Authenticator [42] which prevents this kind of risk.

Keyword analysis is a great tool which can provide awareness to users based on words which are commonly associated with malicious behaviours as well as making sure that users are aware of information they should not be spreading over text. As SMS is inherently insecure, sending texts with information such as addresses, contact details, card numbers and more is an extremely risky move and all it takes is for someone to attempt to view your texts to have access to all this data. Any company sending personal data back to you via text is a security risk and when messaging family and friends make sure you are conscious of not sending information you wouldn't want to fall in the hands of a stranger if it can be avoided.

## 6. Future Work

One idea given more time would be to implement sentiment analysis by looking at syntax, but the complexity of this particular issue means it was something that could not be added within the original project scope. This kind of feature would mean gaining a far more developed understanding of message context on a case-by-case basis and in theory at least would improve the accuracy of the analysis by eliminating several false positives.

Another more detailed feature which was theorised is the ability to extract personally identifiable details such as names, addresses and more. Researching into the application of such a task revealed that a complex classification algorithm would be required and in the case of Rui Zhang et al. research, it was shown that a complex Co-guided Neural Network to be used for name recognition [43]. Such development is far too complex and outside the scope of the project in terms of time and subject as neural networks and syntax recognition are large fields of research besides.

Given the opportunity I would have also liked to do more in depth research based on the "vulnerable and "dangerous keyword ideas, where I could spend more time looking for research and documentation based on syntax related to phishing and malicious messages.

Added functionality, which would improve the program's accuracy with identification, is also something that I would have liked to do.

One of the issues discussed earlier is the false positive identification of some OTPs which are phone numbers or specifically formatted dates. Having the ability to recognise specifically formatted numbers and categorise them would greatly improve the accuracy of this specific analysis element and is something I believe could be done quite feasibly with the given resources if there were more time to develop it.

A more complex idea which would be interesting to pursue is collecting a dataset of known malicious messages from various sources to use as example data. Having this as a source would mean building a far more accurate series of keywords to use in the analysis stage as the number of messages used in the program's keyword dictionary is something that could be improved. This would not only reduce false positive results but would mean keeping up with current trends as they inevitably change, adding longevity to the project.

As a part of the test cases for analysis, incomplete URLs were tested to see if they were picked up and flagged. They were, but obviously these URLs are not really of use if they are incomplete because they cannot be health tested by the program nor stored to make a record of it, so adding in a feature which means the tool only picks up complete functional URLs would be a nice addition even though it would not make much of a difference to the overall performance of the tool.

## 7. Conclusions

### 7.1. Aims & objectives Reflection

For this section I will be referring to the Aims & Objectives outlined in chapter 1.3.

#### 7.1.1 Objective 1: Outline Key Security Flaw or Exploits in SMS Services and Applications

The entirety of objective 1, parts a, b, c and d were all related to researching into SMS, the system implementation and how it is used as a service, including research into large-impact incidents regarding SMS security being breached, as well as inherent flaws and available exploits. This was achieved clearly through the thorough research spanning the entirety of chapter 2.

### 7.1.2 Objective 2: Identify and Explain Both the Advantages and Limitations of SMS as a Service, Providing Quantitative Data to Show Relevance to Security

Related works and SMS limitations is covered in detail also in chapter 2, with the SMS dataset being collected as per part 'b'. Admittedly, the collection of the data did not go as planned and manual collection had to be resorted to, meaning the dataset was not as extensive as it could be, therefore this aim was not met to its initially desired specification. Analysis and documentation were covered completely during the main bodies of sections 4 and 5, with real world context being applied, and the results of analysis on the dataset were also presented, meaning the remaining objectives were successfully met.

### 7.1.3 Objective 3: Reflection on SMS Analysis and Results Recommendations

A full reflection and results evaluation was fully covered in this chapter 5, with extensive discussion on SMS use with relevance to security. Full recommendations were also covered in section 5.5, meeting all the sub-objectives in this case.

Overall, all objectives were met at least to some degree, with the majority being met completely and some even exceeded. There are some more particular areas of the project such as dataset collection and quantity of results output where I would have liked to produce more.

## 7.2 What Has Been Learned and Achieved

During the entire project lifetime, I have learned a wide breadth of different things, all of which will help me greatly moving forward, including soft skills and more niche subject specific knowledge.

Due to the process of data collection and programming the data analysis itself being a longer than anticipated process, there was less time to implement some desired additions to the analysis checklist which were part of possible additions to features. A more forgiving time plan would have perhaps allowed for a more linear and thorough development.

Following this I can say time management is a key aspect in creating a project of this scale and sticking to the originally planned schedule as shown in the project Gantt chart was extremely difficult. In reality I found that workflow is not predictable, nor is it linear. Following the projection for time expectation in each task meant some other tasks which ended up taking more time than expected left the project outside of the



planned timescale. Furthermore, dealing with problems and changes in project goals requires a lot of ingenuity; thinking of a different way to approach a particular task whilst staying as aligned as possible to the initial planning is very difficult and sometimes you must make compromises, sacrificing aspects of the project which can most likely be afforded to be sacrificed. Ultimately in experiencing this I learned that prioritising tasks when behind schedule is important, as otherwise it is easy to get snowed under with work that keeps creeping up on you and pressure mounting makes it harder to focus on individual tasks.

During the research section of the project, I learned a lot of subject specific knowledge especially surrounding cellular network infrastructure and the way in which I had to conduct research into related topics gave me a better understanding of the way in which to approach a project of this scale. This can be applied in the future to both professional and personal projects to ensure that they are planned, developed, and concluded to the same high standard, using the same techniques used here.

Another positive I can draw from the experience is the problem-solving element. Trying to create a piece of software using various programming libraries and techniques required a lot of forethought and planning. Sometimes the best decision I could make was just putting time into reading information from official sources such as documentation pages, as getting an understanding of how a particular feature functions can be a lot more valuable than simply looking for the solution to a specific problem scenario. In doing this myself, I was able to adopt this habit into my skillset and it meant I am concluding this project having more practically applicable skills when it comes to programming.

One aspect which I can appreciate in hindsight was not done as well as possible is the application testing. Though my test cases were well thought out, there were at least 2 cases where undocumented fiddling with the tool brought to light issues that I did not even expect to encounter. Making a more thorough plan for testing all possible flaws and features is a crucial aspect to project development and in my next endeavour I will make sure to apply this.

## 8. Reflection

At first the scale of this project was somewhat overwhelming as there was so much to consider, from planning, to research, to development and more. The first thing to draw me to this project was the focus on security, a passionate topic of mine and I was determined to undertake a project which I cared for and for that I am very glad.

Having that level of drive when part of me was feeling project burnout kept me going and determined to achieve a solid final product and ensure the aims and objectives were going to be met. This has taught me that having genuine passion for a topic can take you far and even when you hit patches where it seems so much harder to get work done you are able to find a way to carry on.

For the subject of SMS, it is so much more in depth than I could have imagined. From doing background research and finding how complex the infrastructure system is, to looking into all the weaknesses, and reading over other projects which also try and solve the issue of SMS security in some way. Almost an overload of information, I tried hard to stay focused on the main objective, another difficult issue where I was finding myself checking back with my supervisor for clarification very often in our weekly meetings. Keeping meeting notes helped enormously as the tips and advice I was receiving was invaluable, so being able to load up the notes document and refer to this was reassuring and allowed me to not get carried away with ideas which strayed from the project goals and scope.

Taking this forward, I will make sure to keep placing high value on the advice of those with experience, without this resource I am sure I would have lost my way on the project more than a few times and produced a far less sophisticated result.

Documenting my own progress as I went is also something I will continue to do in future projects, as when looking back on my notes during report writing stages I realised without the notes, I had no idea of specific details or trains of thought or smaller timescales which the note-keeping retained. Producing these documents was another key factor to keeping the project on track and helping with the report writeup, as it is easy to miss sections of implementation if you are not careful.

This plays into the transferrable skill of autonomy, whereby I had to make sure I was responsible for my own work. Keeping track of the time and being the one to make recalculations when I ran out of it is an important factor to enabling large projects like this to be completed. It is also a skill which can be applied to plenty of different scenarios where there is any degree of personal management in play and is something I will make sure to only improve in moving onwards.

Overall, I am pleased with the outcome of the project and can confidently say the planned aims and objectives have been met, creating a strong overall piece of work. From this experience I will take many lessons and learned skills and keep applying them to every task I consider in my future.

## References

- [1] Aaryaman Aashind, Jan 2022. *Texting Statistics UK [2022 Edition]*. Available: <https://cybercrew.uk/blog/texting-statistics-uk/> [accessed 03/02/22]
- [2] The Open University, 2014. *Text Messaging Usage Statistics*. Available: <https://www.openuniversity.edu/news/news/2014-text-messaging-usage-statistics> [accessed 03/02/22]
- [3] Wickr CTO Cristopher Howell, Nov 2019. *How SMS Works and Why You Shouldn't Use It Anymore*. Available: <https://www.popularmechanics.com/technology/security/a29789903/what-is-sms/> [accessed 03/02/22]
- [4] Postscript. November 2020. *Text Messaging Marketing Statistics 2020*. Available: <https://postscript.io/blog/text-messaging-statistics> [accessed 05/02/22]
- [5] E. Wilde et al., January 2010. *URI Scheme for Global System for Mobile Communications (GSM) Short Message Service (SMS)*. Available: <https://www.ietf.org/rfc/rfc5724.txt#:~:text=GSM%20SMS%20messages%20are%20alphanumeric,CHAR%5D>
- [6] Bradley Reaves et al., December 2018. *Characterizing the Security of the SMS Ecosystem with Public Gateways*. Available: <https://dl.acm.org/doi/10.1145/3268932>
- [7] Clickatell, Unknown. *6 Industries using SMS Messaging*. Available: <https://www.clickatell.com/articles/digital-marketing/6-industries-using-sms-messaging/> [accessed 10/02/22]
- [8] Ikechukwu Ibekwe, Salem Aljareh. June 2012. *SMS Security: Highlighting its vulnerabilities & techniques towards developing a solution*. Available:

<https://researchportal.port.ac.uk/en/publications/sms-security-highlighting-its-vulnerabilities-amp-techniques-towa>

[9] UnboundSecurity. August 17th, 2021. *Why SMS OTP Is Not Enough Security for Authentication*. Available: <https://www.unboundsecurity.com/blog/sms-based-otp-is-just-not-good-enough/> [accessed 01/03/22]

[10] Mohammed Shafiul Alam Khan, Chris J Mitchell. September 2016. *Retrofitting Mutual Authentication to GSM Using RAND Hijacking*. Available: [https://www.researchgate.net/publication/308193943\\_Retrofitting\\_Mutual\\_Authentication\\_to\\_GSM\\_Using\\_RAND\\_Hijacking](https://www.researchgate.net/publication/308193943_Retrofitting_Mutual_Authentication_to_GSM_Using_RAND_Hijacking)

[11] Neil Croft. October 2012. *On Forensics: A Silent SMS Attack*. Available: <https://ieeexplore.ieee.org/document/6320454>

[12] Norton LifeLock Employee. January 2018. *What Is Smishing?*. Available: <https://us.norton.com/internetsecurity-emerging-threats-what-is-smishing.html> [accessed 10/02/22]

[13] Lindsey O'Donnell. February 19th, 2020. *SMS Attack Spreads EMOTET, Steals Bank Credentials*. Available: <https://threatpost.com/sms-attack-spreads-emotet-bank-credentials/153015/> [accessed 15/02/22]

[14] Sandhya Mishra. July 2021. *DSmishSMS - A System to Detect Smishing*. Available: [https://www.researchgate.net/publication/353531607\\_DSmishSMS-A\\_System\\_to\\_Detect\\_Smishing\\_SMS](https://www.researchgate.net/publication/353531607_DSmishSMS-A_System_to_Detect_Smishing_SMS)

[15] Ali M. Sagheer. February 2015. *SMS Security For Smartphones*. Available: [https://www.researchgate.net/publication/283594319\\_SMS\\_Security\\_for\\_Smartphone#read](https://www.researchgate.net/publication/283594319_SMS_Security_for_Smartphone#read)

[16] Rūta Rimkienė. December 2020. *What is AES encryption and how does it work?* Available: <https://cybernews.com/resources/what-is-aes-encryption/> [accessed 14/03/22]

[17] GeeksForGeeks. December 2021. *What is RC4 Encryption?* Available: <https://www.geeksforgeeks.org/what-is-rc4-encryption/> [accessed 14/03/22]

[18] Ikechukwu Ibekwe, Salem Aljareh. June 2012. *SMS Security: Highlighting its vulnerabilities & techniques towards developing a solution*. Available: <https://researchportal.port.ac.uk/en/publications/sms-security-highlighting-its-vulnerabilities-amp-techniques-towa>

[19] Bradley Reeves, Logan Blue et al., July 2016. *Detecting SMS Spam in the Age of Legitimate Bulk Messaging*. Available: <https://dl.acm.org/doi/10.1145/2939918.2939937>

[20] M. Hassan Shirali-Shahreza et al., August 2008. *An Anti-SMS-Spam Using CAPTCHA*. Available: <https://ieeexplore.ieee.org/abstract/document/4609698>

- [21] Tim Parsons. May 2019. *When to Use Waterfall vs. Agile*. Available: <https://www.macadamian.com/learn/when-to-use-waterfall-vs-agile/> [accessed 02/04/22]
- [22] Gaurav Kumar, Pradeep Kumar Bhatia. August 2012. *Impact of Agile Methodology on Software Development Process*. Available: [https://www.researchgate.net/profile/Gaurav-Kumar-175/publication/255707851\\_Impact\\_of\\_Agile\\_Methodology\\_on\\_Software\\_Development\\_Process/links/00b49520489442e12d000000/Impact-of-Agile-Methodology-on-Software-Development-Process.pdf](https://www.researchgate.net/profile/Gaurav-Kumar-175/publication/255707851_Impact_of_Agile_Methodology_on_Software_Development_Process/links/00b49520489442e12d000000/Impact-of-Agile-Methodology-on-Software-Development-Process.pdf)
- [23] Raimo Streefkerk. April 2019. *Qualitative vs. Quantitative Research | Differences, Examples & Methods*. Available: <https://www.scribbr.com/methodology/qualitative-quantitative-research/#:~:text=and%20qualitative%20methods%3F-Quantitative%20research%20deals%20with%20numbers%20and%20statistics%2C%20while%20qualitative%20research,and%20experiences%20in%20more%20detail.> [accessed 04/04/22]
- [24] Octoparse. March 2022. *V8.5.2*. Available: <https://www.octoparse.com/download/windows> [accessed 12/04/22]
- [25] Parsehub. Available: <https://www.parsehub.com/> [accessed 12/04/22]
- [26] Zyte, Scrapy. March 2022. *V2.6.2*. Available: <https://scrapy.org/>
- [27] JT28, Stackoverflow. August 2016. *How to solve 403 error in scrapy*. Available: <https://stackoverflow.com/questions/39202058/how-to-solve-403-error-in-scrapy> [accessed 08/04/22]
- [28] A Bogdanchikov et al. 2013. *Python to learn programming*. Available: <https://iopscience.iop.org/article/10.1088/1742-6596/423/1/012027/meta>
- [29] Tkinter. *v3.10.4*. Available: <https://docs.python.org/3/library/tkinter.html> [accessed 02/05/22]
- [30] Re - Regular Expression Operations. *v3.10.4*. Available: <https://docs.python.org/3/library/re.html> [accessed 19/04/22]
- [31] Requests. *v2.27.1*. Available: <https://docs.python-requests.org/en/latest/> [accessed 23/04/22]
- [32] Urllib. *V3.10*. Available: <https://docs.python.org/3/library/urllib.html> [accessed 23/04/22]
- [33] Ryan Thelin. September 2020. *How to use Object-Oriented Programming in Python*. Available: <https://www.educative.io/blog/how-to-use-oop-in-python> [27/04/22]

- [34] Tom Fellmann et al. March 2007. *A command line interface versus a graphical user interface in coding VR systems*. Available: [https://www.researchgate.net/publication/234818436\\_A\\_command\\_line\\_interface\\_versus\\_a\\_graphical\\_user\\_interface\\_in\\_coding\\_VR\\_systems](https://www.researchgate.net/publication/234818436_A_command_line_interface_versus_a_graphical_user_interface_in_coding_VR_systems)
- [35] Jakob Nielsen. April 1994. *10 Usability Heuristics for User Interface Design*. Available: <https://www.nngroup.com/articles/ten-usability-heuristics/>
- [36] Geeksforgeeks. August 2021. *Python Tkinter - Toplevel Widget*. Available: <https://www.geeksforgeeks.org/python-tkinter-toplevel-widget/> [accessed 04/04/22]
- [37] StackOverflow - Paul Stephenson. January 2009. *Using global variables in a function*. Available: <https://stackoverflow.com/questions/423379/using-global-variables-in-a-function> [accessed 27/04/22]
- [38] StackOverflow - Thomas Wagenaar. February 2015. *How to handle FileNotFoundError when "try... except IOError" does not catch it?* Available: <https://stackoverflow.com/questions/28633555/how-to-handle-filenotfounderror-when-try-except-ioerror-does-not-catch-it> [accessed 06/05/22]
- [39] Gaston L'Huillier et al. May 2010. *Latent semantic analysis and keyword extraction for phishing classification*. Available: [https://ieeexplore.ieee.org/abstract/document/5484762?casa\\_token=BJClmdSCD6MAAAA:A:9X-bR7D7S4FDhynrKIXFSSLN2-uc8DEEN53NXYN342O5jmf7pbcm6wZvbukHgurn9kpEUHf0EA](https://ieeexplore.ieee.org/abstract/document/5484762?casa_token=BJClmdSCD6MAAAA:A:9X-bR7D7S4FDhynrKIXFSSLN2-uc8DEEN53NXYN342O5jmf7pbcm6wZvbukHgurn9kpEUHf0EA) [accessed 20/04/22]
- [40] Ipqualityscore - [used API]. Available: <https://www.ipqualityscore.com/threat-feeds/malicious-url-scanner> [accessed 24/04/22]
- [41] Artists Against 419. Updated – May 2022. *Fake Sites Database*. Available: <https://db.aa419.org/fakebankslist.php> [accessed 21/05/22]
- [42] Microsoft Authenticator – Available now on Google Play and App Store. <https://www.microsoft.com/en-us/security/mobile-authenticator-app> [accessed 24/05/22]
- [43] Rui Zhang et al. March 2021. *NameRec\*: Highly Accurate and Fine-grained Person Name Recognition*. Available: [https://www.semanticscholar.org/paper/NameRec\\*%3A-Highly-Accurate-and-Fine-grained-Person-Zhang-Dai/da25df7fc6480fa3f17f9f1624fa25cb2461edd3](https://www.semanticscholar.org/paper/NameRec*%3A-Highly-Accurate-and-Fine-grained-Person-Zhang-Dai/da25df7fc6480fa3f17f9f1624fa25cb2461edd3)