Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Identifying Clusters of Reused Cryptographic Keys

Rahul Seth

B.A.I Computer and Electronics

Final Year Project

April 2022

Supervisor: Dr. Stephen Farrell

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have completed the Online Tutorial in avoiding plagiarism 'Ready, Steady, Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

_____ April 23, 2022

Rahul Seth

Abstract

Internet Scanning is used for a variety of purposes. While some may use it to detect and investigate flaws and vulnerabilities in a network, others may exploit them. This project aims to migrate, refactor, optimise the existing program, and survey long term cryptographic keys for web, mail and SSH protocols in the IPv4 address space. The target population for these scans are hosts that accept connections on TCP port 25, i.e. hosts that offer mail services. The hosts identified as port 25 listeners are further scanned to get their SSH and TLS session data to check for key reuse for Secure Shell Protocol and Transport Layer Security protocols. Finally, the project investigates some of the causes behind this key reuse and builds directly from Dr Farrell's research in this domain.

Mismanaged key reuse can create vulnerabilities in a network that can go undetected and leave entities open to attacks such as the man-in-the-middle. The program to survey these keys was last run in 2018, and the project explores how key reuse has evolved since then. Internet-wide scanning is a well-researched domain, but this project performs local region scans with only port 25 listeners, hoping that small scale scans could identify vulnerabilities better than internet-scale deployments. The project also entailed code migration, refactoring and optimisation to decrease the run time and memory consumption by identifying bottlenecks in the program and exploring alternate solutions with new developments in the technology landscape since 2018.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Stephen Farrell. Without him and his constant guidance and contributions, this project would not be possible. Thank you for your time and patience. I have gained a lot more than just knowledge from you.

Next, my parents and my sister, without whom I would not be here. You guys are everything.

Last but not least, my friends in Dublin and India. Thank you for having faith in me when I did not believe in myself.

List of Acronyms

- **AS:** Autonomous System
- **ACK:** Acknowledgement
- **API:** Application Programming Interface
- **CIDR:** Classless Inter-Domain Routing
- **DNS:** Domain Name System
- **HTTP/ (S):** Hypertext Transfer Protocol (Secure)
- **IP:** Internet Protocol
- **IMAP/ (S):** Internet Message Access Protocol (Secure)
- **POP3:** Post Office Protocol Version 3
- **PKI:** Public Key Infrastructure
- **RST:** Reset
- **SSH:** Secure Shell
- **SYN:** Synchronization
- **SMTP/ (S):** Simple Mail Transfer Protocol (Secure)
- **TCP:** Transmission Control Protocol
- **TLS:** Transport Layer Security
- **VM:** Virtual Machine
- **JSON:** Javascript Object Notation

List of Figures

List of Tables

Listings

Contents

Chapter 1

Introduction

This chapter introduces the reader to the motivation behind this project and provides a brief outline of the research and personal objectives to be accomplished. It also gives an overview of the structure of this report.

1.1 Motivation

Since the inception of the Internet, it has been in a constant state of evolution. While it provides users easy access to information and services, it also includes several risks. The rapid development of the Internet has revolutionised how humans communicate with each other and has become a natural extension of our lives. However, this rapid evolution leaves an opening for people to exploit the weakness in the infrastructure for their gain. Data from Internet-Wide Scans can be used to identify devices and services exposed to the Internet. In addition, the data can be analysed to determine devices or services prone to vulnerabilities or weaknesses. The risks associated with those vulnerabilities and flaws cannot be underestimated as they can lead to severe ramifications not just for the entity involved, but also for the users that use that service.

In 2014, the Yahoo Data Breach, which is known to be one of the most significant data breaches, occurred. Due to improper input validation systems in place by Yahoo, the attackers were able to take any identity of their choice on the network. They did this by exploiting a weakness in the user creation and identification process [44].

The project implements an Internet-Wide Surveying tool for Ireland, but can be used for any country to check for long-term key reuse and builds on existing research as outlined in Chapter 3.

1.2 Objectives

This section sets out the research questions and objectives for this dissertation. It also outlines the personal goals set out.

1.2.1 Research Objectives

- Recreate Dr. Farrell's existing research in 2018 on surveying cryptographic keys.
- Migrate the current program to Python3 from Python2.
- Refactor code to increase readability and decrease complexity.
- Understand the development in key reuse as compared to previous scans.
- Research existing tools and APIs for accurate geolocation IP data to carry out the scanning process.
- Modify existing code for the program to work with the latest tools and APIs.
- Add DNS over TLS (port 853) in the scanning process.

1.2.2 Personal Objectives

- Develop knowledge on the Public Key Infrastructure (PKI) and how it is deployed to manage internet security for different protocols.
- Develop knowledge of internet protocols like SSH, TLS, SMTP, IMAP, SSH and more to understand the implications of key reuse.
- Familiarise with internet scanning and surveying tools like ZMap and ZGrab to get data needed for analysis.
- Learn to code with industry-standard Python practices and use memory-efficient methods for data storage, retrieval, and access.
- Ability to multitask and efficiently manage time to complete the research, coursework, and other personal work like a job search.

1.3 Thesis Overview

A brief outline of each chapter of this thesis is presented below:

- **Chapter 2: Background -** This chapter presents a detailed theoretical background of the research project.
- **Chapter 3: Related Work -** This chapter provides a summary of existing and relevant scientific literature for this project.
- **Chapter 4: Methodology -** This chapter describes the methods used for the investigation of the problem and the technologies used.
- **Chapter 5: Results -** The chapter presents the results obtained after the network scan.
- **Chapter 6: Conclusions and Future Work -** This chapter concludes the project with a summary of the research done and its validity, along with possible research that can be carried out on top of this.
- **Bibliography and Appendix -** These sections comprise a list of all the papers, articles, and books referred to while writing this thesis, and the appendix contains some supplementary information.

Chapter 2

Background

This chapter presents a detailed discussion of the important theoretical concepts on the subject to provide an understanding of the project undertaken.

2.1 Internet Scanning

Internet Scanning is the process of using network scanning techniques to conduct scans on a large scale. Network Scanning is defined as the process of identifying hosts on a network by using features in the network protocol to ping devices or hosts and of analysing information based on the data received by those pings. The basic concept of network scanning is to identify all hosts connected to a network and map them to their IP addresses. This process is achieved by sending packets to addresses and in turn discover what is on the network based on the data received. All the active hosts in the network will respond to this ping while the inactive hosts will have no response. The feedback from this scan gives information about how the hosts behave with internal and external components of a network. Another technique called "Port Scanning" can be used to gather information about open ports that can receive or send information for the identified hosts in the network.
Performing network scans can have both good and bad implications. While some entities use network and port scanning to identify weaknesses, some may use it to exploit the weakness in the network to gain access to information they are not privy to [30].

2.1.1 MaxMind Geolocation IP Databases and Services

Maxmind provides services packaged in APIs and databases that provide accurate IP intelligence data. The web services give the most accurate IP geolocation data and can be accessed through APIs in almost every programming language. The GeoIP2 databases provide in-depth information for IPv4 network blocks and are locally maintained for high volume, fast lookup purposes and allow unlimited use. Since the GeoIP2 databases are now commercial, the project uses the free version called GeoLite2 databases that are slightly less accurate than the former and are updated weekly. The databases include information about the entire IPv4 address space for all countries. These APIs and databases are used for a variety of purposes like detecting network vulnerabilities and fraud detections [21]. The program for this project required a dataset containing IPv4 addresses and their associated country name and country codes. Multiple Maxmind datasets were used to gather this information, and Python was used to generate the dataset required by combining information from different datasets. To get access to Maxmind, an account was made through their website in order to obtain the license key required to use the services mentioned above.

2.1.2 ZMap and ZGrab

ZMap is an open-source fast single packet network scanner that is capable of scanning the entire IPv4 address space in under 45 minutes from a single standard machine. It provides the user with various probe modules including TCP SYN scans, ICMP, DNS queries, UPnP BACNET and can also send UDP probes. Compared to other tools in the market ZMap achieves a better performance due to its architecture which is optimised for carrying out internet-wide surveys [5]. This project makes use of the SYN module for identifying open ports. A TCP SYN scan involves the sending of an SYN packet to open a connection with a host on a specific port. If there is an SYN/ACK response from the host that indicates there is an open TCP/IP port. In case there is an RST response instead of an ACK, that indicates the particular port is closed [13].

ZGrab is ZMap's sister project and an open-source fast application-layer network scanner designed to perform extensive internet-wide surveys. It works in combination with ZMap, but can also be used independently. It provides detailed information about the network handshakes and captures most of the meta-data during a TLS negotiation like the TLS certificates and banner information. It is built using Golang and is capable of carrying out the scanning process for all standard protocols like HTTP, SSH, IMAP and more. It provides the output in JSON format for each IP scanned for the selected protocol [6].

2.1.3 Ethical Considerations

As stated above, network and port scanning techniques are carried out for various purposes. While network administrators or academics use these techniques to identify vulnerabilities or weaknesses, cyber attackers can use the same techniques to gain access to systems they are not privy to. Since the project requires carrying out active scans to understand the extent of public key reuse, it is important to consider whether there are any ethical implications for the hosts we are scanning. To carry out the scanning process, Durmeric et al. [5] have summarised some best practices one needs to consider as it is next to impossible to get permissions in advance from all hosts that are being scanned and assessed. Some of these practices include:

- Ensuring scans will not overwhelm the network.
- Providing the nature of the scans in the form of webpages and DNS entries.
- Having a clear scope of the project and explaining the purposes of the scans.
- Limiting scanning when possible.
- Having a simple opt-out method.

In considering the points mentioned above, Dr. Farrell carried out these scans using his Virtual Private Server and had a DNS TXT record that indicated the nature of the scans. Also, the project scans hosts that are mail servers, and hence, it is less likely to come across sensitive information since individuals do not run most mail servers. The scan rate for the ZMap and ZGrab tools was limited to not cause any disruptions to the active services in the network. The default blocklists that were used and provided by the ZMap included: local, reserved, and multicast IPv4 addresses.
Another factor that was considered besides the ones stated above was the secure storage and reporting of the data collected. All data was securely stored locally on my machine, and the analysis was also carried out on the same machine to report the key reuse scenarios. All IP addresses were anonymised in this report, and no domain names were released.

2.2 Public Key Infrastructure

The Public Key Infrastructure (PKI) is a framework that comprises of set of policies, guidelines, and technologies that different enterprises, vendors and other entities can use to establish and maintain authentication and confidentiality while communicating over the internet. The PKI works on the core concept of public-key cryptography or known as asymmetric encryption. It comprises of a public and

private key or more commonly referred to as a key-pair. Consider, for example, Bob wants to send a message to Alice securely. Both of them have knowledge about their respective key-pair. The following figure demonstrates how Bob would send a message to Alice using asymmetric encryption.

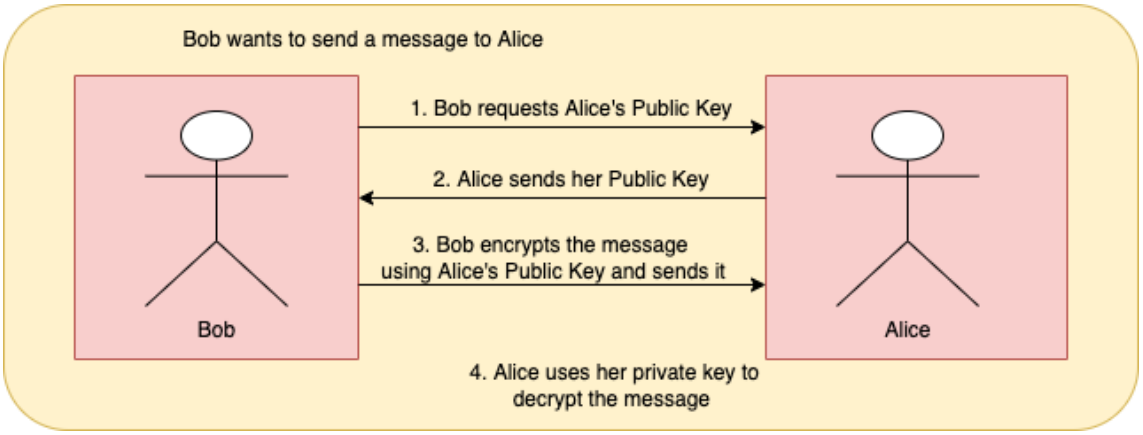

Figure 2.1: Asymetric Encryption

A key over here is defined as a randomly generated sequence of bits. The public and private keys are closely related by some mathematical operations. But the question that arises here is how Bob knows that it was Alice who sent her public key. How can Bob authenticate the identity of Alice? This is where Digital Certificates play a crucial role, as they help in associating a public key with a person or an entity that allows authentication. These certificates are issued by Certification Authorities or commonly known as CAs. They are usually a third-party organisation (Eg: Digicert) responsible for issuing, revoking, and distributing certificates. The CA is trusted by all parties involved in the PKI, which would be Bob and Alice.

Now Bob can ask the CA for Alice's certificate, which contains information about Alice's public key. Since Bob trusts the CA and the CA is vouching for Alice, Alice's identity could be authenticated. The PKI is deployed in a variety of environments over the internet to secure them, and some instances are web browsers, emails, file security etc [25].

2.3 Transport Layer Security

Transport Layer Security (TLS) is a cryptographic protocol used to transfer data between applications over the internet securely. In today's day and age, TLS is one of the widely adopted protocols as it is majorly used in web browsers to ensure a secure session has been established. It can also be used for the safe transfer of data for different applications such as email, file transfers, voice-over-IP, and DNS. TLS does not secure data on the end systems but is used to facilitate secure data transfer from one point to another over the internet. As a result, no attacker can tamper or eavesdrop while the data is

in transit. It is usually implemented over protocols like TCP/IP or UDP layer to secure application layer protocols like HTTP, IMAP, POP3, SMTP, and more. TLS was built on the Secure Socket Layer (SSL) protocol and was designed to be its replacement. It is a multilayered protocol that consists of:

- **Handshake Protocol:** Authenticates the parties involved and negotiates an encryption algorithm and other parameters.
- **Record Protocol:** Ensures data is not tampered with when in transit using parameters negotiated during the handshake protocol.

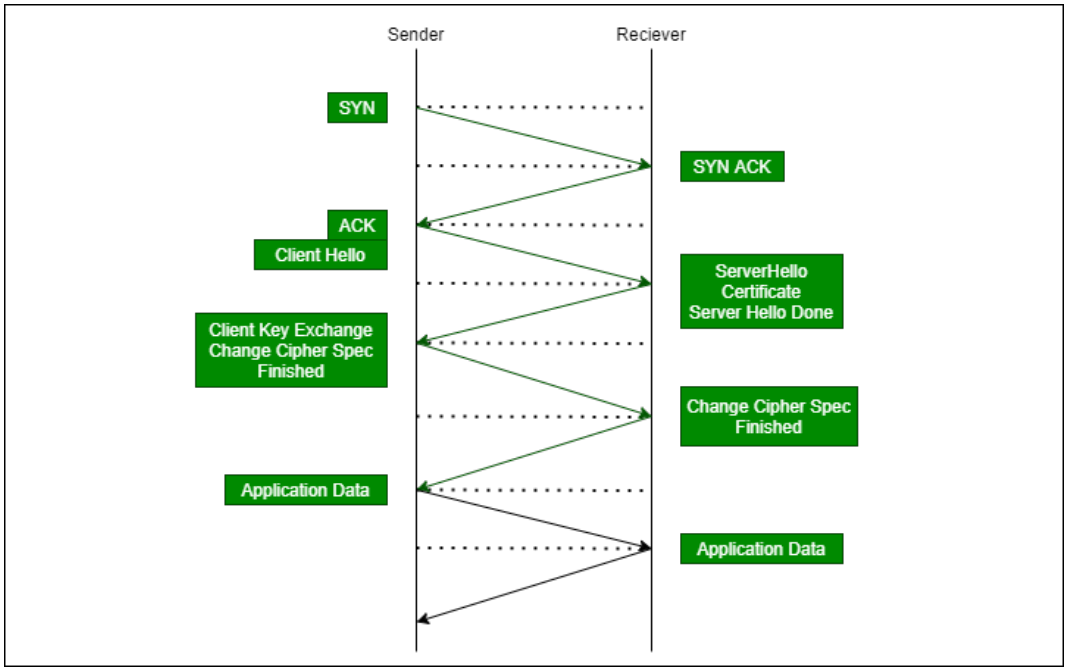

Figure 2.2: TLS Handshake Process

[26]

Since the inception of TLS, it has been deployed in most of the web services on the internet because it protects sensitive information such as passwords, card details, emails, online chats, browsing habits, etc. Since most websites work on a client-server model deploying TLS between the two endpoints protects sensitive information from attackers. TLS also makes use of the technique of asymmetric cryptography which involves a key pair. In this scenario, the public key is used to encrypt the data by the sender and the receiver uses their private key to decrypt the data [35].

2.3.1 Implicit TLS v/s Opportunistic TLS

Around the time when TLS was invented, plain text protocols like SMTP, POP3, and IMAP were already deployed heavily over the internet. While many services supported the usage of the *START-TLS* command to upgrade the connection on the plain text ports, if a client did not support the same

information would be transmitted in plain text before encryption was standardised. *STARTTLS* or Opportunistic TLS was used to upgrade plain text connections to a secure one. Here the connection is upgraded after establishing the initial connection.

To normalise encryption over plain text protocols, new ports were decided upon by IANA. The difference here was that a TLS connection was immediately negotiated between the server and the client. If a server or client did not support TLS and the connection was not established, no information would be exchanged between the two. This is known as Implicit TLS. The use of Implicit TLS is preferred over the former to encourage consistency in how TLS is used [27]. The table below shows the ports used for each protocol using Implicit or Opportunistic TLS as decided upon by IANA [3].

Protocol	Standard Port - No encryption	Implicit TLS Port	Opportunistic TLS Port
SMTP	25	587	25
POP3	110	995	110
IMAP	143	993	143
HTTP	80	443	-

Table 2.1: Implicit TLS vs Opportunistic TLS Ports

2.3.2 TLS Certificates

TLS certificates verify the ownership of a public key and are essential to secure connections and transactions over the internet. They are usually issued by some Certification Authority (CA) by signing the certificates indicating that the CA has verified the ownership. Whenever a user tries to connect to a server, the server sends them a certificate, and then the user verifies the server's certificate using the CA certificate present on the user's machine to establish a TLS connection. The certificates usually contain the following fields of information [2]:

- Subject Domain Name
- Subject Organisation
- Issuing CA
- Alternative Subject Name
- Date of Issue
- Expiry Date
- Public Key

- Digital Signature by the issuing CA

2.3.3 TLS Cipher Suites

A cipher suite is defined as a set of cryptographic algorithms that are used by TLS to encrypt the information. It provides crucial information about securing data when using different network protocols like SMTP, HTTPS, POP3, etc. A cipher will dictate what algorithm is best suitable to make a secure and reliable connection to the server. A cipher suits provides the following information to a server:

- **Key Exchange Algorithm:** Data over the internet is encrypted using a key. This provides the client and server with which algorithm to use for encryption/decryption of data.
- **Authentication Algorithm:** The server needs to verify the client's identity before sending or receiving any data. This field specifies that algorithm.
- **Bulk Data Encryption:** This is to ensure the secure transfer of data.
- **Message Authentication Code Algorithm:** A MAC algorithm is sent along the with data to verify the contents of the data.

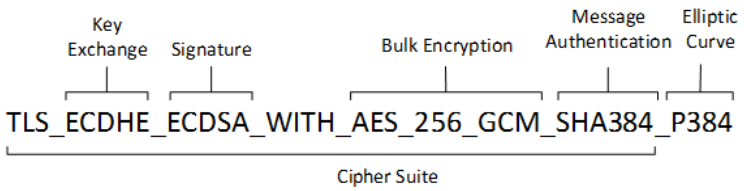

Figure 2.3: TLS Cipher Suite

[24]

2.3.4 Fingerprint SHA-256

The concept of a key pair was introduced above. A fingerprint is defined as a short sequence of bytes used to identify a longer public key and is generated by applying a hash function on a public key. SHA stands for Secure Hash Algorithm, which is used to shorten data into a minor sequence. The resulting output cannot be cracked unless a brute force attack is used, and this is where hashing differs from encryption. SHA256 is a popular cryptographic algorithm that, if applied to a number consisting "n" bits, will return a 256-bit value. It is widely used in digital certificates and signatures.

2.4 Application Layer Protocols

This section describes the ports we scan and the protocols associated with the ports.

Port	Protocol
22	SSH
25	SMTP
110	POP3
143	IMAP
443	HTTPS
587	SMTP Submit
993	IMAPS

Table 2.2: Ports Scanned

2.4.1 Secure Shell Protocol

The Secure Shell (SSH) is a network communications protocol that enables two computers to communicate and share data remotely. Communication between the two machines is encrypted, meaning this protocol can be used over an insecure network to make it secure. SSH consists of three layers:

- **Transport Layer:** Establishes safe connections between the server and the client for communications after the authentication process has been validated. Oversees data encryption, decryption, and integrity and provides caching and compression if needed.
- **Authentication layer:** Conducts the authentication process, i.e. verifies the identity of the user.
- **Connection Layer:** Manages the communication between the two machines once authentication is completed and handles the opening and closing of each session.

SSH requires a login from the user to start performing operations on the remote machine and can be used for the safe transfer of data. It works on a client-server model, i.e. the client will initiate the process by pinging the server, and in turn, the server responds to the client prompting them to finish the authentication process. The SSH server listens on some TCP/IP ports designated for SSH, and usually, TCP/IP port 22 is reserved for SSH servers. Clients contact the server on this port to start the connection process [19].

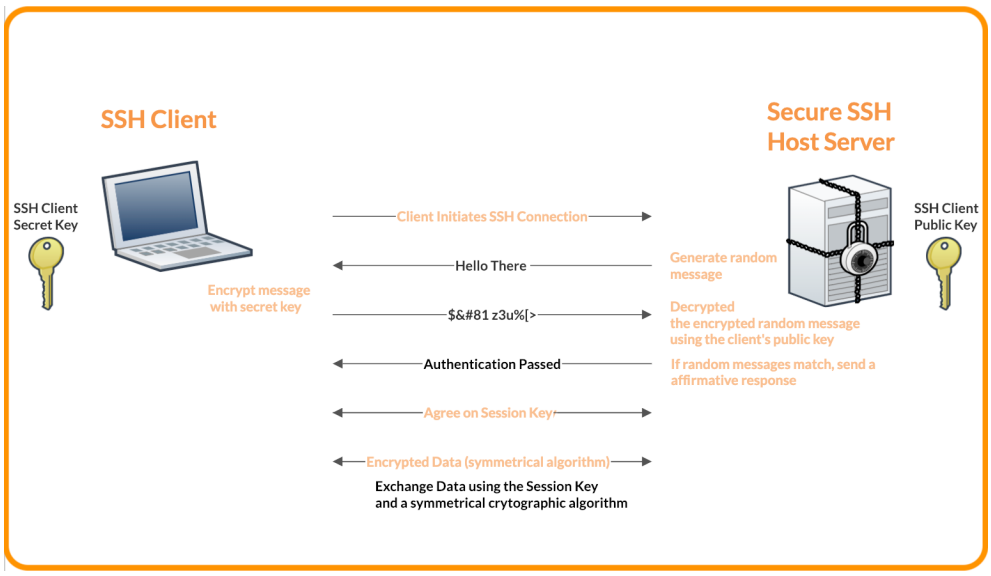

Figure 2.4: SSH Process

[40]

SSH uses asymmetric cryptography similar to the PKI for the authentication process between the client and the server and uses symmetric encryption and other hashing algorithms to encrypt the data transfer between the client and the server, ensuring privacy and integrity. Each SSH server should have at least one host key to ensure that the client communicates with the correct server during the key exchange. Ideally, each host should have a unique key, as key sharing between hosts can leave the host susceptible to man-in-the-middle attacks. However, key sharing may be acceptable and even practical (for instance, multiple hosts sharing keys but all under one entity). The Secure Shell Protocol is widely adopted and used for various purposes by individuals and Corporations. Some use cases are remote access to machines, port forwarding, virtual private networks, and many more.

2.4.2 SMTP/ (S)

Simple Mail Transfer Protocol is used to transfer mail reliably and efficiently. A two-way connection is established with an SMTP server when an SMTP client wants to transmit a message. The main objective of this protocol is to transfer mail messages to an SMTP server or to report failure to incase it fails to do so. Traditional SMTP operates over assigned port 25 and usually does not provide encryption, meaning that the client and server communicate over the internet in plain text [16]. This can be considered a significant flaw as all communications are susceptible to eavesdropping or man-in-the-middle attacks while emails are in transit. SMTP over TLS or SMTPS was introduced to encrypt these communications. By using SMTP over TLS, one is wrapping SMTP commands inside a TLS connection. Usually, port 587 is used for SMTPS compared to port 25 for SMTP to distinguish between the two.

With the use of SMTP over TLS, the client waits for the *STARTTLS* keyword from the SMTP server. After that, the TLS handshake protocol is completed. After the handshake is completed, the client and the server decide whether to continue ahead with the session based on the current privacy achieved. Some client-servers might choose to continue ahead even if no TLS authentication was attained, as traditionally SMTP operates without any encryption, while others may only decide to continue with the session based on specific authentication and privacy achieved.. [15].

2.4.3 POP3 / (S)

Post Office Protocol Version 3 or POP3 is a standard mail protocol used by mail servers and their clients to receive emails from a remote server and send it to a local client. It is a client-server protocol in which email is received and stored on a mail server, and a recipient of an email client can download emails from that server which enables the client to view the email offline. POP3 is embedded into most email clients. Once the email is on the client, POP3 can be configured to delete the email from the server or save the email for a specific period, allowing clients to download the mail as many times. The standard port assigned to POP3 is port 110, but usually, communication is over plain text on this port, but a secure connection can be established using the *STARTTLS* command on this port. In case the client wants to connect securely, POP3 over TLS is used that is assigned to port 995 [36].

2.4.4 IMAP/ (S)

The Internet Message Protocol or IMAP is a standard mail retrieval protocol used to download mail messages by email clients. It enables users with control over mail boxes and enables with to organise them as per requirements. The port assigned to the IMAP protocol is 143, but it does not support encryption over that port. To secure it the *"STARTTLS"* command can be used. Alternatively, port 993 can be used that supports IMAP or TLS (IMAPS) that establishes a TLS connection immediately. [22].

2.4.5 HTTPS

HTTPS or HTTP over TLS is an alternative to simple HTTP over TCP. The difference here is that the HTTP client should also act as a TLS client. The client should initiate the connection, but before making an HTTP request, it should establish a TLS session by initiating the TLS handshake protocol. Once the handshake is completed, the client can start making HTTP requests to the server. All data exchanged between the client and the server is sent as TLS application data. To distinguish between

HTTP over TCP and HTTP over TLS, both protocols have been assigned different port numbers. HTTP operates over port 80 while HTTPS operates over port 443 [34].

2.5 Technology Evolution

The development of the surveying program was previously done in 2017/18 using Python2. However, Python3 has gained popularity and has been widely adopted by corporations, individuals, and others. There are few notable differences between the two, and specific Python3 versions provide better performance than Python2 versions. Since January 1, 2020, support for Python2 has been discontinued. That means there will be no further improvements for Python2 even if a significant bug or security flaw is found [42]. Popular libraries like Pandas, NumPy, and many more have officially stopped supporting Python2 versions, which makes it even more critical to migrate the current code to Python3 [1].
The previous program used ZGrab1 to run the scans back in 2017/18, and since then, ZGrab2 has been released, which has depreciated the last version. It contains a revamped framework that has simplified the tools' usage and allows individuals to add custom protocols over various ports by building them on their own using Golang. It has integration tests available which can help the development process and can be efficiently run using Docker [6, 23].
There were also some minor changes to how Maxmind ships the data needed for the program to work. Additional scripts and modifications to current ones were carried out to get the countrywide IP data to carry out the scans.

2.6 Code Refactoring

Code refactoring can be defined as the restructuring of code to improve code readability, reduce complexity, and improve the maintainability and efficiency of the program. The refactoring process should contribute to the above factors, but the program's functionality should not be compromised. Refactoring enables developers to gain an in-depth understanding of the program and enables them to expand the program quickly by integrating more features efficiently and reducing the amount of technical debt.

Technical debt, also known as code debt, refers to when development is rushed or when code delivery is prioritised over writing quality code. To ensure code refactoring has yielded benefits, one needs to define a few metrics like *"% of duplicate code"* [29]. Techniques like unit testing and functionality tests need to be performed regularly in order to ensure the refactoring process is beneficial [18]. Refactoring a program should be started by analysing the current code and checking whether refactoring is required. There are a few standard practices defined in the software engineering community to carry out the process of code refactoring. Some of the practices used in the project include:

- **Inline:** Simplifying code by eliminating unnecessary elements.
- **Extract:** Break down code into smaller fragments and then move these fragments into a different method.
- **Abstraction:** Reduce the amount of duplicate code by identifying points of similarity. [11]

Chapter 3

Related Work

Henniger et al. [14] perform Internet-wide scans of the entire IPv4 address space for hosts that listen on port 22 (SSH) or port 443 (HTTPS) using the NMap tool. NMap is an open-source network scanner that is used for network audits. They perform TLS or SSH handshakes on hosts that listen on port 443 or port 22, respectively, and gather most of the data that might help discover weak keys. For TLS, they capture most of the metadata like certificate information and other X.509 certificate fields. For SSH, they collect the host keys using a simple client developed in C. All of the scans and data processing were carried out using Amazon's EC2 service, which provided them with an infrastructure to collect and analyse data efficiently. After processing the information, they identify some patterns between a host that shares keys and try to pinpoint some of the causes behind these. In the TLS protocol, the server sends its public key in a TLS certificate during the handshake stage. This key is used to provide a signature during the handshake stage or can be used to encrypt session details depending on whether RSA or DSA encryption algorithm is negotiated between the client and the server. Similarly, in the SSH protocol, the host key allows the server to authenticate itself to the client by providing a signature during the handshake stage. In both these protocols, if the private keys are known by an attacker and depending on the type of the encryption scheme (RSA or DSA), an attacker can perform different attacks like man-in-the-middle or decrypt the message containing the session key and, in turn, use that to decrypt the entire session.

[14] further computes the private keys using the weak public keys by exploiting the weakness of RSA and DSA algorithms when used with insufficient randomness. As a result, they can compute private keys for 0.50% of the total TLS hosts and 1.06% of the SSH hosts found from their respective public keys since the key pair is related mathematically. They scanned 12.8 million TLS hosts and only obtained 5.8 million unique certificates for them. For SSH, they scanned 10.2 million hosts and obtained 6.2 million unique keys. Indicating that about 61% and 65% of the TLS and SSH hosts shared the same key as another host in the scans. Investigating the causes of this key

sharing between hosts, they found that some of the key sharing was not due to vulnerabilities in the infrastructure but large hosting providers that used a single key for multiple IP addresses. Another significant reason was TLS certificates that belonged to the same organisation. Therefore, they excluded exceptions from their data analysis process and clustered the remaining host that shared keys.

Dr. Farrell's work on surveying long-term cryptographic keys for SSH, mail, and web protocols is relevant literature as this project builds directly from [9]. The research surveys keys for ten countries and describes scans that involved hosts in Ireland and Estonia. The program [10] used to carry out these scans made use of Censys databases in 2017 and Maxmind databases in 2018 to perform these scans. Then ZMap and ZGrab are used to collect data on hosts that listen on port 25, and further, the data is processed and analysed. From one of the scans in Ireland carried out in 2018 that involved 18,268 hosts that offer at least one SSH or TLS service, approximately 53% hosts shared keys. Out of the 54,447 host-port combinations, only 36% unique keys were seen through that scan. The research also reports key sharing among countries and shows how widespread this resue is. The clustering of hosts is based on the fingerprint information collected from SSH hosts and hosts that use TLS on top of the standard protocols. "If any two IPs share a fingerprint irrespective of the port, they are clustered together" to check for key reuse among hosts. Other metadata during TLS and SSH handshakes are collected for extensive data analysis. After the data analysis, some scripts are used to visualise the cluster information to communicate with local asset holders to understand better the causes of this key reuse from an asset holder's point of view and how the network security infrastructure can be improved. [9] introduces a metric called HARK %: "Host that is resuing keys" for quantifying this key reuse. One of the main hypotheses of the work was to see if there is a correlation between the HARK% and improvement in the security posture.

As outlined in section 1.2, one of the goals of this project is to migrate the code to Python3, refactor it to increase readability, and carry out some optimisation to reduce the memory overhead. This is critical since most of the work carried out in [9] was done using a combination of Python2 and Bash Scripts, is quite memory intensive, and Python2 has deprecated since. The project also checks how key reuse has evolved over the years since 2018.

3.1 Summary

Summarising the section above, there is widespread key reuse in a vulnerable due to a variety of reasons. Although a single entity cannot be blamed, regular scanning might help in better understanding some of the causes and dangers behind the key reuse from both an asset holder and a user point of view. This section summaries some of the causes and dangers of this key reuse as described in [9, 14].

3.1.1 Causes of Key Reuse

- **Hardcoded Keys:** Many devices are shipped with default keys by manufacturers in the firmware of the device. This contributes to key reuse as some users might not have the knowledge on how to change these keys.
- **Lack of Randomness:** A significant reason for key reuse was the lack of entropy during key generation. This might be due to faulty random number generators or due to high costs of computations.
- **Multihomed Hosts:** If a host has more than one network interface with different IPv4 addresses, it will show as individual hosts in the scans.
- **Virtual Machines:** If a few hosts share a key and all of those hosts are VMs, that will contribute to this. However, in this case, it can be passed as acceptable as all hosts are under one physical entity.

3.1.2 Dangers of Key Reuse

- **Cross Protocol Attacks:** Even though new versions of TLS are available, the migration from the older version is still slow. Key reuse can increase the probability of cross-protocol attacks, and old versions of TLS/SSL may be vulnerable to these.
- **Masquerading:** If asymmetric encryption is used for authentication, then key reuse may enable an attacker to pose as a host if a breach of a host occurs in a cluster.
- **Credential Exposure:** If sensitive information like passwords are sent using plain text protocols like SMTP, and if an active attacker can masquerade as a host in a cluster, they can capture those credentials.
- **Cost of Leaks:** Since private keys are usually stored locally, there is always some probability of leaks due to hardware failures. If there is a leak from a host that shares keys with other hosts, this will affect all of them. Moreover, as the cluster size increases, the impact of these leaks would also increase.

Chapter 4

Implementation

This chapter specifies the approach taken to execute the project. Section 4.1 gives an overview and describes the technologies used to carry out this project. Sections 4.2, 4.3, 4.4, 4.5 explain how the program operates, and 4.6 provides information about some additional tooling developed to get better insight into the data. Section 4.7 provides an insight into the techniques used to refactor and optimise the orignal code [10]. The last section talks about the challenges faced during implementation.

4.1 Overview

The program for surveying keys [8] had various stages in which it performed different functions. Figure 4.1 depicts the flow of the program, and it can be divided into the following stages:

- **Environment Setup:** Downloads all the required dependencies required for the program to work, including Python libraries, Maxmind databases, ZMap, ZGrab, and setups Golang.
- **Stage 1:** This stage can be called the Maxmind stage, where the MaxMind APIs are set up and filters out IP addresses for the country selected.
- **Stage 2:** This is the scanning stage where ZMap and ZGrab are used. The IPs from Stage 1 are scanned using ZMap for open port 25. Once those IPs are available, ZGrab is used to capture data for the ports.
- **Stage 3:** The final stage of the program where data from Stage 2 is processed, stored and analysis are carried out.

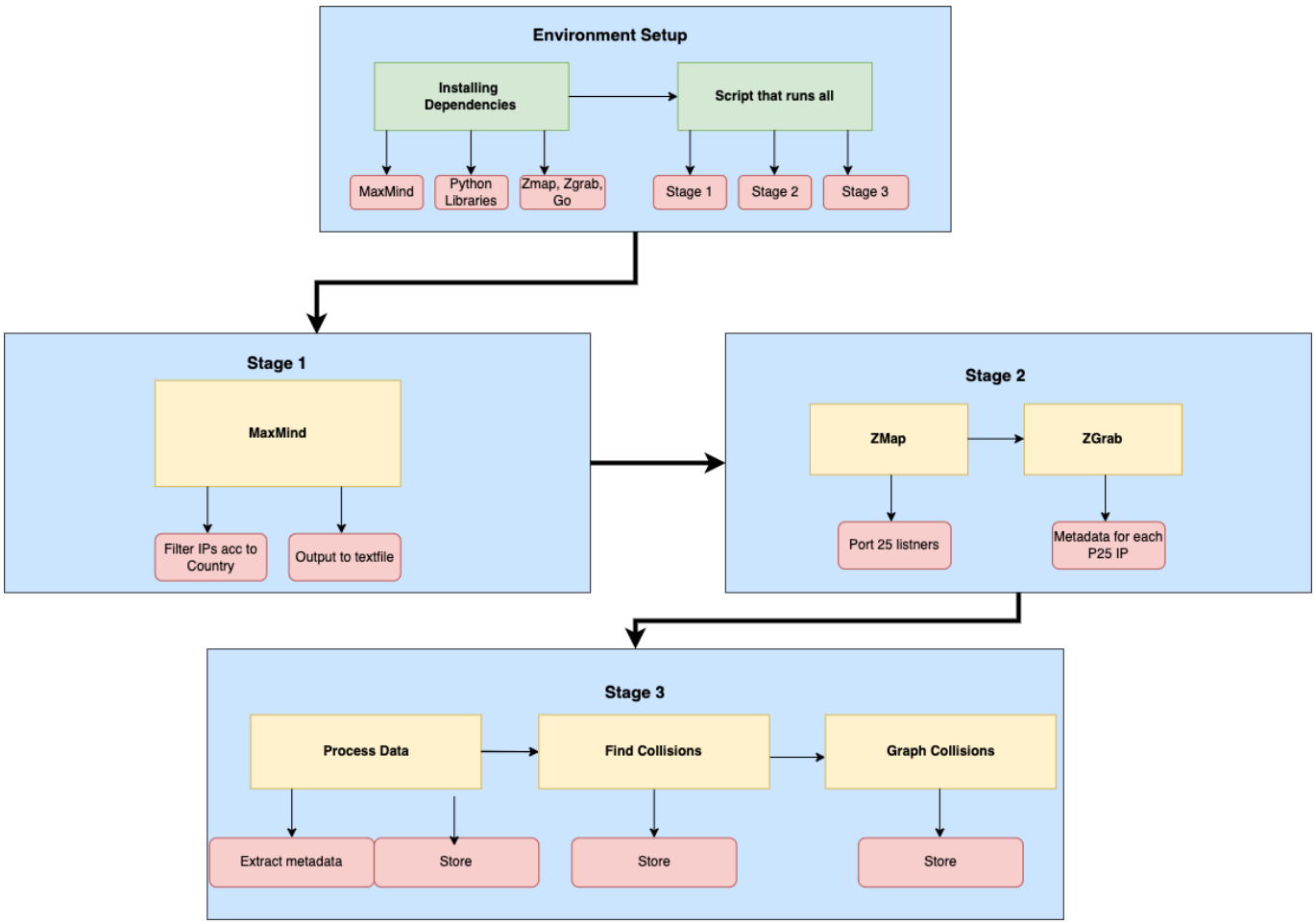

Figure 4.1: Program Strcuture

4.1.1 Technologies Used

The following lists the technologies that were used to carry out the project:

- **Python3:** Majority of the program was developed using Python3. Used for data processing, visualisation, and calling APIs and tools used.
- **Bash Script:** Used for automation of tasks, downloads and specific data extraction tasks.
- **ZMap:** Port scanner used to identity open port 25 listeners.
- **ZGrab:** Banner Grabber to obtain information about hosts in question.
- **Maxmind:** Used databases and APIs provided by MaxMind to carry out the network scans.
- **Pylint:** A code analysis tool used to measure code quality and enforce a standard coding structure.
- **cProfile and pstats:** A deterministic profiling tool used to optimise memory consumption and runtime.

4.2 Environment Setup

Before the program execution begins, a script called *"install-deps.sh"* downloads and installs all dependencies for the program to work. It starts by creating directories where the source code is available and another directory where the results for each scan are stored. After doing so, it will install dependencies like the Python libraries required, ZMap, ZGrab, and the MaxMind databases. It also installs Golang and configures the *"gopath"* as ZGrab requires a valid *"gopath"* to function. Since this program was the last run in 2018, the Maxmind databases have changed significantly. Therefore, an additional script called *"MMIPs.py"* was developed to create a CSV dataset called *"GeoIPCountryWhois"* was required for the program to run. The Python script takes in two CSV files as input that contain the entire IPv4 network blocks with their associated Geonames. In addition, there was another CSV file that had the Geonames and the country code and name associated. The script processes these two CSV files and then maps the IPv4 network blocks to their associated country code (and country names) using the Geonames. Finally, it produces the final CSV dataset needed.

4.3 Stage 1: Maxmind Stage

The program's first stage was to filter out the IP addresses for the country specificed to scan. For this project, the country choof interest was Ireland. The top script *"skey-all.sh"* does all the work and calls these stages sequentially, as indicated in figure 4.1. The script *"IPsFromMM.py"* performs the first stage. It takes in the input file *"GeoIPCountryWhois.csv"* and filters the IPv4 CIDRs according to the selected country. The final output from this stage is a text file that contains IPv4 CIDRs for the country chosen.

4.4 Stage 2: Port Scan and Banner Grab

This stage involved two parts. The first one was using ZMap to check for open port 25 listeners, and the next was using ZGrab on those hosts to gather the SSH and TLS meta-data.

4.4.1 ZMap

After getting the list of IPs for the country selected, the program moves onto the second stage and uses ZMap to map which IPs listen on port 25. ZMap is called using the main script *"skey-all.sh"* as it requires system privileges. The final output from ZMap is a list of IP addresses that listen on port

25. Figure 4.2 shows how the ZMap output looks like. It expands each IP in CIDR notation to the IP range and pings each IP in the range using the TCP SYN scan. The fields indicate the time left on the scan, the "send: 2947" shows that ZMap has pinged 2947 IPs address, and the "recv:2" means that ZMap has found two port 25 listeners. The "hitrate: 0.07%" indicates that out of the total IPs pinged 0.07% were port 25 listeners.

```
381 starting zmap
382 Mar 29 19:49:02.981 [INFO] zmap: output module: csv
383 Mar 29 19:49:02.981 [INFO] csv: no output file selected, will use stdout
384  0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
385  0:00 0%; send: 0 0 p/s (0 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
386  0:01 0%; send: 147 146 p/s (142 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
387  0:02 0%; send: 293 145 p/s (144 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
388  0:03 0%; send: 438 144 p/s (144 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
389  0:04 0%; send: 586 147 p/s (145 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
390  0:05 0% (1d07h left); send: 734 147 p/s (145 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
391  0:06 0% (1d07h left); send: 883 148 p/s (146 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
392  0:07 0% (1d07h left); send: 1031 147 p/s (146 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
393  0:08 0% (1d07h left); send: 1179 147 p/s (146 p/s avg); recv: 0 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.00%
394  0:09 0% (1d07h left); send: 1327 147 p/s (146 p/s avg); recv: 1 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.08%
395  0:10 0% (1d07h left); send: 1474 146 p/s (146 p/s avg); recv: 1 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.07%
396  0:11 0% (1d07h left); send: 1622 147 p/s (146 p/s avg); recv: 1 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.06%
397  0:12 0% (1d07h left); send: 1769 146 p/s (146 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.11%
398  0:13 0% (1d07h left); send: 1918 148 p/s (147 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.10%
399  0:14 0% (1d06h left); send: 2066 147 p/s (147 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.10%
400  0:15 0% (1d06h left); send: 2214 147 p/s (147 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.09%
401  0:16 0% (1d06h left); send: 2362 147 p/s (147 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.08%
402  0:17 0% (1d06h left); send: 2510 147 p/s (147 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.08%
403  0:18 0% (1d07h left); send: 2652 141 p/s (146 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.08%
404  0:19 0% (1d07h left); send: 2800 147 p/s (146 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.07%
405  0:20 0% (1d07h left); send: 2947 146 p/s (146 p/s avg); recv: 2 0 p/s (0 p/s avg); drops: 0 p/s (0 p/s avg); hitrate: 0.07%
```

Figure 4.2: ZMap Output

4.4.2 ZGrab

The following process in this stage was to gather metadata about each port for IP addresses we obtained from ZMap. This is where the ZGrab2 tool was used, and the process was done using the script *"FreshGrab.py"*. The script specifies the ZGrab parameters and calls the ZGrab binary. It requires an input file that is a list of IPs (ZMap output) and first verifies whether the IP belongs to the specified country using methods defined that make use of the Maxmind databases and APIs. If the IP does not match the country according to Maxmind, they are marked as "out of country" and are not processed further. The remaining IP addresses are processed using the ZGrab tool. Since the output is in JSON, the data is stored in a file called *"records.fresh"* that contains one JSON structure per line. Each line has information about all seven ports for each IP. This part of Stage 2 can take anywhere from a few hours to a day to complete since the scan rate of the ZGrab tool was limited by adding a 100 ms wait between each IP so as not to cause any congestion in the network.

```
ports=['22', '25', '110', '143', '443', '587', '993']
ztimeout=' -t 2'
pparms={
    '22': 'ssh -p 22',
    '25': 'smtp -p 25 --starttls',
    '110': 'pop3 -p 110 --starttls',
    '143': 'imap -p 143 --starttls',
    '443': 'http -p 443 --use-https',
    '587': 'smtp -p 587 --smtps',
    '993': 'imap -p 993 --imaps',
    }
for port in ports:
    cmd=zgrab_path + " " + pparms[port] + ztimeout
    proc=subprocess.Popen(cmd.split(),stdin=subprocess.PIPE,stdout=subprocess.PIPE)
    pc=proc.communicate(input=ip.encode())
```

Listing 4.1: ZGrab2 Parameters

Listing 4.1 shows how ZGrab is used in the *"FreshGrab.py"* script. The parameters specify the protocols and associated ports for it to scan for. To capture the TLS metadata for the plain text ports, the *STARTTLS* command was used, and implicit TLS for the other ports.

4.5 Stage 3: Data Processing and Visualisation

Stage 3 of the program involved parsing out the metadata from the JSON structures the project required, performing DNS and reverse DNS lookups, storing the data, and finding where the same key is being used for each IP for every possible host-port combination. Stage 3 can be broken down as follows:

4.5.1 Data Processing

The script that handles the data processing is *"SameKeys.py"* and starts by iterating through data for each IP address in file *"records.fresh"*. Since plenty of metadata is captured for each IP, the data is stored using Python, a class instance with multiple attributes. Then, the program starts by performing reverse DNS lookups for the IP and storing the names obtained in the same class instance. After that, Fully Qualified Domain Names, TLS certificates, and fingerprint information are parsed and stored for every port. The FQDNs are stored to assist with verifying the asset owners that are operating

the said service. The program also keeps the information about the autonomous system associated with each host using the Maxmind databases. The methods to parse and store these fields are defined in the *"SurveyFuncs.py"* script. Before moving onto the analysis stage, the program verifies names associated with the IPs by performing DNS lookups with the SANs related to the IP. The program only stored a maximum of 20 SANS per host, as some hosts have a large amount of SANs that slow down the program. If the IP addresses from the DNS lookup match the IP in *"records.fresh"*, it is recorded as "good". Otherwise, it is recorded as "bad". The IPs marked as "good" with a key are stored in a JSON file for further analysis. The same is done with the "bad" IPs, and another file contains both IPs marked as both "good" and "bad" with their associated information. Appendix A provides the output of ZGrab and shows the information collected for all seven ports for each IP address. The appendix only provides data for one IP address in JSON format.

4.5.2 Data Analysis

The data analysis process entails checking for the duplicate keys for different services within and across IPs. The clustering is based on fingerprint SHA-256 for each service. If a specific key is shared by two hosts irrespective of the protocol, they are put in the same cluster. This is because it is not uncommon to see key reuse for different services, as proved in [9]. When key reuse is identified, the initial collisions are cross-checked with each other and are merged if the same key is being used there. Once, this is complete three files are produced as follows:

- "collisions.json": Contains key reuse information among hosts.
- "dogies.json": Contains information about IP that are recorded as "bad".
- "all-key-fingerprints.json": Contains information that is included in the first two files.

4.5.3 Data Visualisation

After obtaining the data as describe abovethe collisions are graphed using the Graphviz library in Python. The file used to graph the collisions is *"collisons.json"*. Each host in the file is iterated through, and the hosts with the same cluster number are finally graphed using custom methods defined based on Graphviz.

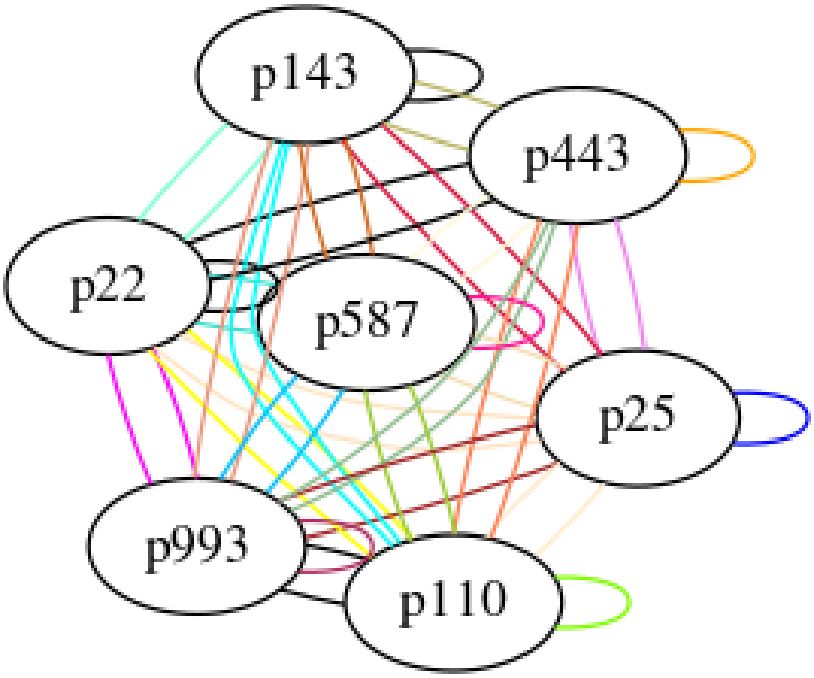

Figure 4.3: Graph Legend

Figure 4.3 represents the graph legend, and the graphs can be visualised as follows:

- The nodes are represented as IP addresses.
- The colour of the nodes represents the entities or hosts.
- The edges represent key reuse across ports.
- The colour represents the pair port combination.

4.6 Additional Tools

Some additional scripts are available that provide in-depth analysis of the data collected. For example, the script *"ProtocolVersions.py"* provides an insight into the TLS/SSL versions seen throughout the scans and provides a count for them. It also provides details of the SSH versions seen on port 22. More tooling offers information on how many IP addresses were mapped by ZMap, how many host port combinations are there, how many unique fingerprints are seen, and many more. All of these scripts were used that assisted in the analysis of the results.

4.7 Code Migration, Refactoring and Optimisation

One of the primary goals of this project was to migrate the code to Python3 and refactor it to increase readability, improve the structure, and optimise the run time and memory usage. This section gives details about the migration and the refactoring process. It also provides information about the techniques used to increase code performance.

4.7.1 Code Refactoring

The refactoring process was carried out by first analysing the entire program to check for duplicate code. Methods that were complex methods and involved heavy nesting were identified. In the *"SameKeys.py"* script where the data processing takes place, all information parsing was done manually according to the JSON headers fields for each port. Due to this, the code was not so readable and looked complex. Therefore, the techniques outlined in section 2.6 were used to carry out the refactoring. The following code snippets give an example of this.

```
try:
    p25=j_content['p25']
        if thisone.writer=="FreshGrab.py":
            banner=p25['data']['banner']
        else:
            banner=p25['smtp']['starttls']['banner']
        ts=banner.split()
        if ts[0]=="220":
            banner_fqdn=ts[1]
            nameset['banner']=banner_fqdn
        elif ts[0].startswith("220-"):
            banner_fqdn=ts[0][4:]
            nameset['banner']=banner_fqdn
except Exception as e:
    print >> sys.stderr, "FQDN banner exception " + str(e) + " for record:" +
        str(overallcount) + " ip:" + thisone.ip
    nameset['banner']=''
try:
    if thisone.writer=="FreshGrab.py":
        tls=j_content['p25']['data']['tls']
        cert=tls['server_certificates']['certificate']
    else:
        tls=j_content['p25']['smtp']['starttls']['tls']
```

```
        cert=tls['certificate']
        fp=cert['parsed']['subject_key_info']['fingerprint_sha256']
        get_tls(thisone.writer,'p25',tls,j_content['ip'],thisone.analysis['p25'],scandate)
        get_certnames('p25',cert,nameset)
        thisone.fprints['p25']=fp
        somekey=True
except Exception as e:
    print >> sys.stderr, "p25 exception for:" + thisone.ip + ":" + str(e)
    pass
```

Listing 4.2: Code before Refactoring

```
try:
    p25 = j_content['p25']
    bn = "y"
    if thisone.writer == "FreshGrab.py":
        banner_fqdn = get_mail_data(p25, bn)
    else:
        banner = p25['smtp']['starttls']['banner']
        nameset['banner'] = banner_fqdn
except Exception as e:
    print(sys.stderr, "FQDN banner exception " + str(e) + " for record:" +
        str(overallcount) + " ip:" + thisone.ip)
    nameset['banner'] = ''
try:
    if thisone.writer == "FreshGrab.py":
        data = j_content['p25']['data']['smtp']['result']['tls']
        cert, fp = get_mail_data(data, None)
    else:
        tls = j_content['p25']['smtp']['starttls']['tls']
        cert = tls['certificate']
    get_tls(thisone.writer, 'p25', data, j_content['ip'], thisone.analysis['p25'],
        scandate)
    get_certnames('p25', cert, nameset)
    thisone.fprints['p25'] = fp
    somekey = True
except Exception as e:
    print (sys.stderr, "p25 exception for:" + thisone.ip + ":" + str(e))
    pass
```

Listing 4.3: Code After Refactoring

Listing 4.2 and Listing 4.3 shows the code before and after refactoring respectively. In this instance, a method was introduced called `get_mail_data(data,banner)` that was used to extract the banner information for port 25, but also to gather the TLS certificate and fingerprint data for all mail protocols instead of doing this manually for each mail port.
Other methods were created as well, and the existing ones were modified for simplicity. This was done to decrease the number of lines of code in the *"SameKeys.py"* script and to avoid writing duplicate code. It also had another benefit that in case there are changes to the ZGrab output in the future, one will have to make changes in the program at one point instead of doing it manually for all ports.

Another instance where refactoring was carried out was in the *"SurveyFuncs.py"* script that contains custom utility functions. While it is still somewhat acceptable to have manual code in this script as it is not used directly, refactoring was carried out here to improve performance. For example, existing methods that were used to call the Maxmind APIs contained a lot of nesting of *if* statements and *for* loops. To decrease the nesting, the abstraction technique was used to break down the nested statements by defining different methods to perform the same function overall.

4.7.2 Code Optimisation

To optimise this program, memory analysis had to be carried out that gave information about the memory overheads of the program and identified what was taking the most amount of time to complete the whole process. This analysis was carried out using the *cProfile* and *pstats* libraries available in open-source. A profile here is defined as a "set of statistics that describes how often and for how long various parts of the program are being executed" [33]. When used with the scripts, this library gives a detailed report about how the program behaves. It gives information about the number of monitored calls and how many of those were primitive or recursive. In addition, it provides the total time spent in each method and the cumulative time spent in the method and all sub-methods. Once this data was available, it was investigated, and areas of the program were identified where bottlenecks were being created. Finally, alternative solutions were explored to decrease run time and to decrease memory usage.

```
rs@rs:~/surveys/rsstuff$ ./memory_profile.py
Wed Apr 13 14:32:03 2022    /home/rs/survey_results/old_mem_stats

         248238208 function calls (198949372 primitive calls) in 73816.116 seconds

   Ordered by: cumulative time
   List reduced from 2574 to 15 due to restriction <15>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    363/1    0.018    0.000 73816.116 73816.116 {built-in method builtins.exec}
        1  222.382  222.382 73816.116 73816.116 SameKeys.py:22(<module>)
    43941    0.975    0.000 70991.462    1.616 /home/rahulseth/surveys/SurveyFuncs.py:765(get_dns)
    43941 70985.715    1.615 70989.834    1.616 {built-in method _socket.gethostbyname}
    17647    0.483    0.000 1816.256    0.103 /home/rahulseth/surveys/SurveyFuncs.py:753(get_rdns)
    17647 1813.977    0.103 1815.126    0.103 {built-in method _socket.gethostbyaddr}
    17647    0.442    0.000  402.074    0.023 /home/rahulseth/surveys/SurveyFuncs.py:710(mm_info)
    17647    0.552    0.000  401.633    0.023 /home/rahulseth/surveys/SurveyFuncs.py:726(extract_from_mm)
    52941    0.575    0.000  385.286    0.007 /home/rahulseth/.local/lib/python3.6/site-packages/geoip2/database.py:232(_get)
    52941    1.809    0.000  384.711    0.007 /home/rahulseth/.local/lib/python3.6/site-packages/maxminddb/reader.py:123(get_with_prefix_len)
    35294    0.779    0.000  330.803    0.009 /home/rahulseth/.local/lib/python3.6/site-packages/geoip2/database.py:245(_model_for)
6407352/105882   98.081    0.000  247.852    0.002 /home/rahulseth/.local/lib/python3.6/site-packages/maxminddb/decoder.py:141(decode)
467844/105882   31.748    0.000  232.761    0.002 /home/rahulseth/.local/lib/python3.6/site-packages/maxminddb/decoder.py:85(_decode_map)
    17647    0.245    0.000  218.798    0.012 /home/rahulseth/.local/lib/python3.6/site-packages/geoip2/database.py:142(city)
    52941    4.989    0.000  186.128    0.004 /home/rahulseth/.local/lib/python3.6/site-packages/maxminddb/reader.py:154(_find_address_in_tree)
```

Figure 4.4: Memory Profile before Refactoring

Figure 4.4 depicts the output upon profiling the code before any optimisation or major refactoring was carried out. The above profiling was done on the same data that was used to carry out the data analysis. Referring to the figure, it is visible the total execution time of the program was about 73816 seconds, but most of the time was consumed by the function `gethostbyname()` that is used to perform DNS lookups on the names parsed out from the metadata. The names include banner information and subject alternative names (SANs). Once this bottleneck was identified, alternative DNS resolution solutions were looked at (discussed in the section below), and different benchmark tests were created in order to see the timing of each solution and which met the scope of the project.

4.7.3 DNS Resolving

After memory profiling the program, it was found that the DNS resolution part of the code was consuming much time. Few alternate solutions were explored for faster resolution to decrease the run time. Reasons why DNS resolution was creating a bottleneck were investigated. Some solutions that were looked at to mitigate this problem were: DNSpython, MassDNS Resolver, Berserker Resolver, and Stubby plus Unbound. A benchmark timing test was created for the four, and it was found that using a combination of stubby plus unbound was the fastest among them. The problem with the first three options were as follows:

- MassDNS [38] is used to make queries in the range of millions to billions and was not fitting the scope of the project.
- DNSpython [12] had the same performance as `gethostbyname()` function in the socket library unless a timeout value was set for lookups which was not the most efficient way to solve this issue.
- Berserker Resolver [4] had an upgrade in performance but used DNSpython in the backend.

A more permanent solution was required, and that was using a combination of two open-source tools called Unbound and Stubby. Unbound is a validation, recursive, caching DNS resolver that is designed to be fast and lean and provides modern services like DNS over TLS and DNS over HTTPS, which allows encryption while making name resolutions [17]. Stubby is an application that acts as a local DNS stub resolver and uses DNS-over-TLS for resolutions. The combination of Unbound and Stubby was used to speed up the DNS resolution for our program.

Unbound was used behind stubby, and all queries made using Unbound were forwarded to Stubby. Since Stubby uses DNS over TLS, which is assigned to port 853 as compared to traditional DNS that operates on port 53. The configuration had to be set up for the same. Before testing the code with this combination, Wireshark was used to analyse the traffic on port 853 while performing DNS lookups to ensure it was functioning as expected. Wireshark is a network packet analyser that captures in-depth detail about the captured packet [45].

4.8 Challenges

This project came across a few expected challenges. One of the significant challenges was configuring and visualising the ZMap and ZGrab tools. Although it was relatively easy to understand how ZMap works and how the output would look, the ZGrab posed a real challenge since the output from the tool was in JSON and involved heavy nesting. Development for this program was done using various Linux systems, and the table below shows the Virtual Machines that were set up and their purpose during the course of this project.

Ubuntu Version	Purpose
22.04	Development
21.04	Target VM for testing ZGrab
18.04	Testing

Table 4.1: VM Setup

Service	Protocol
Apache Server	HTTPS,SSH
Dovecot	IMAP and POP3
Postfix	SMTP

Table 4.2: Target VM Setup

The target VM was set up with different servers available in open-source that offered the protocols required. Table 4.2 shows the server setup on the target VM and the protocols provided by each. These servers were set up using minimal configurations to test the ZGrab tool over localhost. Since the program used a Python Script to use the ZGrab tool and performed IP checks according to the MaxMind databases, the reserved IP addresses could not be used as MaxMind does not recognise them as it operates the same blacklist as ZMap. Due to this, the ZGrab tool was tested manually using the command-line interface for each protocol by adding multiple IPv4 addresses and mapping them to each service configured.

Chapter 5

Results and Discussion

This chapter explains the results obtained after carrying out the Data Analysis process. It demonstrates key reuse across numerous hosts through the help of graph visualisations and other tables to provide an in-depth analysis of the clusters. This chapter also provides insights into the results obtained after optimisation and how it affected the program's run time and memory usage.

5.1 Overview of Results

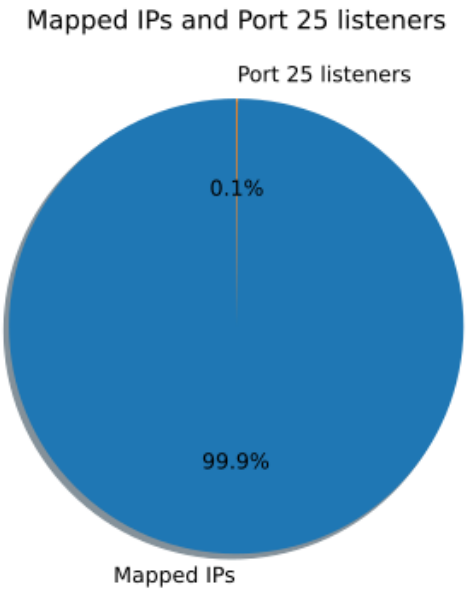

Figure 5.1: IE Port 25 listeners

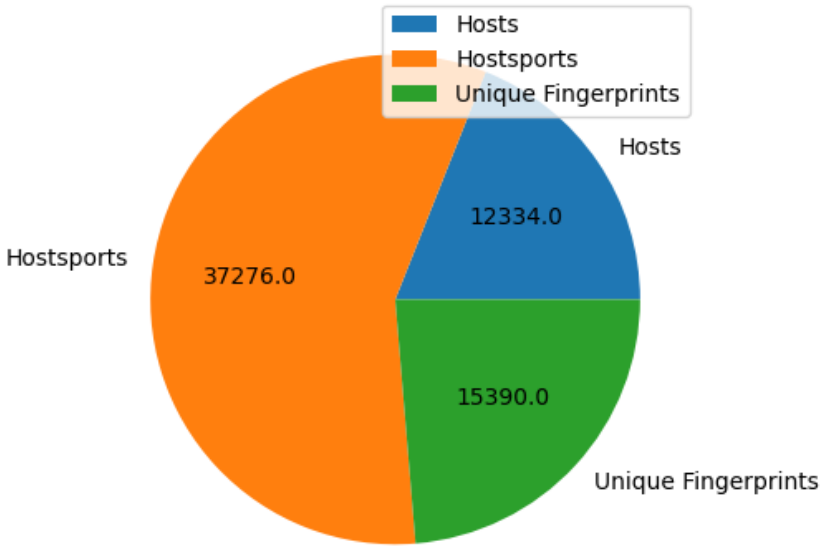

Figure 5.2: Hosts v/s Host Ports v/s Fingerprints

Figure 5.1 shows the number of IP addresses identified by ZMap as port 25 listeners. Out of the total IPs for Ireland (16421058), only 17,665 were identified as mail servers making it 0.1% of the total IPv4s assigned to Ireland.

12,333 hosts did some cryptography out of the 17,665 hosts identified as port 25 listeners, as indicated by figure 5.2. Out of 12,333, about 37,276 host-ports combinations did some cryptography, but there were only 15,390 (41%) unique fingerprints seen throughout the scans. As only 41% fingerprints were unique, it can be concluded that there is key sharing among hosts. The sections below further analyse this key reuse and provide analysis of some of the intriguing clusters found.

5.2 Protocol Versions

5.2.1 Cryptography per port Count

Figure 5.3 (below) depicts how many ports do some sort of cryptography across all IP addresses. Surprisingly, only 32 hosts on port 587 were found to be offering cryptographic services, and port 443 had the most number of hosts.

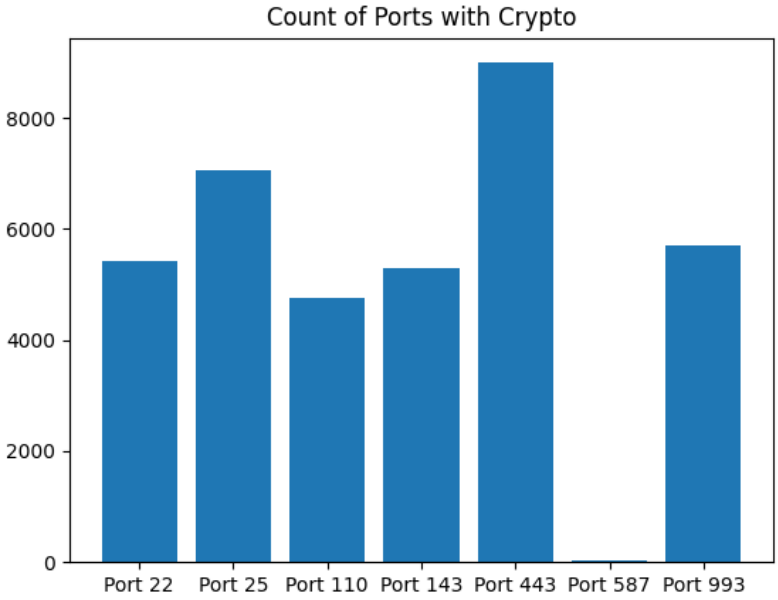

Figure 5.3: Ports that offer Crpto

5.2.2 SSH Versions

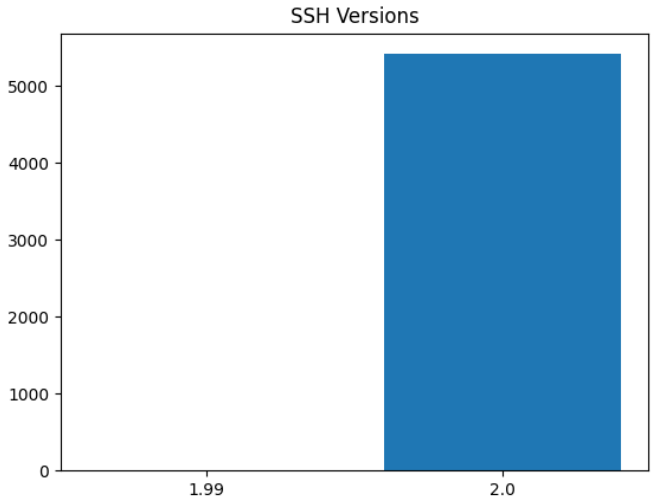

Figure 5.4: SSH Versions

Figure 5.4 above represents the SSH versions seen throughout the scans. Only two SSH versions are operating with only one host using them. SSH 1.9 and the rest of them using SSH 2.0. SSH 2.0 was introduced in 2006 and is the current standard version of SSH introduced by the IETF. It provides significant improvements in terms of efficiency and security. SSH versions 1.0 and 2.0 are incompatible. Hence, SSH 1.9 was introduced by the IETF to provide some backward compatibility between the two versions [20].

5.2.3 TLS Versions

port	SSLv3	TLSv1.0	TLSv1.1	TLSv1.2	Total
p25	0	120	4	6930	7054
p110	1	166	3	4600	4770
p143	0	179	3	5108	5290
p443	0	266	3	8729	8998
p587	0	2	0	30	32
p993	0	144	7	5563	5714
Total	1	877	20	30960	31858

Table 5.1: TLS Versions

Table 5.1 shows the TLS versions seen throughout the scans irrespective of the IP address belonging to a cluster. Most of the TLS versions seen are TLS 1.2, followed by TLS 1.0, with the least SSLv3. There was no instance of TLS 1.3 in our scans. Surprisingly, it was observed that a significant number of hosts are still operating TLS 1.0, and some are employing TLS 1.1 even though both TLS versions have been depreciated by the IETF. The old versions were depreciated because of significant security flaws associated with the old versions. Some of the reasons behind the depreciation of TLS 1.0 and 1.1 are:

- Old TLS versions require implementation using old cipher suites that are no longer desirable from a security point of view.
- Old versions do not support the modern and recommended cipher suites.
- The integrity of the handshake and the authentication process depend on SHA-1 hashes and signatures, respectively, which can easily be broken and has been superseded by SHA-2.

TLS 1.0 was released in 1999 and is considered to be the weakest of all TLS versions, and the IETF does explicitly not permit the use of it [28]. This is because it is known to suffer from attacks like the BEAST due to improper implementation of the Cipher Block Chaining [39].

Even though TLS 1.2 has also been upgraded by TLS 1.3, it was not seen during the scans. This might be because the TLS 1.3 upgrade is one of the most significant upgrades as compared to other TLS versions. The new upgrade introduced new features like 0-RTT that provide significant speed upgrades during a TLS connection establishment. Since TLS 1.3 was drastically different, and due to middleboxes all over the internet, the TLS handshake might seem like a TLS 1.2 connection even though it is TLS 1.3 [35].

5.3 Key Reuse Results

Five thousand eight hundred nine collisions were found, producing about 1049 clusters. The graphs below are a few of the selected clusters from the results. Some were selected based on the number of hosts, while others were chosen randomly.

5.3.1 Cluster 15

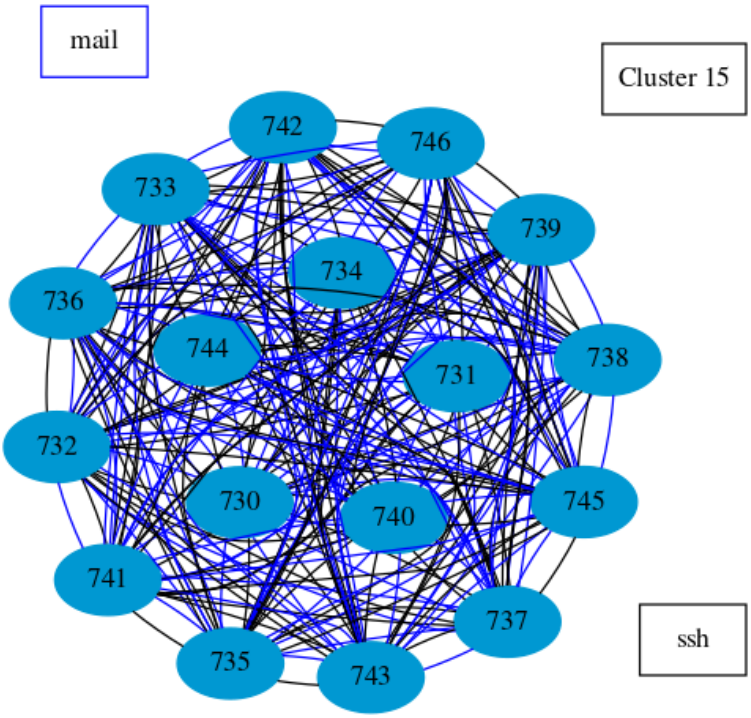

Figure 5.5: Cluster 15

Figure 5.5 is the biggest pure SSH cluster and has about 15 hosts in total belonging to the same AS. There are 68 hostport combinations, with 34 of them being SSH and the other 34 being TLS ports. For both SSH and TLS, only see one key was used among all hosts for both protocols.

Although Cluster 5 is the biggest SSH cluster found in our results but was not rendered due to a lack of memory in the machine. It has 226 hosts belonging to the same AS with about 1322 host port combinations. Out of the total host port combinations, 418 are SSH ports, with only one SSH key being shared. The rest of the 904 host-ports combinations are TLS ports, and one TLS key is shared among them.

5.3.2 Cluster 43

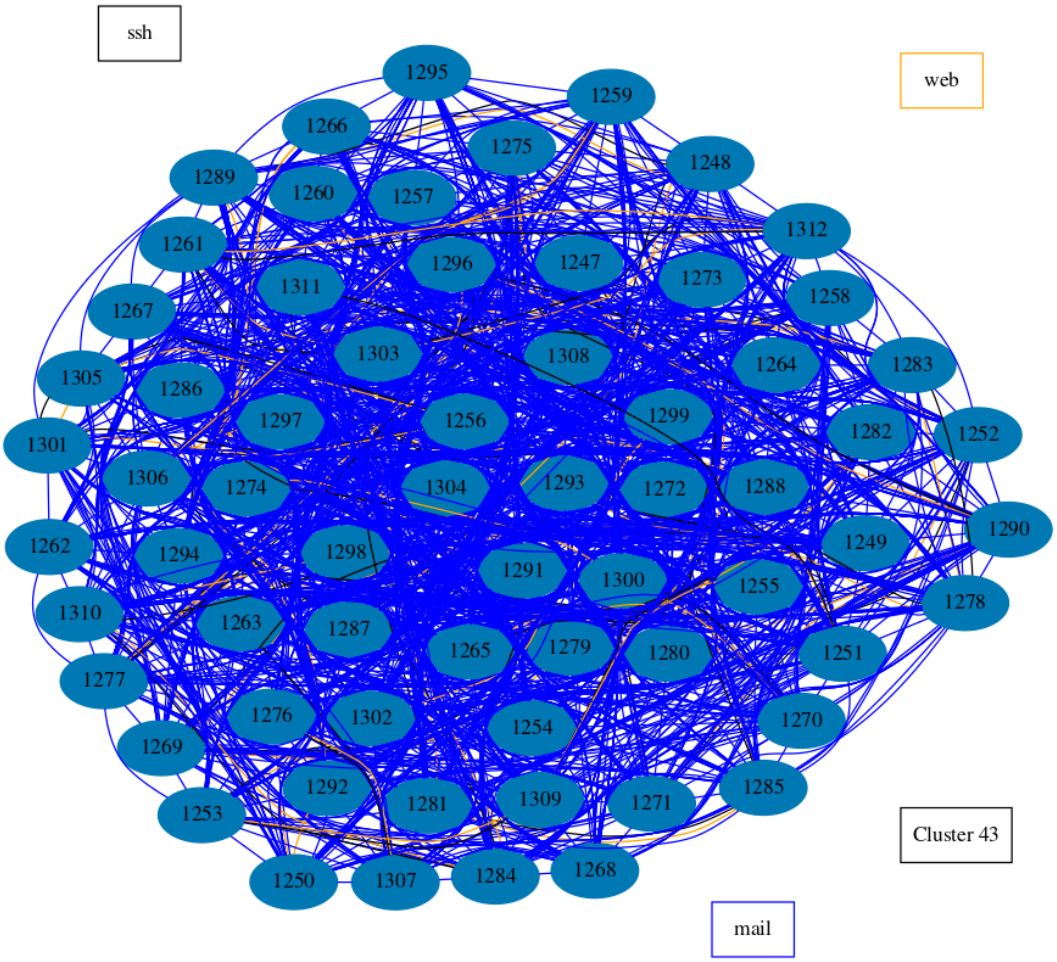

Figure 5.6: Cluster 43

Figure 5.6 represents Cluster 43 from the results. It is a cluster consisting of 66 hosts sharing the same keys for SMTP, SSH and HTTPS protocols. All of the hosts belong to the same Autonomous System in this case. There are about 722 host-port combinations, with 86 of them being SSH. There were only 14 unique SSH keys seen for the 86 host-port combinations. Out of the 722 host-port combinations, 636 were TLS ports with only 22 unique TLS keys seen. The AS in question here is a web-hosting service with a local presence.

5.3.3 Cluster 72

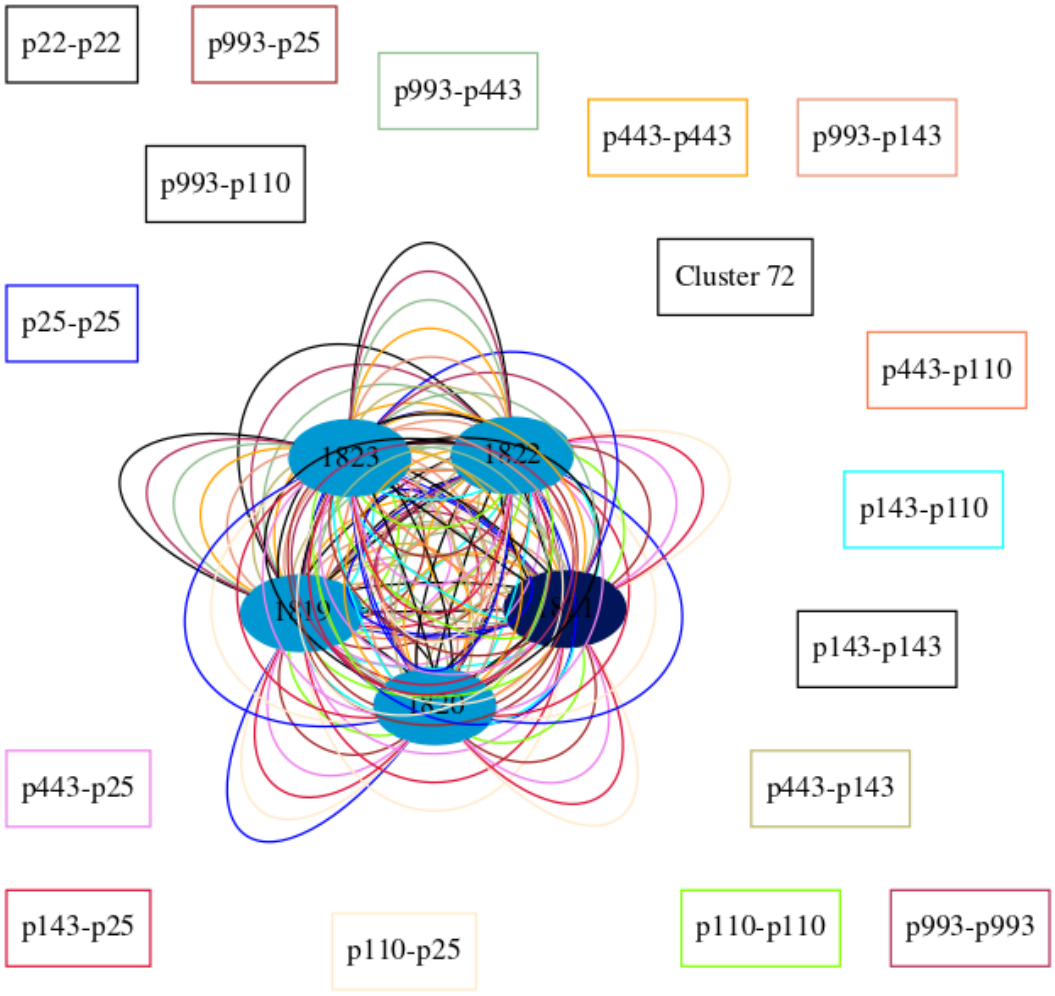

Figure 5.7: Cluster 72

Figure 5.7 is an interesting one. Key sharing across almost every pair port combination is observed. There are five hosts in this cluster, out of which four belong to the same AS. There is key sharing across all mail protocols, but even cross-protocol key sharing among hosts for HTTPS and SMTP, IMAP and POP3 is observed. Upon inspection of the cluster data, it was found that the four ASes that are the same are again a web-hosting service, while the other AS seems to belong to a Telecommunications company.

5.3.4 Cluster 133

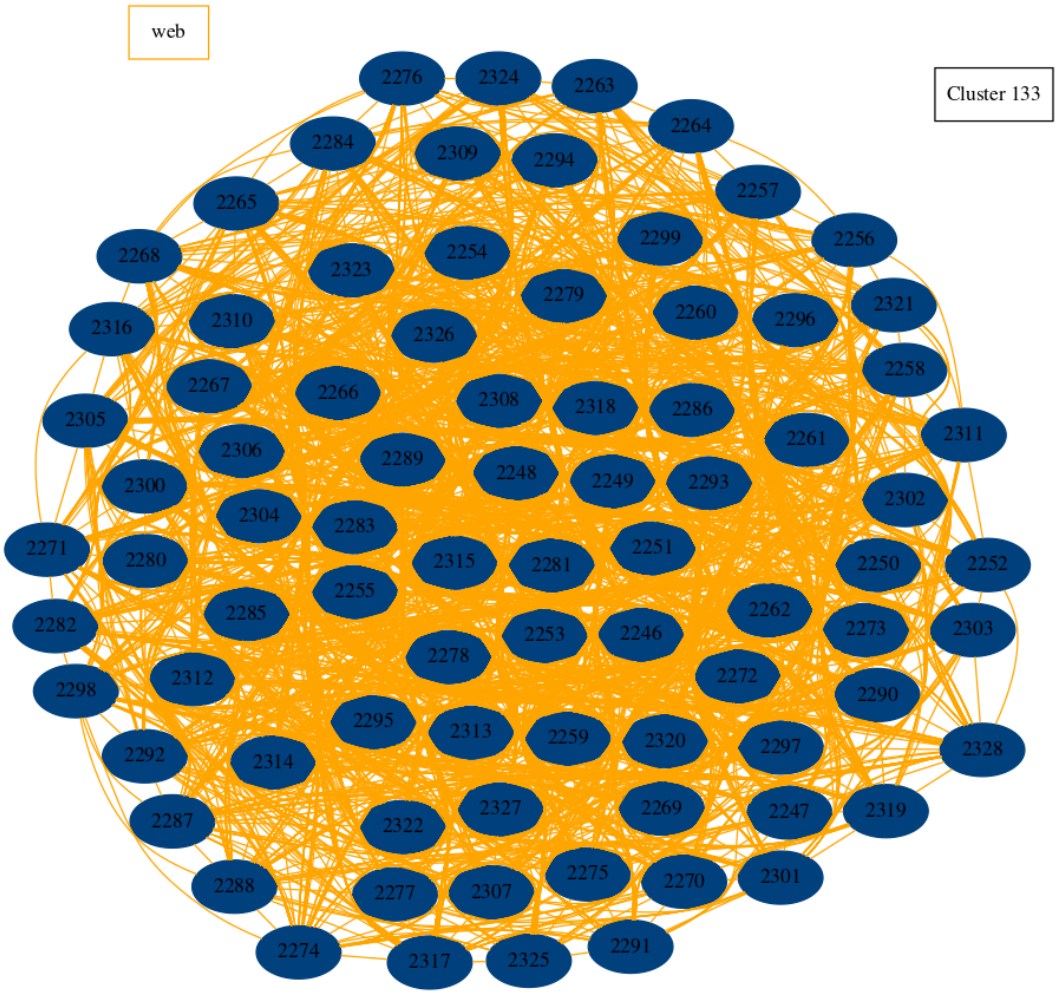

Figure 5.8: Cluster 133

Figure 5.8 represents Cluster 133 from the results, and it consists of 83 hosts sharing web server keys (HTTPS). This is one of the busier clusters found in the results, and there are a few clusters larger than this. All 83 hosts belong to the same AS, a global web hosting service, and share keys for only port 443. There are about 166 host-port combinations, sharing only a single TLS key.

5.3.5 Cluster 148

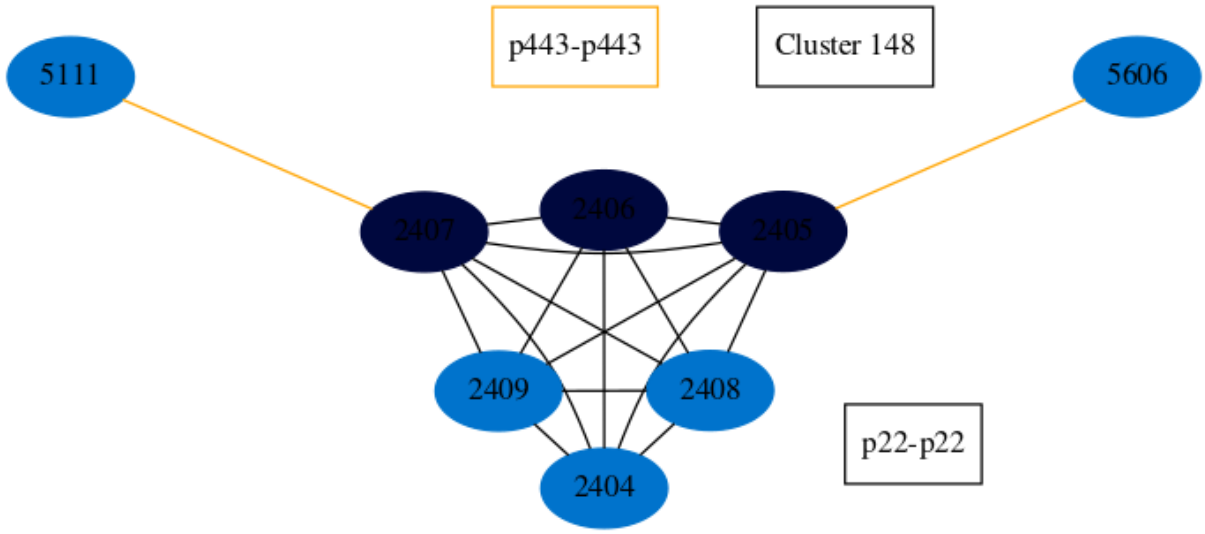

Figure 5.9: Cluster 148

Figure 5.9 represents Cluster 148 and consists of eight hosts belonging to two different ASes. Out of the eight, three belong to the same AS and 5 to a different one. All hosts in the middle share keys for SSH, while the the two host *5111* and *5606* (refer figure) at the edges share keys with the hosts *2607* and *2605* for port 433 respectively. There are about 22 host-port combinations seen, out of which 12 are classified as SSH, and the remaining 10 are TLS. Only 1 SSH key is shared among the 12 host-port combinations, while 3 TLS keys are shared for ten host-port combinations. The AS with three hosts belong to a local ISP, and the other one with five hosts belongs to a telecommunications entity with a huge local presence.

5.3.6 Cluster 536

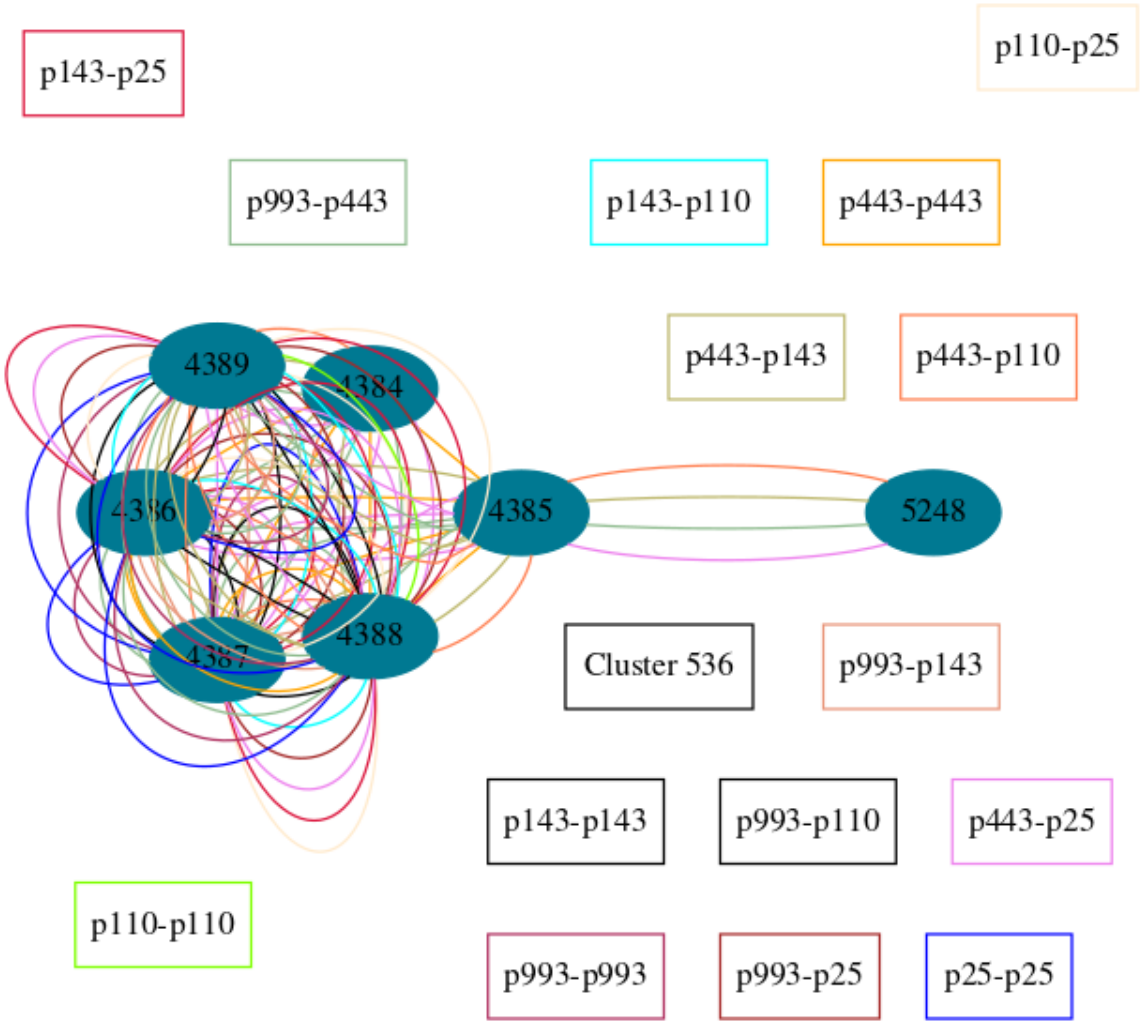

Figure 5.10: Graph 536

Figure 5.10 represents Cluster 536 and has about seven hosts, all belonging to the same AS. There are seven hosts with 58 hosts-port combinations and only four unique TLS keys. Key reuse across almost port combinations is seen here. The max key usage for a single seen was 38 times in this cluster. The hosts involved in this cluster seem to provide web services like mail and DNS using cloud platforms.

5.3.7 Cluster 786

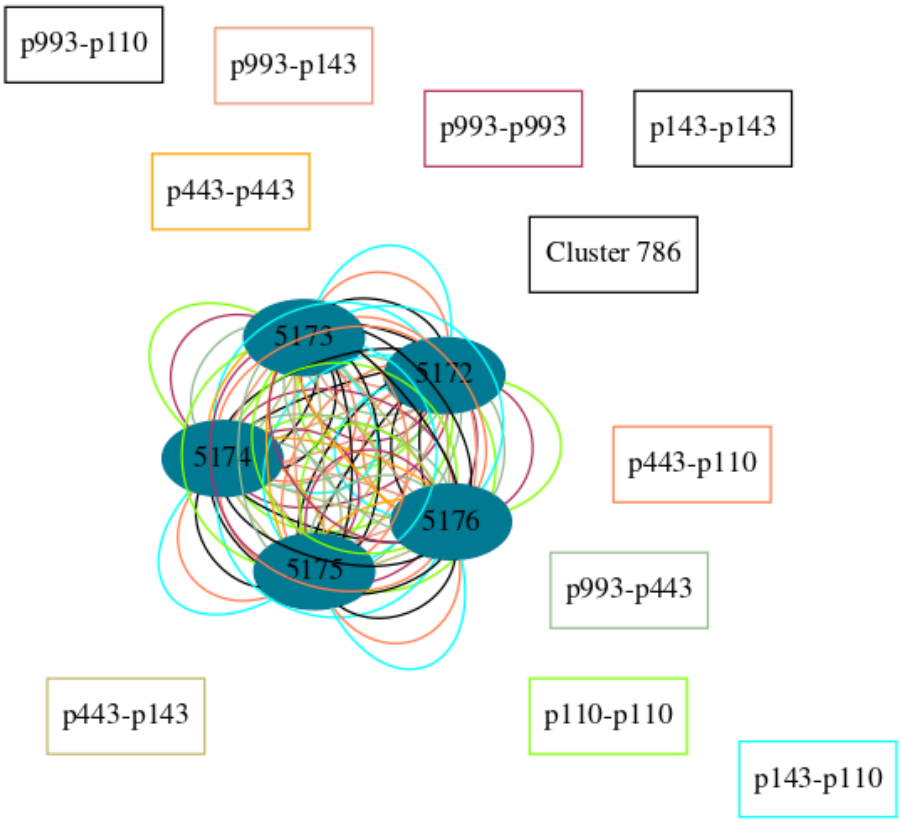

Figure 5.11: Cluster 786

Figure 5.11 consists of five hosts; all belong to the same AS and share keys for TLS. There were 40 host-port combinations with only two TLS keys, out of which one of the keys was reused 38 times. This cluster is interesting as one of the hosts has a domain name that belongs to Trinity College, Dublin. The domain name associated was on port 443 (HTTPS). For instance, if Trinity's website is "www.tcd.ie", the domain name in question here looked like "www.tcdxxxx.ie". The AS here is a telecommunications company in Ireland.

5.3.8 TLS Cipher Suites

The TLS Cipher Suites seen for the selected clusters:

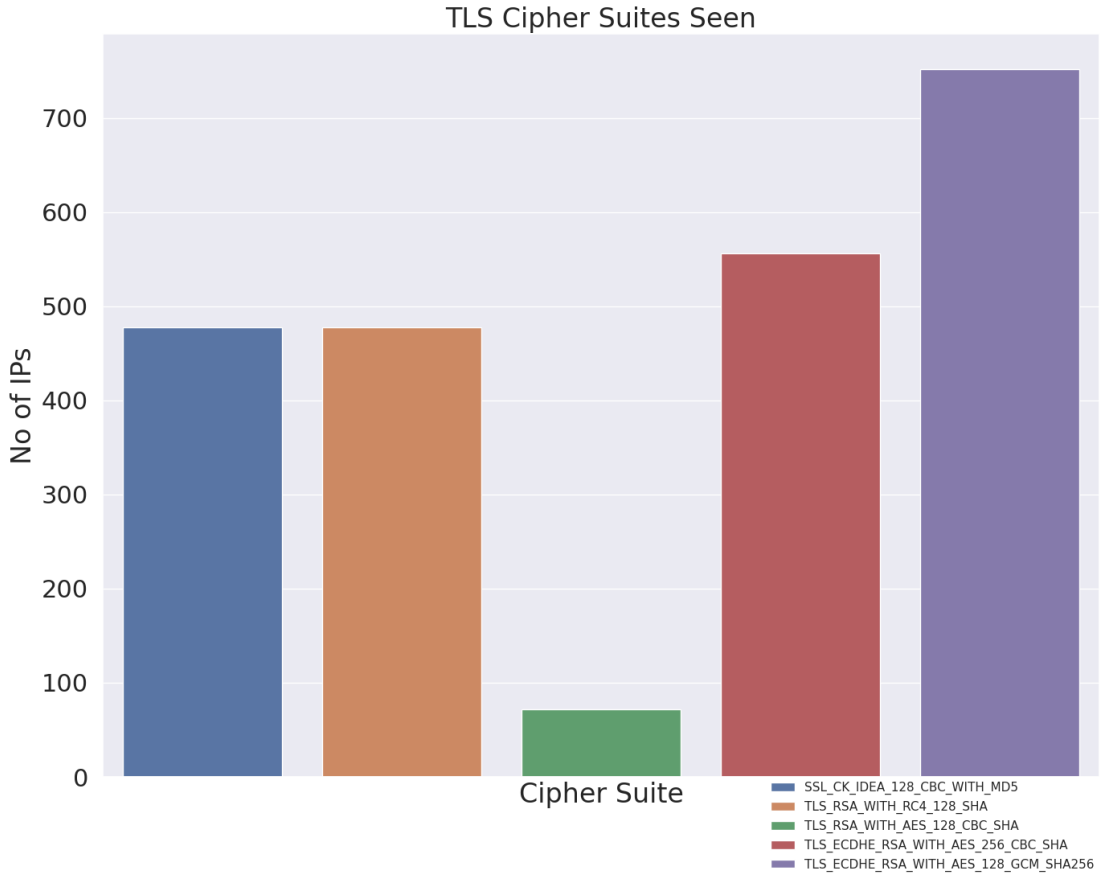

Figure 5.12: TLS Cipher Suites

Table 5.12 shows the TLS cipher suites seen for the selected clusters. The most common cipher suite was found to be `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` followed by `TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA`. Many hosts were using `TLS_RSA_WITH_RC4_128_SHA`, which is known to have cryptographic weaknesses. The IETF prohibits the use of any RC4-based Cipher Suite as they do not provide desired security [31]. The least used Cipher Suite was `TLS_RSA_WITH_AES_128_CBC_SHA`. There were also significant hosts using the `SSL_CK_IDEA_128_CBC_WITH_MD5` cipher that was introduced in 1992. Since then extensive studies have been carried out on it and is known to be broken. [41]

5.4 Post Refactoring and Optimisation

This section discusses the results obtained after code refactoring. The profiling is done again after making the changes and modifications in the program. There is also a timing graph presented that compares the average time per IP spent analysing using the two DNS setups.

```
rs@rs:~/surveys/rsstuff$ ./memory_profile.py
Tue Apr 12 02:20:04 2022    /home/rs/survey_results/new_setup_profile

         237109343 function calls (187821058 primitive calls) in 3750.811 seconds

   Ordered by: internal time
   List reduced from 2184 to 15 due to restriction <15>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    43952 2135.976    0.049 2137.045    0.049 {built-in method _socket.gethostbyname}
    17647 1111.867    0.063 1112.373    0.063 {built-in method _socket.gethostbyaddr}
        1  244.595  244.595 3750.811 3750.811 SameKeys.py:1(<module>)
6407352/105882   23.982    0.000   50.327    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:141(decode)
  5974621   21.024    0.000   25.256    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/reader.py:190(_read_node)
45503685/16193627   15.464    0.000   18.983    0.000 /usr/lib/python3.10/json/encoder.py:333(_iterencode_dict)
    52448   13.905    0.000   13.905    0.000 {built-in method builtins.print}
    17647    9.857    0.001    9.857    0.001 /usr/lib/python3.10/json/decoder.py:343(raw_decode)
    17647    8.376    0.000   38.150    0.002 /home/rs/surveys/SurveyFuncs.py:691(mm_setup)
    70596    6.681    0.000    6.872    0.000 {built-in method io.open}
467844/105882    6.297    0.000   42.340    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:85(_decode_map)
1394148/462479    5.986    0.000   31.131    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/decoder.py:93(_decode_pointer)
    52941    4.986    0.000   22.162    0.000 /usr/local/lib/python3.10/dist-packages/maxminddb/reader.py:39(__init__)
 16193630    4.632    0.000   24.292    0.000 /usr/lib/python3.10/json/encoder.py:413(_iterencode)
   828334    4.329    0.000    6.946    0.000 /usr/lib/python3/dist-packages/dateutil/parser/_parser.py:83(get_token)
```

Figure 5.13: Memory Profile after Refactoring

Figure 5.13 represents the memory profiling after refactoring and optimisation as depicted in the sections above. The runtime for data processing and analysis was reduced from 73816 seconds to 3750 seconds. In addition, after refactoring, the number of recursive calls were reduced, contributing to optimising the memory consumption as recursive methods can take up a lot of memory. The new DNS setup significantly improved the runtime of the program, and this might be due to the following factors:

- Typical DNS setups rely on servers supplied by ISPs that might be slow.
- Configurations for caching might not be adequately implemented by the ISP, which may contribute to slow lookups. [37]

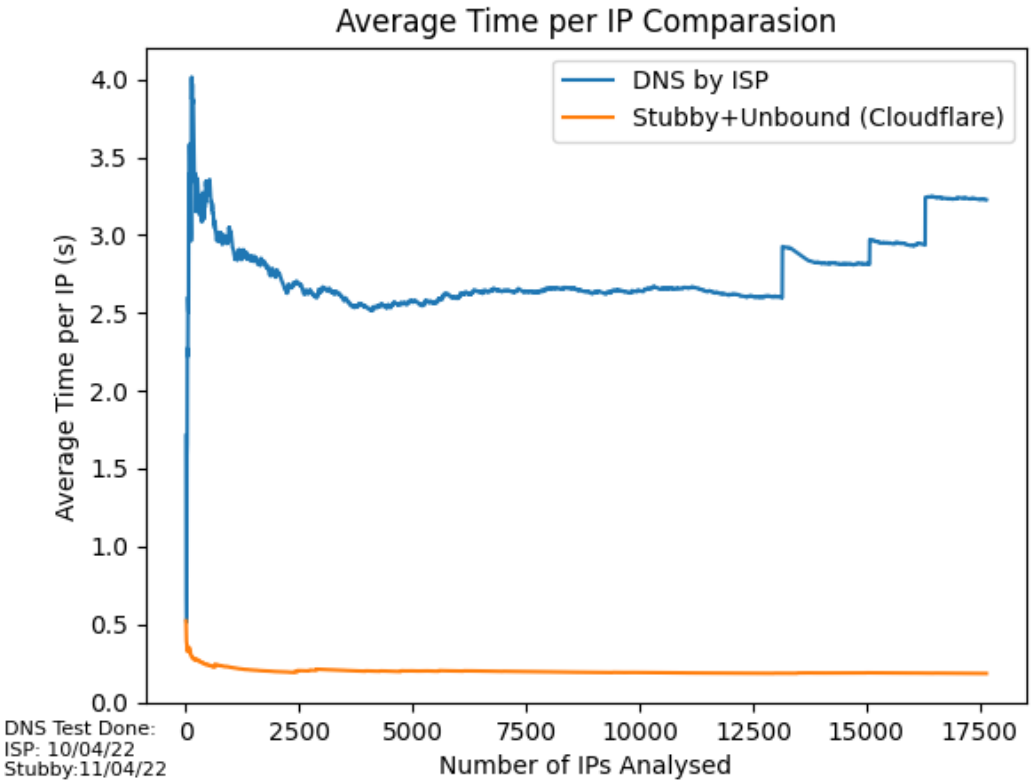

Figure 5.14: DNS Setup Comparisons

Figure 5.14 above shows the difference between the average time spent per IP when using the two DNS setups. The average time processing each IP was bought down to 0.15s from 3.4s when using the default setup. This might be due to the following reasons:

- Using Unbound, one can get more control over their DNS setup as it allows one to configure the DNS servers one might want to use.
- It will also cache all queries made for faster lookups the next time.
- Using Stubby allows one to use DNS over TLS for sending queries to resolvers. It allows for increased privacy.
- Since Unbound is not as advanced as Stubby and does not have the provision to use the same TLS connection for queries, a combination of the two can help speed things up. Stubby can use the same TLS connection to make multiple queries, saving the overhead and time to open up a new connection for each query. [17, 32]

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Both research and personal objectives were outlined in section 1.2 to guide the project and provide a means to complete the project in the time frame given. This chapter discusses the goals completed and provides conclusions on the work carried out.

6.1.1 Goals Revisted

One of the primary goals of this project was to migrate the code for the current surveying tool to Python3 and refactor it along the way to optimise it. The refactoring and migration process was carried out successfully, and there were significant improvements in the program by making use of new functionalities available in Python3. Since support for Python2 has depreciated, even though some may argue the benefits of Python2 over Python3 but since there is an industry-wide shift towards the adaptation of Python3, this would prove beneficial if other entities wish to use this program. In addition, the migration makes it more accessible and easy to run. While carrying out the refactoring process, the Python Enhancement Proposal 8 (PEP8) [43] was followed to standardise the code to industry standards to increase understanding of the program. Following this convention allowed to maintain a single coding style throughout the program amongst many scripts, which could benefit other entities later if they wish to make changes to the program to fit their needs. All technologies used for this project, like Maxmind, ZMap and ZGrab, were upgraded to their latest versions.

The program's methodologies and design should allow other entities or institutions to carry out scans for different populations. They can replicate the same work as carried out here with ease.

The program was updated to scan populations using data from the Maxmind database, and it also has provisions to do so with the Censys databases. Although it was not possible to upgrade the program to work with the updated versions of the Censys metadata due to Censys going commercial, attempts were made by contacting them to get some sample data to extend this work. However, unfortunately, Censys did not provide the data needed to upgrade this program.

Another goal of this project was to add port 853 (DNS over TLS) to the scans parameters. Code had to be written in Golang, and integration tests had to be run using the ZGrab integration tools available. However, it was unsuccessful due to a lack of programming experience in Golang and time constraints. Although a simple banner grab over port 853 using TLS was successfully added, it was not the most efficient way to go about it.

6.1.2 Final Remarks

Overall, all goals but adding port 853 were accomplished, and there is still widespread key reuse seen, as proven back in 2018 by Dr. Farrell. However, key reuse between SSH and other protocols that use TLS was not seen as was the same case in 2018 [9].

6.2 Future Work

This section discusses some of the possible extensions of this work.

6.2.1 Scanning other Countries and IPv6 address Space

Due to the time constraints, only a couple of scans for Ireland were able to be carried out, and data analysis was only done on one of the scans. Still, one could get a better understanding of some of the causes behind this key reuse by carrying out scans over a long period at regular intervals. While the accuracy of these scans is acceptable for this project, one could extend this work to distinguish between hosts that operate with more than one IPv4 address, i.e. multi-homed hosts. Currently, the program can not differentiate between multi-homed and single-homed hosts. Still, introducing some techniques to distinguish between the two would provide a more accurate picture of the key reuse scenario for a population. Another possible extension of this work would be to scan the IPv6 addresses space.

6.2.2 Adding Additional Protocols

An exciting extension would be to add additional protocols to scan for, like the MQ Telemetry Transport Protocol (MQTT) used by IoT devices and adopted widely. MQTT has provisions for using TLS, and since the number of IoT devices is growing exponentially, they have a reputation for being insecure. It would be interesting to see if there is key reuse across these protocols.

6.2.3 Database Management

A place where this work could be improved upon is by improving data management. Since a large amount of data is captured using JSON and each structure is highly sparse, one could look at integrating databases like Elasticsearch, which is an open-source NoSQL database that can be used to store unstructured data and query it with ease using SQL commands [7]. This could be highly beneficial for entities carrying out scans for a considerable period, gathering a lot of metadata.

Bibliography

[1] (2008). Installation — pandas 0.23.1 documentation. https://pandas.pydata.org/pandas-docs/version/0.23/install.html.

[2] Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., and Cooper, D. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280.

[3] Cotton, M., Eggert, L., Touch, D. J. D., Westerlund, M., and Cheshire, S. (2011). Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry. RFC 6335.

[4] DmitryFillo (2015). berserker_resolver · pypi. https://pypi.org/project/berserker_resolver/. (Accessed on 04/19/2022).

[5] Durumeric, Z., Wustrow, E., and Halderman, J. A. (2013). ZMap: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 605–620, Washington, D.C. USENIX Association.

[6] Durumeric, Z., Wustrow, E., and Halderman, J. A. (2015). Zgrab2. https://github.com/zmap/zgrab2.

[7] Elasticsearch (2010). Free and open search: The creators of elasticsearch, elk & kibana — elastic. https://www.elastic.co/. (Accessed on 04/12/2022).

[8] Farrell, S. (200). sethr07/surveys at rahul-01. https://github.com/sethr07/surveys/tree/rahul-01. (Accessed on 04/22/2022).

[9] Farrell, S. (2018a). Clusters of re-used keys. Cryptology ePrint Archive, Report 2018/299. https://ia.cr/2018/299.

[10] Farrell, S. (2018b). sftcd/surveys: Code for various survey-related stuff. https://github.com/sftcd/surveys. (Accessed on 04/22/2022).

[11] Gillis, A. S. (2021). Last Accessed 5th April.

[12] Halley, B. (2022). dnspython · pypi. https://pypi.org/project/dnspython/. (Accessed on 04/19/2022).

[13] Hanna, K. T. (2021). Syn scanning. Last Accessed 24th March 2022.

[14] Heninger, N., Durumeric, Z., Wustrow, E., and Halderman, J. A. (2012). Mining your ps and qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, Bellevue, WA. USENIX Association.

[15] Hoffman, P. E. (1999). SMTP Service Extension for Secure SMTP over TLS. RFC 2487.

[16] Klensin, D. J. C. (2008). Simple Mail Transfer Protocol. RFC 5321.

[17] Labs, N. (2022). Nlnet labs - unbound - about. https://www.nlnetlabs.nl/projects/unbound/about/#:~:text=Unbound%20is%20a%20validating%2C%20recursive,is%20more%20resilient%20than%20ever. (Accessed on 04/10/2022).

[18] LAWRENCE, C. (2021). The complete engineer's guide to code refactoring. https://www.stepsize.com/blog/the-ultimate-engineers-guide-to-refactoring. (Accessed on 04/09/2022).

[19] Lonvick, C. M. and Ylonen, T. (2006a). The Secure Shell (SSH) Connection Protocol. RFC 4254.

[20] Lonvick, C. M. and Ylonen, T. (2006b). The Secure Shell (SSH) Transport Layer Protocol. RFC 4253.

[21] MaxMind (2022). Geolite2 free geolocation data — maxmind developer portal. https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en. (Accessed on 04/23/2022).

[22] Melnikov, A. and Leiba, B. (2021). Internet Message Access Protocol (IMAP) - Version 4rev2. RFC 9051.

[23] Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2.

[24] Microsoft (2021a). Cipher suites in tls/ssl (schannel ssp) - win32 apps — microsoft docs. https://docs.microsoft.com/en-us/windows/win32/secauthn/cipher-suites-in-schannel. (Accessed on 04/13/2022).

[25] Microsoft (2021b). Public key infrastructure - win32 apps — microsoft docs. https://docs.microsoft.com/en-gb/windows/win32/seccertenroll/public-key-infrastructure?redirectedfrom=MSDN. (Accessed on 04/14/2022).

[26] Mishra, O. (2020). Transport layer security (tls) handshake - geeksforgeeks. https://www.geeksforgeeks.org/transport-layer-security-tls-handshake/. (Accessed on 04/13/2022).

[27] Moore, K. and Newman, C. (2018). Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access. RFC 8314.

[28] Moriarty, K. and Farrell, S. (2021). Deprecating TLS 1.0 and TLS 1.1. RFC 8996.

[29] MULDERS, M. (2022). The power of code refactoring: How to measure refactoring success. https://www.stepsize.com/blog/how-to-measure-refactoring-success. (Accessed on 04/19/2022).

[30] ODOGWU, C. (2021). What is network scanning and how does it work? Last Accessed 24th March 2022.

[31] Popov, A. (2015). Prohibiting RC4 Cipher Suites. RFC 7465.

[32] Project, D. P. (2021). Dns privacy daemon - stubby :: dnsprivacy.org. https://dnsprivacy.org/dns_privacy_daemon_-_stubby/. (Accessed on 04/18/2022).

[33] Python (2022). The python profilers — python 3.10.4 documentation. https://docs.python.org/3/library/profile.html. (Accessed on 04/19/2022).

[34] Rescorla, E. (2000). HTTP Over TLS. RFC 2818.

[35] Rescorla, E. (2018). The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446.

[36] Rose, D. M. T. and Myers, J. G. (1996). Post Office Protocol - Version 3. RFC 1939.

[37] Rubenking, N. J. (2022). How (and why) to change your dns server. https://uk.pcmag.com/security/138870/how-and-why-to-change-your-dns-server. (Accessed on 04/18/2022).

[38] Scheitle, Q. (oct). blechschmidt/massdns: A high-performance dns stub resolver for bulk lookups and reconnaissance (subdomain enumeration). https://github.com/blechschmidt/massdns. (Accessed on 04/19/2022).

[39] Sheffer, Y., Holz, R., and Saint-Andre, P. (2015). Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). RFC 7457.

[40] Simko, C. (2019). Learn ssh keys in minutes. https://www.foxpass.com/blog/learn-ssh-keys-in-minutes/. (Accessed on 04/13/2022).

[41] Turner, S. (2011). Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms. RFC 6151.

[42] Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

[43] van Rossum, G., Warsaw, B., and Coghlan, N. (2001). Style guide for Python code. PEP 8.

[44] Williams, M. (2017). Inside the russian hack of yahoo: How they did it — cso online. `https://www.csoonline.com/article/3180762/inside-the-russian-hack-of-yahoo-how-they-did-it.html`. (Accessed on 04/14/2022).

[45] Wireshark (1988). Chapter 1. introduction. `https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs`. (Accessed on 04/10/2022).

Appendix A

Sample Scan Data


```
  s{
"ip": "XX.XX.XX.XX",
"writer": "FreshGrab.py",
"p22": {
  "ip": "XX.XX.XX.XX",
  "data": {
    "ssh": {
      "status": "connection-timeout",
      "protocol": "ssh",
      "result": {},
      "timestamp": "2022-04-02T01:40:09Z",
      "error": "dial tcp XX.XX.XX.XX:22: connect: connection refused"
    }
  }
},
"duration": 0.5397200584411621,
"average": 4.158559069103385,
"p25": {
  "ip": "XX.XX.XX.XX",
  "data": {
    "smtp": {
      "status": "application-error",
      "protocol": "smtp",
      "result": {
        "banner": "XXXXX\r\n",
        "starttls": "454 4.3.3 TLS not available after start\r\n"
      },
```

```
      "timestamp": "2022-04-02T01:40:09Z",
      "error": "SMTP error code 454 returned from STARTTLS command (454 4.3.3 TLS not
          available after start)"
    }
  }
},
"p110": {
  "ip": "XX.XX.XX.XX",
  "data": {
    "pop3": {
      "status": "connection-timeout",
      "protocol": "pop3",
      "timestamp": "2022-04-02T01:40:09Z",
      "error": "dial tcp XX.XX.XX.XX:110: connect: no route to host"
    }
  }
},
"p143": {
  "ip": "XX.XX.XX.XX",
  "data": {
    "imap": {
      "status": "connection-timeout",
      "protocol": "imap",
      "timestamp": "2022-04-02T01:40:09Z",
      "error": "dial tcp XX.XX.XX.XX:143: connect: no route to host"
    }
  }
},
"p443": {
  "ip": "XX.XX.XX.XX",
  "data": {
    "http": {
      "status": "success",
      "protocol": "http",
      "result": {
        "response": {
        }
        }
          }
        }
```

```
      },
      "timestamp": "2022-04-02T01:40:09Z"
    }
  },
  "p587": {
    "ip": "XX.XX.XX.XX",
    "data": {
      "smtp": {
        "status": "connection-timeout",
        "protocol": "smtp",
        "timestamp": "2022-04-02T01:40:10Z",
        "error": "dial tcp XX.XX.XX.XX:587: connect: no route to host"
      }
    }
  },
  "p993": {
    "ip": "XX.XX.XX.XX",
    "data": {
      "imap": {
        "status": "connection-timeout",
        "protocol": "imap",
        "timestamp": "2022-04-02T01:40:10Z",
        "error": "dial tcp XX.XX.XX.XX:993: connect: no route to host"
      }
    }
  }
}
```