HERIOT WATT UNIVERSITY

BUILDING AND TESTING A WHITE-HAT BOTNET CHECKER FOR EVALUATING THE EFFECTIVENESS OF MIRAI AGAINST OTHER BOTNET MALWARE

FINAL YEAR DISSERTATION

BSc COMPUTER SCIENCE

Ryan Shah

supervised by

Dr. Mike Just

Declaration

I, Ryan Shah confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: Ryan Shah

Date: 26th November 2017

Abstract

A botnet is a network of infected devices, that are usually remotely controlled, used to perform large-scale attacks such as a distributed denial-of-service (DDoS) attack and conduct other illegal activities.

Mirai is a self-propagating, IoT ("Internet of Things") botnet virus that infects out-of-date and vulnerable IoT devices running the Linux operating system. It does this by identifying a vulnerable device by rooting into them using a list of common default usernames and passwords. These infected devices are now effectively "bots" and are under control by the command and control program that spreads the virus. They will continue to run as normal until asked to perform a specific task. It is an effective botnet as these IoT devices are vulnerable and easily infected and for example, since there are so many of them, it is possible to generate enormous amounts of throughput.

In this dissertation, I will investigate why some state that the Mirai botnet is more effective than other major botnets in the past and why it is effectively unstoppable. I will be creating a program which will allow system administrators to scan their networks for potential vulnerable devices, and attempt to patch the device. Whilst comparing the program's ability to prevent Mirai through a known vulnerability with other programs that attempt to stop Mirai, I will be evaluating how effective it is for helping system administrators patch this vulnerability.

Contents

1. Introduction

Botnets have been a growing problem for a long time and although we constantly find ways to prevent or stop them altogether, new methods of infecting devices (*attack vectors*) are constantly on the rise. In early September 2016 the botnet named Mirai first appeared, prominently announcing its existence by flooding a security journalist's website with traffic from a botnet consisting of IoT devices. This attack had a large impact on millions of internet users by overwhelming Dyn. After this attack, Mirai was made opensource on Github and was found to be very flexible and adaptable. This opened up the possibility for others to create different strains of Mirai to infect new vulnerable IoT devices.

Several studies have investigated the anatomy of botnets and different attack vectors to infect vulnerable devices, but since Mirai is very adapatable; it is hard to find and mitigate infections. A study in September 2016 identified that Mirai had disrupted internet services for hundreds of thousands of Deutsche Telekom customers in Germany, as well as thousands of routers in the UK. It was also discovered that many CCTV cameras as well as other commonly used IoT devices are vulnerable to being infected by Mirai.

Since IoT devices have a substantial amount of insecurity issues, billions of units are left vulnerable to all sorts of malware. Mirai tends to be a favoured choice nowadays, both due to the vulnerable nature of IoT devices, with their security not properly integrated into their development, as well as the design of its source code. With Mirai's flexible design, it is what hackers tend to use; applying different strains of the virus for use in different campaigns and few attempts have been made to try and mitigate IoT devices being infected, such as ensuring people limit the "smart" devices in their homes that access the internet for no important reason.

1.1   Aims and Objectives

The goal of this project is to design a program, Miraihilate, to patch a known infection vector of Mirai malware, and to evaluate its effectiveness through means of different metrics. The infection vector takes advantage of a security vulnerability in IoT device software, which involves exploiting default combinations of root usernames and passwords. It is not intended to be the first of programs that counter botnet malware, but will aim to prevent current and future strains of the Mirai virus from using the known infection vector to corrupt IoT devices.

1.1.1 Aims

- **Research Aims**

  - To understand how botnets work and differentiate between the operation of a regular botnet and an IoT botnet

  - To research known security vulnerabilities in the general population of IoT devices, and which of those vulnerabilities can be patched

  - To investigate why Mirai is much more effective to other well known botnets

- **Project Aims**

  - To develop a program to patch a security vulnerability in IoT devices to further diminish Mirai's target device population

  - To research other countermeasures to Mirai malware

The first project aim listed above, "*To develop a program to patch a security vulnerability in IOT devices...*", is the main aim of this project, and the other aims are supportive of achieving it. The other aims are used to investigate the effectiveness of Mirai to other botnets, and to help develop the program to patch the security vulnerability, as well as any other possible vulnerabilities that may exist.

1.1.2 Objectives

- **Research**

  - Research previous *open-source* botnet malware to compare with Mirai malware

  - Investigate how other botnet malware removal programs operate

  - Research known vulnerabilities in IoT devices

- **Design and Functionality**

  - Produce a program with a Graphical User Interface (GUI), which can be run on a standard linux operating system

  - Develop a secure, strict access policy for the program

  - Develop logging and debugging tools, and unit tests for the program

- Develop an effective scanning tool which can be manipulated by the user

- Create a simple, yet effective design

- **Testing and Performance**

  - Analyse the effect on time to complete scans based on the size of the IP address range determined by its CIDR suffix

    * Classless inter-domain routing (CIDR) is a set of Internet protocol (IP) standards that is used to create unique identifiers for networks and individual devices[1]

  - Trial the program on an artificial network as well as a small private network

  - Test the program to find vulnerabilities in different Linux operating systems and possibly others

- **Documentation and Evaluation**

  - Develop an in-depth user guide/manual for the program to help system administrators with anything they might not understand when using the program

  - Analyse evaluation metrics and write a report describing the results found

2. Background

2.1 Internet of Things

The "Internet of Things" (IoT) is a concept that has the potential to impact how we live from day to day, but what impact does it have on people, if any? The concept surrounds the connection of any device, so long as it can be turned on and off, to the Internet and other connected devices. The term IoT *generally* refers to the "networked interconnection of everyday objects, which are often equipped with ubiquitous intelligence"[2].

In this collection there are a vast amount of objects, which include smart home systems, self-driving cars, and even wearable devices. Objects and/or devices that have in-built sensors are connected to an IoT platform, which collects the data from them and shares the most important information with other applications developed to address specific user needs.
However, with the sheer number of these devices constantly increasing, newer problems are introduced relating to them. More specifically, the security of these devices becomes a greater issue, as new vulnerabilities[3] are created and identified.

With these vulnerabilities left open to the Internet, and some not being identified, threats to the security of these devices is an important issue that needs to be addressed. Most of these devices dont directly interface with the user(s) associated with the device, and therefore it becomes difficult to identify any abnormal behaviour.
Recent events show a trend where IoT devices are becoming increasingly involved in cyber-attacks, with reports indicating IoT attacks have grown 280% in the first half of 2017[4]. Therefore, more secure communication with the Internet and eachother to ensure safety of keeping and using these devices in people's day-to-day routines.

2.2 Botnets

A botnet is a network of inter-connected and infected devices. These devices, referred to as bots or 'zombies', become infected by malware distributed to purposely take control of them. Botnets are then primarily used to distribute the same or other malware, or launch Distributed Denial-of-Service (DDoS) attacks. The individual or group that develops and/or runs the botnet is referred to as the botmaster. They constantly add bots to the network by distributing bot malware

to infect new devices.

2.2.1 Botnet Architectures

Botnet mainly use a client-server architecture, but some instead use a peer-to-peer (P2P) architecture[5]. In a client-server architecture, there is a centralized computer that issues commands to, and receives information from, the infected bots. It frequently consists of several servers and other technical components. Bots report back to the botnet's command and control (C&C) server, whilst the C&C issues instructions to the bots to perform illegitimate tasks. Unlike the client-server architecture, a P2P architecture is a *decentralized* network of bots - with no command and control server. The bots act as both a command distribution server, as well as a client that receives commands. The random structure of the botnet further obfuscates itself and increases protection against it being taken down. Although they cannot be easily identified, the botmaster cannot easily monitor command delivery.

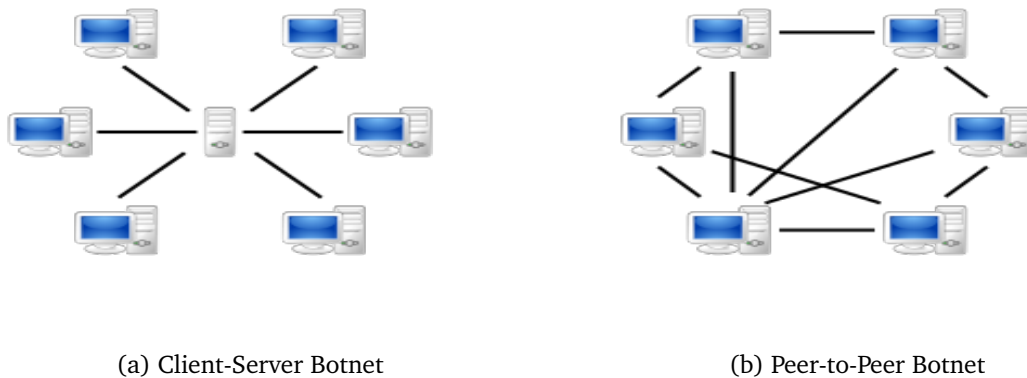
(a) Client-Server Botnet　　　　　　　　　　(b) Peer-to-Peer Botnet

Figure 2.1: Client-Server and Peer-to-Peer Botnets[6]

2.2.2 Classification

Command and control botnets can be classed under three main types[5]:

1. Internet Relay Chat (IRC) botnets

2. HTTP botnets

3. P2P botnets

IRC-based botnets[7] use a *push-based* model in which the botmaster pushes new commands to the botnet, through the use of an IRC channel, where bots respond directly to the commands given. HTTP-based botnets use a *pull-based* model of communication, where the bots individually poll the command and control server to request new commands to execute.

P2P botnets are *decentralized*, where the bots themselves operate as a command distribution server, as well as the client that receives commands.

Many botnets use a DNS technique called **fast flux**[8]. This allows the botnet to hide their central command and control server. This does not apply to P2P botnets due to the nature of the botnet architecture. This adds an extra layer of protection against takedowns.

2.2.3   Infection Vectors

In order to create a substantial botnet, a botmaster must initially distribute bot malware to the device. Then bots can also individually spread the malware to other devices, as well as the botmaster themselves[9].

While some malware while have a direct impact on the user of the vulnerable device, botnet malware can take on different levels of visibility. However, it tends to attempt running silently or with minimal disruption to avoid detection.

Botnets recruit additional bots through a variety of different attack/infection vectors. Pathways for infection include, but are not limited to: website vulnerabilities and silent drive-bys, Remote Administration Trojan (RAT) malware and weak authentication. Once access has been obtained, the result involves the bot malware being installed and allows the botmaster full remote control of the device.

Once the device is infected, it may self-propagate the botnet and re-distribute the bot malware to other vulnerable devices.

2.2.4   Threats Surrounding Botnets

Given the potential scale of botnets and vast amounts of damage they can cause, prevention of threats posed by them is imperative.

Research has shown[10] that a botnet network consisting of a small number of infected devices doesn't *necessarily* pose much of a threat, but some of the largest botnets range from thousands to

millions of bots. Such large botnets have the desired resources to perform malicious actions on a high-scale[11], such as:

- Mass email spam, flooding millions of inboxes in a matter of seconds

- DDoS attacks, crashing entire websites

- Parallel brute-force hacking techniques

- Identity theft and internet fraud

2.2.5   Previous Botnet Threats

Since the progression of network attacks, botnets have become increasingly powerful. There have been many large-scale botnets, but there were some which had incredible impacts on regular internet users[12].

One such botnet, first discovered in 2008, was called Grum. This bot was one of the largest in terms of distributing spam email traffic. It was responsible for up to 26% of the worlds spam email traffic, capable of emitting up to 40 billion messages a day. Another botnet, which was similar in the way that it operated, was called Windigo. This only emitted around 35 million emails a day but is infamous because it infected a large number of Linux servers. This raised a lot of security questions due to the fact that at the time more than 60% of web servers use a Linux operating system.

ZeroAccess was also another powerful botnet in history. Unlike others, this botnet was used for click fraud and bitcoin mining. It used its mass of distributed devices for CPU and GPU bitcoin mining and for performing fraudulent clicks on pay-per-click ad revenue websites to make money illegally. It was a very resilient and powerful P2P botnet.

Two botnets named Metulji and Mariposa[13] also make it on the list of infamous botnets in history. These two botnets operate differently to some of the others. This is because they were developed from a *Butterfly Framework*, a kit used for creating botnets. Both botnets managed to enslave greater than 10 million devices each. Metulji was thought to have stolen millions of dollars with of private passwords, credit card details and social security numbers. This made it a very scary botnet to individuals whose personal information had been compromised. Both botnets, however, were

shut down. One flaw in the Butterfly Framework, was that recrods were kept for who paid for the service, unveiling the creators of both botnets.

2.2.6   Detection and Prevention

Detecting botnets has proven to be a difficult task, as bots are designed to infect and operate without user consent or knowledge. However, they are not completely invisible[14].

One indicator that a device may be part of a botnet involves extremely high CPU usage or slow computing speeds. Problems with Internet access can also be another indicator alongside this. Preventing this issue can include keeping all software up-to-date with their respective security patches, as well as monitoring network activity so that irregular behaviour can be identified[15].

Outgoing spikes in traffic from specific ports can also be monitored to detect possible botnet activity. Ports such as *Port 6667* (IRC), *Port 25* (Email Spam) and *Port 1080* (Proxy Servers) are usually the most used.
As well as this high outgoing SMTP traffic, found as a result of transmitting spam, is another possible indicator of infections.

With these detection procedures in mind however, symptoms that are indicative of bot infections can also be signs of regular malware infections or network problems. Therefore, they should not be taken as a sure sign that a device has been compromised by bot malware.
There is no sure fire way of preventing being infected, however regulating network activity, keeping programs up-to-date and being vigilant whilst using the internet are preventative methods against being infected by bot malware.
Anti-botnet tools provide botnet detection procedures by finding and blocking known bot viruses before infection can occur. Network-based and Host-based *Intrusion Detection Systems* (NIDS/HIDS) and network sniffers, as well as anti-bot programs can provide more sophisticated measures for detection, prevention and removal of bot malware, and provide security policies on outgoing communication from the IoT devices[16].

For organisations, on a larger scale, removing botnet malware often requires disabling or taking down the command and control server operating the botnet. An example of this being done in the past was Microsoft's campaign against the Zeus botnet[17].

2.3   Denial of Service

A Denial-of-Service (DoS)[18] attack is a type of network attack, that is designed to take down a network. It does this by flooding it with useless traffic so it crashes. It makes it extremely difficult or impossible for legitimate users to use resources. Whilst an attack that crashes a server can be successful, it can be solved by rebooting the system. However, *flooding* attacks can be a more difficult task for recovery.

The main motive behind DoS attacks is to cause harm to an individual or organisation. However, DoS attacks are usually not large enough to cause substantial damage to an organisation. Therefore, many high-profile DoS attacks are distributed. This means that the traffic outbound to a network is directed from multiple systems performing the attack. An attack originating from multiple sources is far more difficult to detect and defend against because it is harder to differentiate the traffic with malicious intent from legitimate traffic into the network.

2.3.1   Distributed Denial of Service

A Distributed Denial-of-Service (DDoS)[18] attack involves a series of network-connected devices flooding a network with useless traffic, compared with a DoS attack usually originating from a single source.
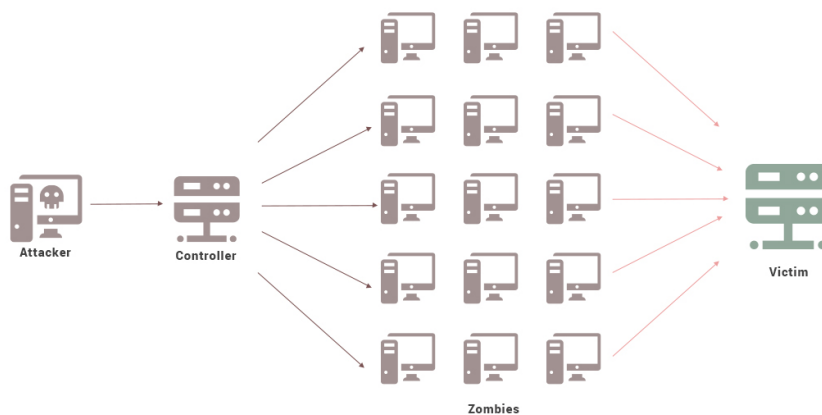

Figure 2.2: Visualising a DDoS Attack[19]

In a DDoS attack, the connected devices are usually infected by some sort of malware and operate as part of a botnet. Command and control servers are used to direct the actions of the infected devices in the botnet and dictate what type of attack to launch, data to transmit and what system

or network is the target of the attack.

After the first detected DoS attack was launched in the 1970s, DoS and DDoS attacks have remained the most damaging and persistent cyber-attacks to date. One of the first large-scale DDoS attacks occurred in 1999, when a hacker disabled the University of Minnesota's network for over two days. It allowed the attacker to send a DoS instruction to a few command and control servers, which the instructed hundreds of bots to flood the target IP address.

As technology advanced and hackers began to focus more on DDoS attacks, the nature of distributing the attack became significantly more powerful. This ultimately lead to this network attacking technique to become an extremely formidable weapon, and hackers taking on much larger targets.
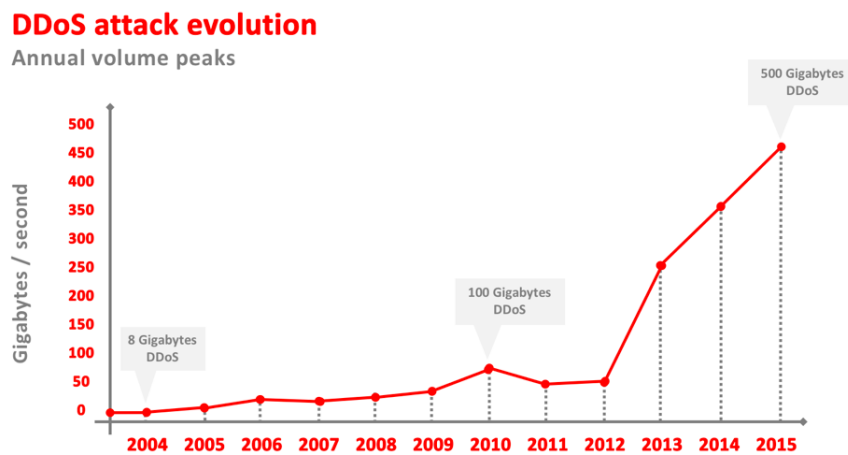
Figure 2.3: DDoS Attack Evolution 2004-15[20]

Recent years have lead to an exponential increase[21] in the number of DDoS attacks, driven by ever-changing motivations and complex, creative ideas.

In 2010 DDoS attacks had become political, with groups such as Anonymous attacking global payment sites. The tool most commonly used in these attacks was known as the Low Orbit Ion Cannon (LOIC). It was originally developed for server stress-testing, but hackers adjusted the software to bring down large servers.

In 2016, DDoS threats evolved to become more materialized. Cyber attacks launched from multiple connected devices were turned into botnets. This propelled the power of attacks to reach around 1TBps of flooding traffic.

2.3.2 Network Types

There are two types of DDoS **attack networks**[22]: the *Agent-Handler* model and the *Internet Relay Chat* (IRC) model.

The Agent-Handler model consists of: clients - where the attacker communicates with the rest of the DDoS system, handlers - which are software packages located on the Internet that the clients use to communicate with the agents, and agents - which carry out the attack(s).

The IRC model operates through the use of an online chatting system. It allows multiple parties to connect and message eachother in real time. IRC servers are located throughout the Internet and have public and private channels. Public channels allow multiple users to chat and share content with eachother, and users of that channel can see all other users and messages within the channel. Private channels, however, are setup by individuals to communicate directly with specific users. This protects the names and messages of the users from those who do not have access to the channel.
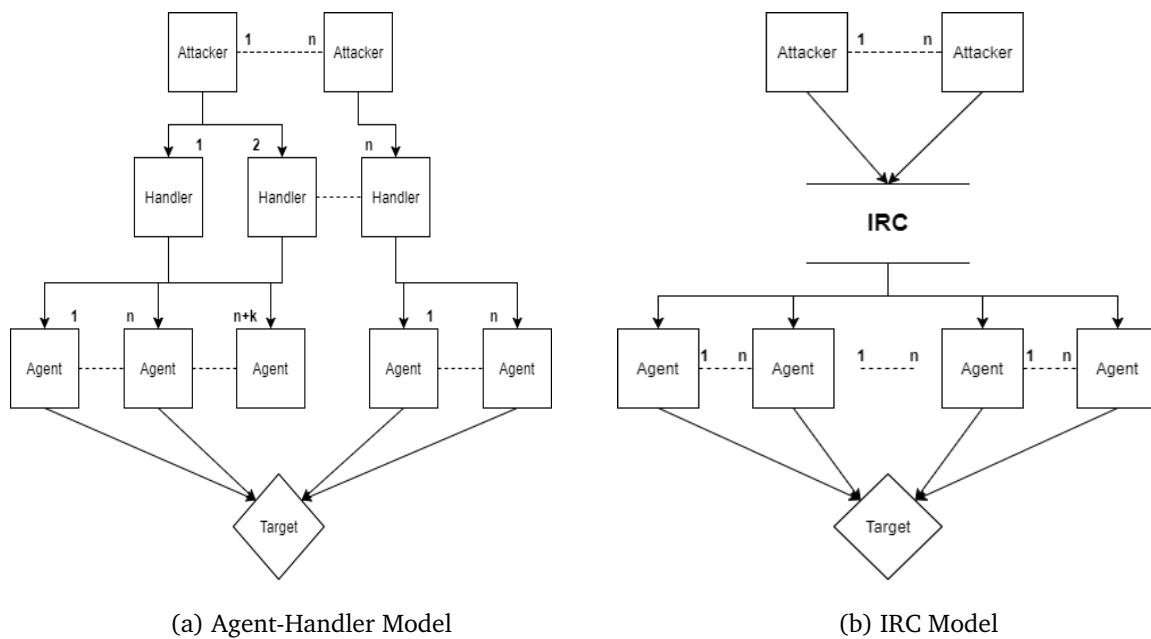

(a) Agent-Handler Model   (b) IRC Model

Figure 2.4: Agent-Handler and IRC Models[22]

2.3.3 Motives

It is often difficult to establish an attackers motives when they perform a DDoS attack, and sometimes people never learn what motivated the attack, but any organisation or individual can be

targeted.

One motive that drives attackers to perform a DDoS attack is ideological or political reasons. They target websites, individuals and/or organisations they disagree with ideologically.

Businesses can also use DDoS attacks to strategically impair or take down competing business websites, possibly keeping them from participating in a significant business-related event.

Attackers can also DDoS or threaten with DDoS attacks as a means of extorting money or other assets from their targets.

Another motive is cyber warfare. Authorized attacks, such as those authorized by a government official, can be used to cripple opposing websites or even an enemy country's infrastructure.

Finally, boredom is another main motive for performing a DDoS attack. Individuals may use pre-coded scripts to launch attacks, or even use a free minor DDoS service, looking for a small adrenaline rush or even as an attempt to impress friend groups.

2.3.4 Attack Vectors

In order to further understand how a DDoS works and why it can be effectively unstoppable, we have to take a deeper look into the taxonomy of a DDoS attack.

Although there are a wide array of DDoS attacks, there are no specific methods a single attack would use - it all depends on the aim of the attack. Specht and Lee[22] proposed a taxonomy of the main DDoS attack vectors and describes that there are two primary classes of attacks: *bandwidth depletion* and *resource depletion*.
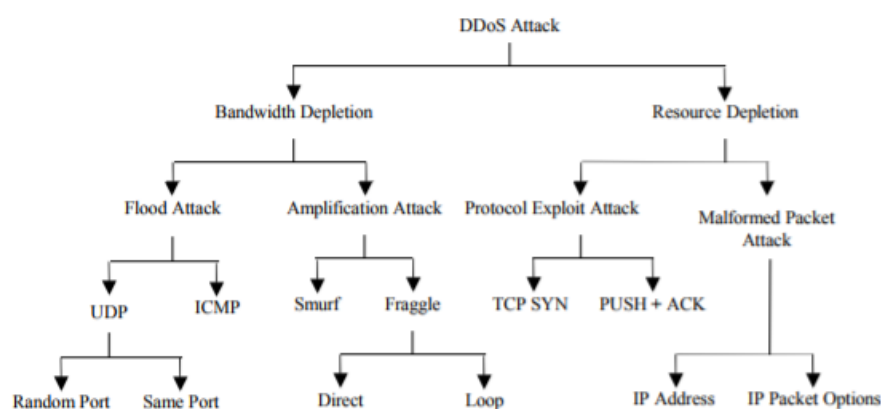
Figure 2.5: Taxonomy of a DDoS Attack

The most common types of attacks performed include:

- UDP, ICMP and SYN Floods

- TCP/SYN

- Zombie Attack

Zombie attacks are not described in the taxonomy provided by Specht and Lee[22], however they are an important mention. This type of attack occurs when non-spoofed connections overload services, causing network paralysis. These are extremely difficult to stop unless the target somehow possesses a form of behavioural mitigation technology.

2.3.5 Prevention and Mitigation

To this day, there have been many proposals and solutions to mitigate the *effects* of a DDoS attack, however there are no methods to protect against all forms of DDoS attacks. This is because that new types of attacks are continually being developed to bypass the constant countermeasures employed by victims.
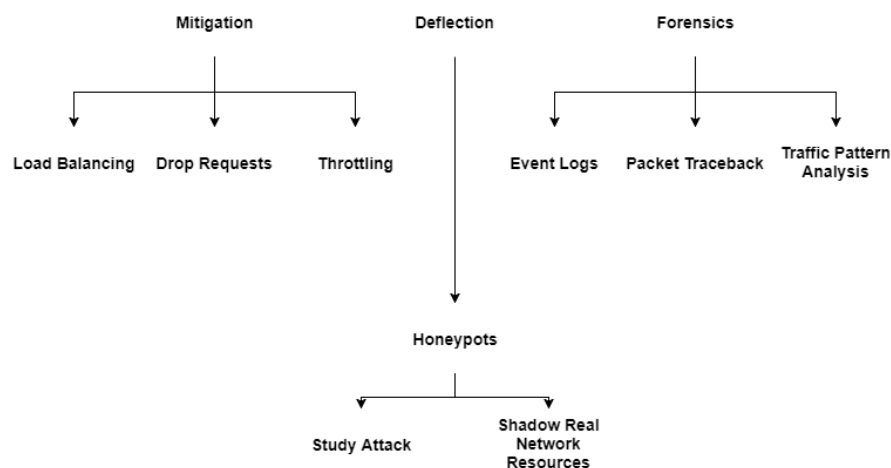
Figure 2.6: Countermeasures Employed Against DDoS Attacks[22]

The tree above shows three main categories of preventing and mitigating a DDoS attack, as well as examples of procedures in each category. These however are not the only countermeasures employed, as described above there are no methods to protect against all forms of DDoS attacks due to the continuous development and advancement of attack vectors.

2.4   IoT Devices and Botnets

The increasing popularity of IoT devices, especially in the home, have made them a popular target and powerful amplification platform for cyber-attacks[23].

In September 2016, a network security blog owned by Brian Krebs was hit with 620Gbps of traffic. At around the same time an even bigger attack was performed peaking at 1.1Tbps, making it one of the biggest attacks ever performed. Both of these attacks were performed by an IoT botnet named Mirai.

2.4.1   Vulnerability

Amongst these major attacks, there has been a frequent recurrence of attacks involving IoT devices, clearly demonstrating they are extremely vulnerable in terms of security. The large quantity of these devices around the world, as well as being extremely vulnerable, has attracted a new era of botnet malware.

It comes down to three primary factors, which determine the vulnerability of IoT devices:

- An embedded or simplified OS, such as Linux, which is comparatively easier to compromise

- Lack of capacity and processing power for standard security capabilities

- Aspects of hardware and software are reused to save engineering time, resulting in default passwords

2.5   IoT Botnets

Compared to a traditional botnet, an IoT botnet is comprised of hacked computers, smart appliances and other IoT devices co-opted for illegal purposes. Bots in an IoT botnet have been coined the term *thingbot*[24], to refer to devices other than computers that have been compromised.

2.5.1   Are they more of a threat?

The attack vectors and motives of an IoT botnet are not considered more of a threat than a regular botnet, but because of the vast number of vulnerable IoT devices available to be exploited, an IoT botnet has the potential to be become a much larger threat than a regular one. By 2016, the

number of IoT devices attached to the Internet was nearly double the number of users on the Internet itself, demonstrating a rapid influx of IoT devices. IoT botnets have also introduced the Internet to 1TBps DDoS attacks.

As well as this, because a lot of IoT devices do not directly interface with its end-user(s), it can be extremely hard to detect whether or not a device has been compromised. Therefore, it makes it harder to detect an IoT botnet while its not performing a task. Most IoT malware also resides in the device's temporary memory (RAM) and do not use reflection or amplification techniques to launch attacks, making it much more difficult to recognize and mitigate IoT botnet DDoS attacks[25].

2.5.2 Mirai

Mirai is IoT botnet malware that was responsible for taking down security journalist, Brian Kreb's website in September 2016, as well as Dyn[26]. The magnitude of the attack, being one of the largest ever recorded, made Mirai a high-profile botnet malware.
Although it is not the first, or only IoT botnet, it is the first open-source IoT botnet after its source code was leaked on Github. It's command and control code is written in the language Go, while its bot malware is coded in C. Like most botnets, Mirai was built for two core purposes[27]:

- Locate and compromise IoT devices to increase the capacity of the botnet

- Launch DDoS attacks based on instructions received from its remote command and control center

2.5.3 Infection

Mirai malware is initially spread by first entering a rapid scanning phase, where it asynchronously sends requests to pseudorandom IPv4 addresses on Telnet TCP ports. It had a hardcoded blacklist of IP addresses to avoid, such as the US Department of Defense, to further obfuscate its existence[26].
The purpose of this scanning phase is to locate vulnerable IoT devices that could be remotely accessed by simple passwords, usually factory-default usernames and passwords. Mirai uses a brute-force technique for guessing passwords, such as *admin/admin*, based on a predefined list. Some strains of Mirai have also been shown to self-propagate the malware, where infected bots will also perform scanning and infection.

2.5.4 Prevention and Mitigation

The prevention and mitigation procedures are the same as a regular botnet, however due to the diversity and higher capacities of IoT botnets such as Mirai, tools must evolve to become more specific. Several programs have already been released to scan a device to check for a vulnerability to Mirai, such as the Mirai Scanner by Incapsula[28]. Although these programs address general Mirai vulnerabilities, they do not attempt to fully address the *future* strains of Mirai.

2.6 Summary

Overall, it can be seen in comparison to IoT botnets that a regular botnet can be exactly as damaging. However through research it is clear that, due to the large current size of IoT and the prediction of the increase in IoT devices in upcoming years, that the possible capacity of an IoT botnet such as Mirai will supercede the capacities of botnets in the past. This suggests that with the influx of larger quantities of resources available to a botnet, attacks will undoubtedly become much more powerful and *effectively* unstoppable.

Ultimately, the program I shall develop will focus on the primary functions of all strains of Mirai malware, and attempt to prevent and mitigate the primary infection vectors Mirai uses. The program will have the ability to warn devices and/or modify the default user/password vulnerability, whilst possible adding a security policy to limit remote address access to the devices to local networks.

3. Program Design

3.1 System Structure

The system, named *Miraihilate,* is a software program intended to be used by system administrators to scan their private network(s) to detect a vulnerability in Linux devices that Mirai malware exploits, and attempts to patch this vulnerability if it is found. It will enable users of the system to perform simple operations such as a quick scan on the network, as well as providing advanced scanning options and other utilities for the user.

The functionality of Miraihilate will be separated into two parts, the *back-end* scripts and the *front-end* client interface, as well as coming with a setup utility to setup the program with an initial administrator user.

As well as just the interface that the user interacts with, Miraihilate will also implement a secure role-based entry system as well as a logging system. Because of Miraihilate's potential to access IoT device root accounts to patch the vulnerability to Mirai malware, it is eseential to create a secure access control system as well as securing access/scan logs to prevent malicious activites such as:

- Unauthorised access of scanning utilities

- Unauthorised accessand reading of log files

3.1.1 User Access Control

Before a user can access the system, they must login to access the features. The login functionality will use BCrypt for hasing passwords, which is a default password hashing algorithm for some Linux distributions, such as OpenBSD[29] and SUSE Linux.

It incorporates a salt to protect against rainbow tables and is also adaptive to remain resistant to brute-force attacks, by increasing the iteration count (work factor) to make it slower. When using a suitable work factor and a secure password, it is practically infeasible to try every possible combination to crack the password. For Miraihilate, I will use a work factor of 12 and passwords **must** contain at least one uppercase letter and a symbol.

The form of access control used in Miraihilate will be role-based access control, which will regulate the tasks that certain users can perform. For the initial version of the program, there will be two

roles: operator and administrator. Operators will be able to perform all operations that an administrator can with the difference being they cannot read log files that are not their own, nor can they add and remove other users from Miraihilate.

3.1.2 Log System

There will be two types of logs recorded in the system, which are access and scan logs. Access logs will be stored with a timestamp of when a user has accessed the system, to help identify who was on the system in case a problem arises. Scan logs will contain the start and end timestamps of the scan, as well as raw data about the scan that has taken place such as who started the scan and the number of vulnerable devices found in a given network range.

3.1.3 Database Structure

Before working on the implementation, I prepared the the database by creating an ER diagram to describe its structure and key relations between the data. Below is the ER diagram for the database that Miraihilate will use:
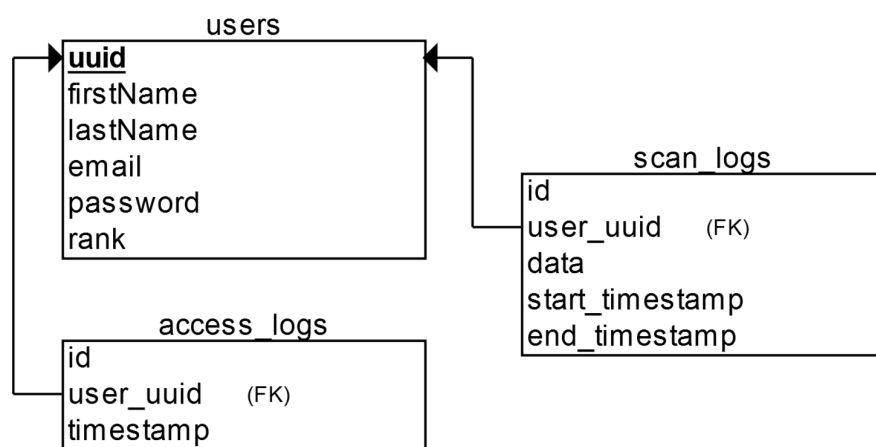

Figure 3.1: Miraihilate Database ER Diagram

The *data* field described in the above ER diagram will store raw HTML data, which will then be displayed in the HTML client. Since the database will not allow script tag characters, I will encase all disallowed characters with a backslash and then remove them upon retrieval in the client before being displayed.

23

3.2   Interface Decisions

Before starting to develop and implement Miraihilate, I first designed wireframe mockups to decide what the interface would look like before implementation. The wireframes were designed in Balsamiq, and will be used to gain initial feedback on the layout of the design rather than usability, before designing a working prototype for usability feedback.

3.2.1   Mockup Design

The mockups that I designed for this initial feedback are of primary components of the system which users have the most interaction with. This includes:

- Client Interface

- Advanced Scanning

- Scan Logs

3.2.2   Feedback Survey

Recruitment

To conduct the feedback survey, I recruited participants who are studying Computer Science or a similar discipline, as well as others who have moderate confidence with computers and general applications such as word processing applications. I did this by confronting potential participants from within the MACS department at Heriot-Watt university and before asking if they would take a short amount of time to participate in the study, I asked what they were studying.

After finding a candidate willing to participate in the feedback survey they had to first sign a consent form, shown in Appendix A.1, informing them about the aims of the survey as well as to notify that they can withdraw their results at any time. They were then given three question sheets shown in Appendices A.2-4 to answer, each of which had images of the respective mockup being surveyed.

Initial Feedback Analysis

Explain feedback I gained...

3.2.3 Pre-Implementation

To reflect on the feedback survey results, I can conclude that there were minimal comments made to the structure of the design. Because the designs were only mockups, the feedback is only intended for helping with designing the interfaces for the client program before conducting a final usability study on a working prototype. Overall, based on the analysis of the feedback I gained, there will be no major changes to the interfaces, aside from reducing the amount of whitespace. One comment however was that it would have been better to show how the scan result text would be shown. This is not a major factor for feedback on the structure of the interface, but this factor will be more evident in the usability study on the working prototype.

4. Implementation

4.1 Sofware Decisions

Before starting development, I first had to decide upon platforms, programming languages and programs that I will use during the development process.

4.1.1 Platform and Portability

Since the program primarily handles IoT devices running embedded versions of the Linux operating system, it makes the most sense to ensure it can run on Linux as well as macOS. The program could be able to run on Windows, but is not entirely appropriate as the program in its first iteration will not focus on Windows OS IoT devices.

When considering the portability of the application, I took into account that not all system administrators will use a Linux operating system from within their organisation. As well as this, I took into account the programming languages that I will use that help aid with making this a cross-platform application.

4.1.2 Programming Languages

The back-end functionality of the program will use Python, whilst the front-end client interface will be written in Java. This is primarily because of SSH and Telnet libraries having much better documentation, and are much more defined than using similar libraries in Java. Doing this will ease the development process, by separating the primary functions of the program from its interface, as well as have the potential for backend scripts to be collated and released as an open-source Python package. To be able to execute the Python code in Java, I wrote a small code executor which executes a local python script within the application (or its directory) and returns its output.

```
Process p = Runtime.getRuntime().exec("python3 <file>
    <args..>");
BufferedReader stdInput = new BufferedReader(
  new InputStreamReader(p.getInputStream())
);
```

Listing 4.1: Executing a Python script in Java

26

I use a Process to execute the Python script at runtime, and then read the resulting output through a standard input stream.

Swing vs. JavaFX

When deciding the library for user interface design, there are questions that need to be addressed to make a fully informed decision:

- Which library is cleaner and easier to maintain?

- What will be faster to build from scratch?

- Is it required to maintain a system look-and-feel?

Because the front-end design of the application is written in Java, there are two possible UI libraries primarily used for desktop application design: JavaFX and Swing. Swing can be seen as a borderline-legacy library when compared to the up-and-coming JavaFX library. With this statement in mind, Swing is fully featured and supported heavily at the moment, whereas JavaFX still doesn't have important features implemented yet (such as using a default system look-and-feel). When considering the usability of these libraries, JavaFX is much more consistent across components. However, this is mainly dependent upon how the code is written and structured. Even though JavaFX is more consistent, Swing has more usable components (third-party and built-in) and not all of them have been ported into the newer JavaFX platform. With these factors in mind, I have chosen to use the Swing library due to time constraints, ease of development and a much larger component set available for interface design.

4.2 Development Environment

Before I could start developing Miraihilate, I first had to setup my development environment. This included getting the right programming languages installed, and selecting the modules/libraries and frameworks I will use.

4.2.1 Python Environment

Miraihilate's back-end will require Python version 3 and upwards and the most important modules for backend productivity include:

- Paramiko

  – Paramiko is a library which makes it easier and cleaner to create client or server SSH2 connections. Since Mirai uses SSH as one of its attack vectors to root into an IoT device, Miraihilate will take advantage of this module to make SSH connections to the vulnerable devices.

- Telnetlib

  – This module provides classes and utilities that implement the Telnet protocol. This will allow me to use Telnet as another access point to check the vulnerability of an IoT device.

- MySQL Connector

  – This is a MySQL driver which will allow me to connect to the local Miraihilate database and store relevant scan data.

- BCrypt

  – This module will be used in the setup utility to create the correct BCrypt hash for the initial Miraihilate user's password, which will work with the front-end client.

4.2.2 Java Environment

Miraihilate's front-end client interface will be written in Java, and will require at least Java 7 to run. This will be due to the potential for newer features being used in the code, such as lambdas and certain factory patterns. To maintain the development process for the client program, I used Maven to manage the project and allow me to easily add required libraries. The main libraries used for the client program include:

- JBCrypt

  – The JBCrypt Java library is an OpenBSD-style Blowfish password hashing library. This will be used for checking the password hashes in the database when a user wants to login to Miraihilate.

- MySQL Connector

28

– This is a JDBC driver which will be used to connect to the MySQL database for retrieving scan results to be displayed in the client program.

4.3   Setup Utility

Before a system administrator (and others within their organisation) can use Miraihilate, it will first have to be setup. The setup utility will use a command-line interface (CLI) and is written in Python. It will ensure that an initial user of an admin role is created, so that there is a user available upon the application being run for the first time.

Figure 4.1: Setup Program CLI

As shown in Figure 4.1, the setup program asks for the user's first name, last name and email address. The email address will be what they use to login to the system. The setup utility does not ask for a password, but instead generates a cryptographically-secure 16 character long password made up of letters, numbers and symbols.

Before sending the resulting user information to the database, it first generates a Universally Unique Identifier (UUID) and the BCrypt hash for the raw password.

```
pwd_hash = bcrypt.hashpw(raw_pwd.encode('ascii'),
```

```
bcrypt.gensalt(prefix=b"2a"))
```

Listing 4.2: Hashing a raw password using BCrypt

As seen in the code above, a salt is generated for the hash with the "*2a*" prefix. This is because the JBCrypt library in Java has not been updated to use the newer prefix, and therefore I have to ensure that the hashes can work both with the Python scripts and the front-end client in Java.

4.4   Backend Development

To purpose of splitting the back-end scripts from the front-end interface is to separate the functionality from the graphical interface, following a model-view pattern where the model is the back-end scripts which manages the logic and data and the view is the front-end interface. The backend was chosen to be written in Python, due to it having more support and separated libraries for SSH and Telnet compared to Java. For the back-end scripts, I separated the functionality using two scripts, one for the quick scanning utility and the other for advanced scanning.
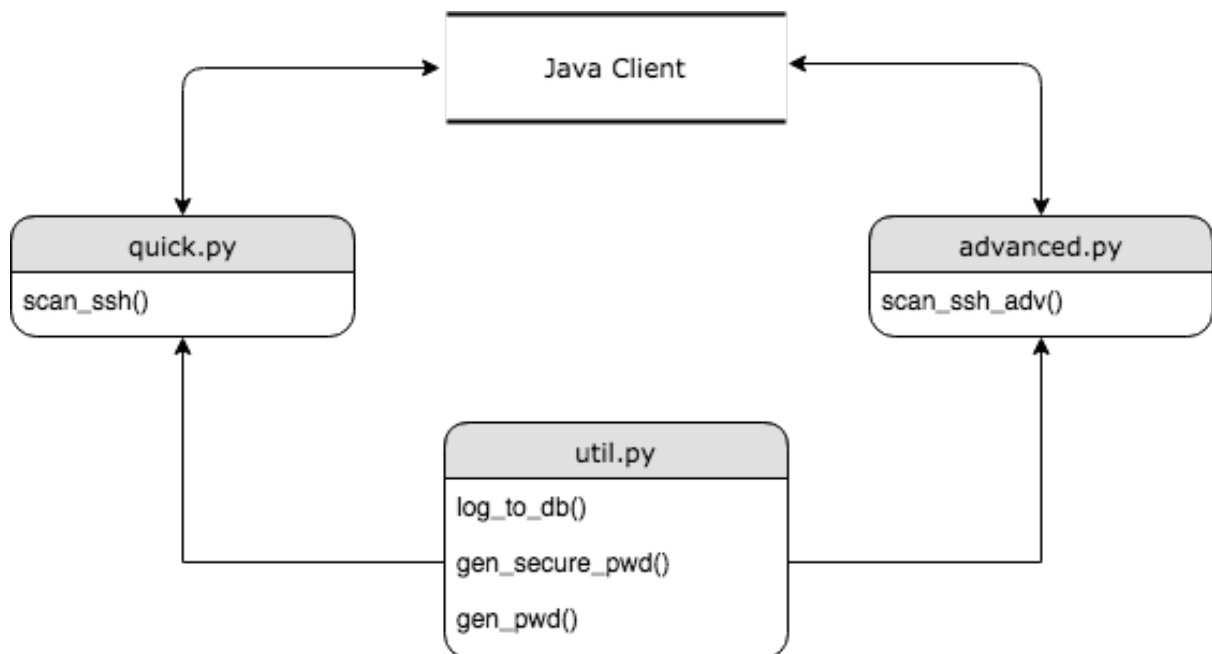
4.4.1   Backend Structure

Figure 4.2: Back-end Structure

As seen in the above image, the functionality is separated into quick and advanced scanning scripts. Initially the scripts both contained the utility functions as described by *util.py*, but were

30

refactored to increase the degree of modularity in the code.

There are two functions for password generation which are *gen_pwd()* and *gen_secure_pwd()*. The *gen_pwd()* function generates a **cryptographically secure** 16 character long password, made up of letters, numbers and symbols. It ensures a cryptographically secure password by using the *secrets* module to select a random character, generated by SystemRandom (which generates the number from sources provided by the operating system), from the alphabet shown in Listing 4.3.

```
alphabet = string.ascii_letters + string.digits +
    string.punctuation
```

Listing 4.3: Password Alphabet

The second function, *gen_secure_pwd()* is used to ensure that the password ultimately generated contains at least one uppercase letter in the password, following the Linux root password standard.

4.4.2 Logging

As shown in Figure 4.2, the utility script consists of a logging function, which sends the resulting scan log to the database attached with the UUID of the user who performed the scan.

The Java client uses a *JEditorPane* to display scan results, which supports HTML 3.2. This is an outdated version of HTML, but provides more than enough elements to support the requirements of displaying scan logs. Therefore during the scan, a list of HTML elements is collected along with relevant information for each part of the scan. This list is then joined together and output to the *log_to_db()* function to store the scan log. Appendix B.1 shows an example scan log for an SSH scan.

4.4.3 Scanning

Miraihilate's scanning capability is divided into quick scanning and advanced scanning. The aim of the quick scan utility is to enable a simple scan to run on a network range once the client has started, whilst the advanced scanning allows users to set the timeout for scanning and provide extra commands to the devices. Once a scan has completed, the data will be sent to the database and the client will update to display the latest scan results.

SSH Scanning

The SSH scanning function scans a specified network range and attempts to bruteforce into device root accounts found on the IP address range, using a default list of username/password combinations through an SSH connection. It loops through a list of ip addresses specified by an IP address and a CIDR suffix and then tries to establish an SSH connection to the device, trying each of the username/password combinations. If a connection is made it will look at the parameters sent to the scan, relative to whether it was a quick or advanced scan, and perform its respective function such as changing the password of an infected device.

<Telnet Scanning?>

Note: This has no effect on usability, but if there is time to ultimately implement Telnet functionality, I can fill this in.

4.5   Client Development

The Java front-end client is the main program that users will interact with to use the scanning utilities, which provides better visual representation of the utilities and logs, and increases usability compared to directly using the backend scripts.
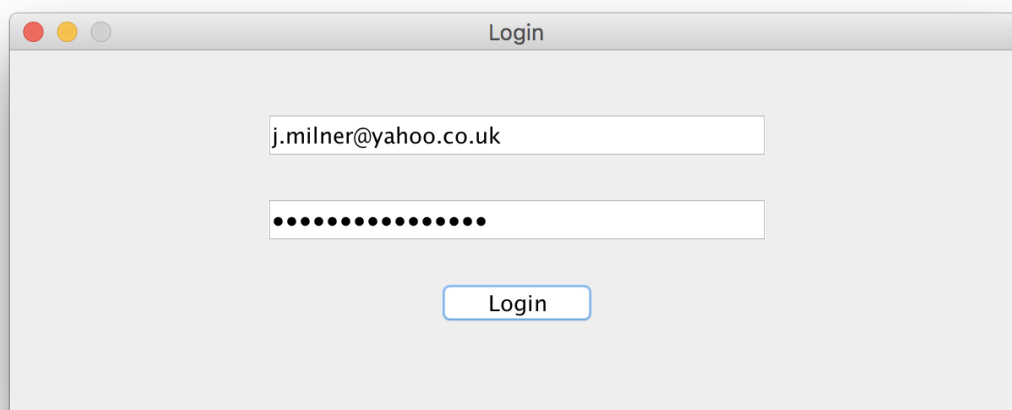
Figure 4.3: Login Window

As a user starts Miraihilate they will first be prompted to login to the program using the email and

password shown in the setup program or from the admin user creation utility. The raw password entered by the user is then hashed using BCrypt to validate the hash stored with the email address in the database. If the two hashes are equal, then the user gains access to the functionality of Miraihilate's client interface.
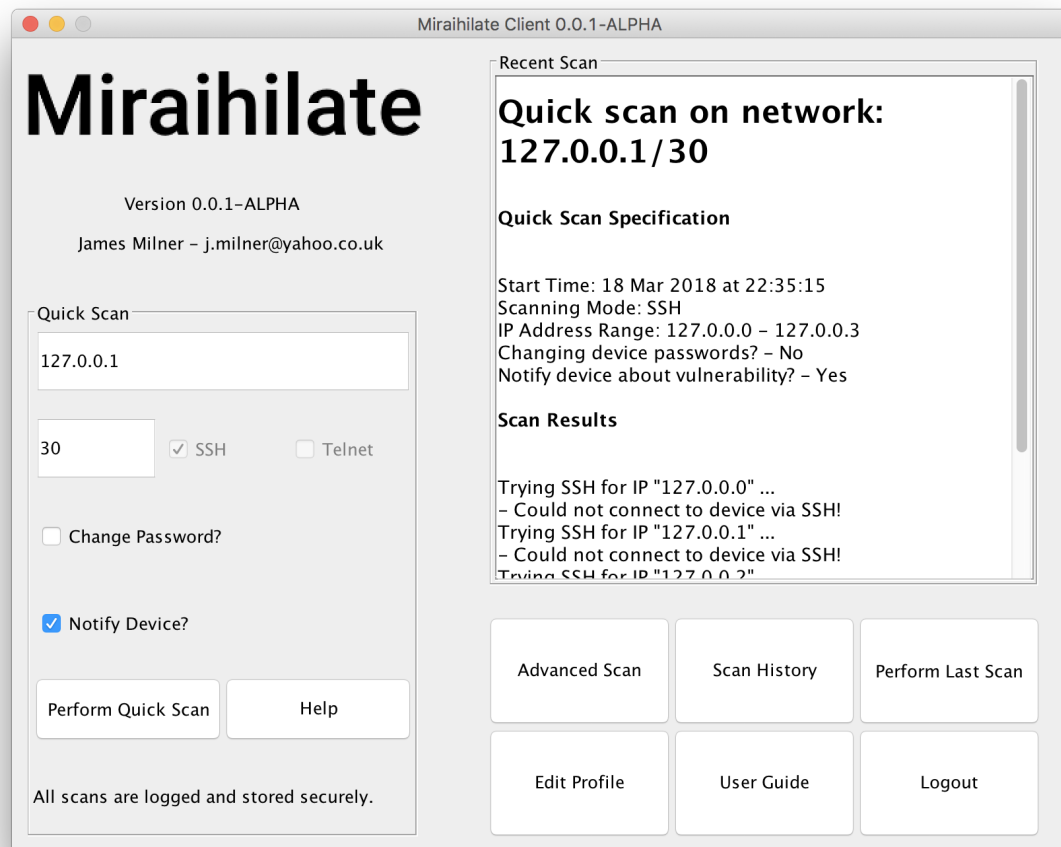
Figure 4.4: Client Window

After successfully gaining access to the client, the user will be shown the main interface as shown in Figure 4.4. As seen in the image, the user will first see the quick scanning utility, a view for the log of their most recent scan and buttons for accessing other options the client offers.

4.5.1 Quick Scanning

The quick scanning utility provies users with the ability to initiate a simple scan on a network range. It provides minimal options for communicating with vulnerable devices and its primary aim is to check for vulnerable devices within the network. If a user does not remember how to operate

the quick scanning utility they can either refer to the user guide, which is also accessible from the client interface, or they can click the help button from the quick scan panel. This will bring up a quick recap on what each of the fields for quick scanning mean and require, as shown in Figure 4.5.
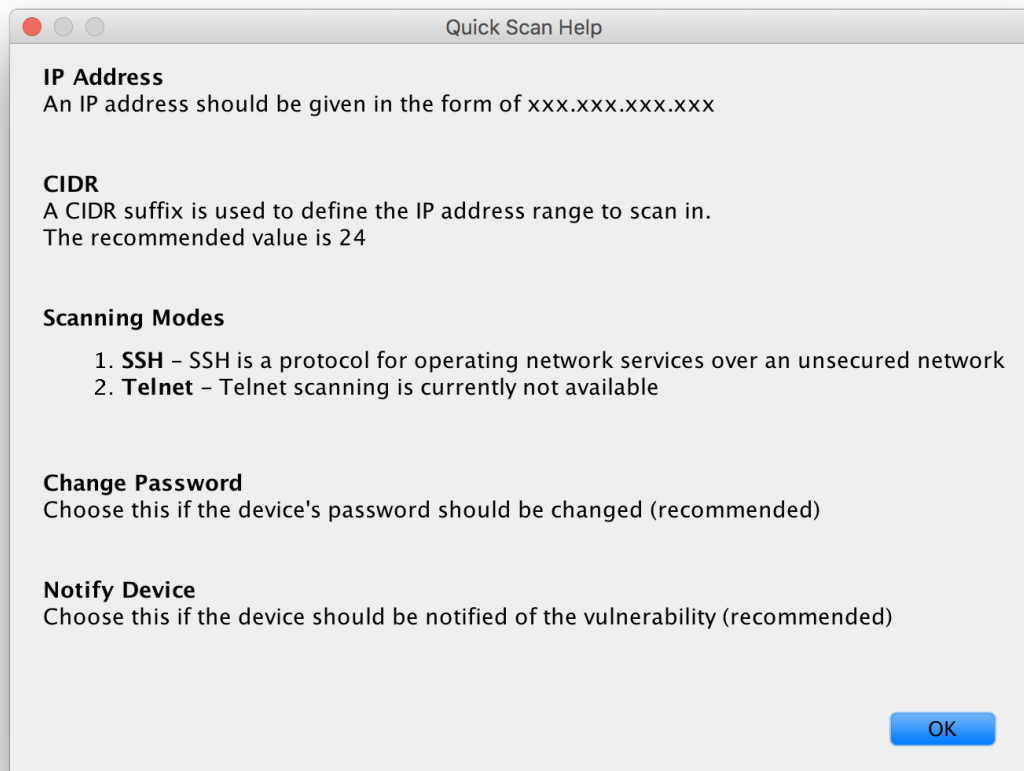
Figure 4.5: Quick Scan Help Window

4.5.2 Viewing Scan Logs

A user of Miraihilate will have two ways of viewing scan logs. The first is a fixed view of their most recent scan performed and the second being a list of all their scan logs to date. Originally, the scan log with stored as JSON data, but parsing JSON data into a text area on the client did not provide a good enough visual representaiton of the data. Therefore I decided to store the scan data as a HTML document and use a *JEditorPane* to display HTML scan data from the database, so no parsing has to be done.

As seen in Figure 4.6, whilst viewing their scan history, users can select a scan item from list on the left panel to display its corresponding scan result in the view on the right. The client connects to the database to retrieve a list of the users scans in descending order, starting from the latest scan they have done. To support non-repudiation of scans, they cannot be deleted and scans are linked with the time they were performed and the UUID of the user who did the scan.

(a) Recent Scan View
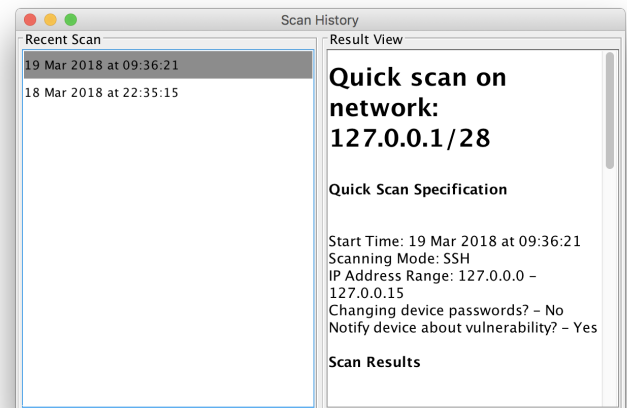
(b) Scan History

Figure 4.6: Two Methods of Viewing Scan Logs

4.5.3 Advanced Scanning
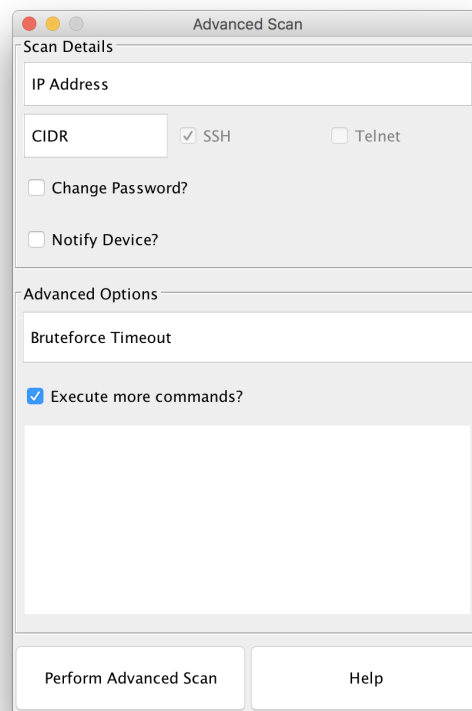
Figure 4.7: Advanced Scan Interface

As well as the quick scanning utility, the client interface also provides a window for advanced scanning. As shown in figure 4.7, the interface provides the same fields as a quick scan, with the added advanced options of choosing a timeout for the scans as well as the ability to provide extra commands to the device. These commands are not client-specific, but commands that can be executed via SSH to a unix-based operating system. Example commands include:

```
# Print system information
uname -a
# Download a shell file from a server and execute it
wget http://<hostname> -O- | sh
# Show who is logged on and what they are doing
w
```

5.   Usability Evaluation

Along with the feedback from the mockups before implementation, I performed a usability test on an initial working prototype of Miraihilate. The aim of this study was to gain final usability feedback before finalising the program prior to testing the final working version.

5.1   Recruitment

In order to conduct a reliable user evaluation I aimed to test at least 20 people, due to the study having both qualitative and quantitative data being collected. Because Miraihilate will initially consist of two user roles (operators and administrator), I will design questions to test the usability of both the roles and select participants with at least a moderate technical background

For selecting participants, I used global design company IDEO's method of recruiting "*Extreme*" and "*Mainstream*" participants. Extreme partipants allow me to include participants that cover skills included in my target user group, whilst having mainstream participants allows me to also include participants who have minimal computing experience. If the mainstream participants do not have much difficulty with the mockups, it suggests that the majority of participants will be able to perform well on the study as well. To identify the extreme and mainstream candidates, before conducting the study they answered a brief question sheet consisting of questions to analyse their technical capability, so that their results can be categorised into what I would consider extreme and mainstream users. The brief question sheet used for categorising participants can be found in Appendix C.2.

5.2   Conducting the Study

Before the study commenced, participants were asked to sign a consent form (shown in Appendix C.1) to the study, explaining what the purpose of the study was and what would happen. It also identified that they can opt-out of the study at any time and their results will be withdrawn. As described in the recruitment section, participants were first given a brief question sheet to analyse their technical capability. This had no effect on the study itself, but was used in the analysis of results to categorise the participants into extreme and mainstream users.

After this, participants were then given the program to use alongside four scenario questionnaires (found in appendices C.3-6). The scenarios given to participants are:

1. Setting up Miraihilate

2. Performing a quick scan and viewing recent scan history

3. Changing their password from the auto-generated one

4. Performing an advanced scan with extra commands from a given list

The first scenario involves a user running the setup utility from the commandline from a set of instructions, which downloads the program and sets up their own admin account to use for the next three scenarios. The second involves using another set of instructions to perform a quick scan on a local network setup prior to the study, and then viewing the latest scan result by opening up the scan history window. Thirdly, participants will be asked to edit their profile by changing their password. If they wish to change anything else they have an option to do so. Finally participants, having already experienced the quick scan, will perform an advanced scan. They will be given a list of commands that they must choose at least one from, to use in the extra commands option in the scan window. Upon completing the scenario questionnaires, participants were asked verbally how they felt the study went.

5.3   Results Analysis

References

[1] Classless inter-domain routing (cidr) and notation for beginners. `https://whatismyipaddress.com/cidr`. Accessed: 2017-12-20.

[2] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1:9–52, 2011.

[3] Christer Eriksen Hole. Analysis of security for iot: A review of architecture and cryptographic algorithms. Master's thesis, Universitetet i Agder; University of Agder, 2017.

[4] The hunt for iot: The rise of thingbots. `https://f5.com/labs/articles/threat-intelligence/ddos/the-hunt-for-iot-the-rise-of-thingbots`. Accessed: 2017-10-21.

[5] Gregory Fedynyshyn, Mooi Chuah, and Gang Tan. Detection and classification of different botnet c&c channels. *Autonomic and trusted computing*, pages 228–242, 2011.

[6] Botnet. `https://en.wikipedia.org/wiki/Botnet`. Accessed: 2017-10-04.

[7] Thomas Dübendorfer and Bernhard Plattner. Analysis of internet relay chat usage by ddos zombies. 2004.

[8] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 24–31. IEEE, 2008.

[9] Anoop Chowdary Atluri and Vinh Tran. Botnets threat analysis and detection. In *Information Security Practices*, pages 7–28. Springer, 2017.

[10] MARJZ Fabian and Monrose Andreas Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Understanding Botnets, Cambridge, USA*, 2007.

[11] Four ways cybercriminals profit from botnets. `https://www.symantec.com/connect/blogs/four-ways-cybercriminals-profit-botnets`. Accessed: 2017-10-23.

[12] Nathan Goodman. A survey of advances in botnet technologies. *arXiv preprint arXiv:1702.01132*, 2017.

[13] Prosenjit Sinha, Amine Boukhtouta, Victor Heber Belarde, and Mourad Debbabi. Insights from the analysis of the mariposa botnet. In *Risks and Security of Internet and Systems (CRiSIS), 2010 Fifth International Conference on*, pages 1–9. IEEE, 2010.

[14] Wenke Lee, Cliff Wang, and David Dagon. *Botnet detection: countering the largest security threat*. Springer Science & Business Media, 2007.

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *NDSS*, volume 8, pages 1–18, 2008.

[16] Lukas Braun and Hans-Joachim Hof. A system to save the internet from the malicious internet of things at home. In *submitted for âĂIJThe Eleventh International Conference on Emerging Security Information, Systems and Technologies, SECURWARE*, 2017.

[17] Microsoft helps fpi in gameover zeus botnet cleanup. `https://blogs.microsoft.com/blog/2014/06/02/microsoft-helps-fbi-in-gameover-zeus-botnet-cleanup/`. Accessed: 2017-10-26.

[18] Christos Douligeris and Aikaterini Mitrokotsa. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, 44(5):643–666, 2004.

[19] Ddos attacks. `https://securebox.comodo.com/ssl-sniffing/ddos-attacks/`. Accessed: 2017-10-26.

[20] Ddos attacks: the new cyber boogeyman. `https://www.reveelium.com/en/ddos-attacks-the-cyber-boogeyman-part-i/`. Accessed: 2017-10-27.

[21] The top 10 ddos attack trends. `https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf`. Accessed: 2017-10-27.

[22] Stephen Specht and Ruby Lee. Taxonomies of distributed denial of service networks, attacks, tools and countermeasures. *Princeton University Technical Report CE-L2003-03*, 2003.

[23] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. Ddos in the iot: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[24] Thingbots, the future of botnets in the internet of things.
`https://securityintelligence.com/`
`thingbots-the-future-of-botnets-in-the-internet-of-things/`. Accessed:
2017-11-02.

[25] Kishore Angrishi. Turning internet of things (iot) into internet of vulnerabilities (iov): Iot
botnets. *arXiv preprint arXiv:1702.03681*, 2017.

[26] Manos Antonakakis Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime
Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak
Kumar, et al. Understanding the mirai botnet.

[27] Breaking down mirai: An iot ddos botnet analysis. `https:`
`//www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html`.
Accessed: 2017-11-02.

[28] Mirai scanner by incapsula. `https://www.incapsula.com/mirai-scanner/`. Accessed:
2017-11-07.

[29] Openbsd cvs log for src/lib/libc/crypt/bcrypt.c.
`https://cvsweb.openbsd.org/cgi-bin/cvsweb/src/lib/libc/crypt/bcrypt.c`.
Accessed: 2018-02-15.

A.   Mockup Design Appendix

A.1   Mockup Design Consent Form

Mockup Feedback Consent Form

Participant No.

Project: Building and testing a white-hat botnet checker for evaluating the effectiveness of Mirai against other botnet malware

The objective of the experiment is to provide feedback on wireframes of the initial primary components of the botnet checker program. The experiment will involve completing **three** short questionnaires, which entail reviewing a wireframe image of the interface component and answering short questions about the design and placement of elements on the interfaces.

All results will be held in strict confidence, ensuring the privacy of all participants. There will be no record kept that would allow your results to be tracked back to you. All data will also be held securely in a locked office. A feedback sheet (containing summaries of the data mentioned above) will be sent to all participants who request it, after the data has been analysed. Your participation in this experiment will have no effect on your marks for any subject at this, or any other university.

You may withdraw from the experiment at any time without prejudice, and any data already recorded will be deleted.

Project Student:	Project Supervisor:
Ryan Shah	Mike Just
Email: rs10@hw.ac.uk	Email: m.just@hw.ac.uk

Name:

Signed:

Date:

Email Address:

I would like to receive information on the results of this study (circle **one**): **Yes / No**

A.2 Client Interface Feedback Questionnaire

Client Interface Feedback Questionnaire

Participant No.

The above image shows the initial window a user will see after logging in to the system. It contains: a logo, a panel for performing a quick scan, an area to view their most recent scan result and a section containing buttons leading to other aspects of the system.

1. On a scale from 1 to 10, how clear would you consider the above interface to be?

Unclear   Clear

1   2   3   4   5   6   7   8   9   10

2. Briefly describe how you would operate the quick scan section of the interface.

3. Are there any improvements (additions, changes, removals, etc.) you would make to the current design?

A.3 Advanced Scan Feedback Questionnaire

The above image shows the advanced scan window the program will produce upon pressing the *Advanced Scan* button on the client interface. It shows the same options as the regular quick scan, although with two more options for specifying a scanning timeout and any extra commands the user wants to send to the device.

1. On a scale from 1 to 10, how clear would you consider the above interface to be?

Unclear									Clear
1	2	3	4	5	6	7	8	9	10

2. Briefly describe how you would perform an advanced scan.

3. Are there any improvements (additions, changes, removals, etc.) you would make to the current design?

A.4   Scan History Feedback Questionnaire
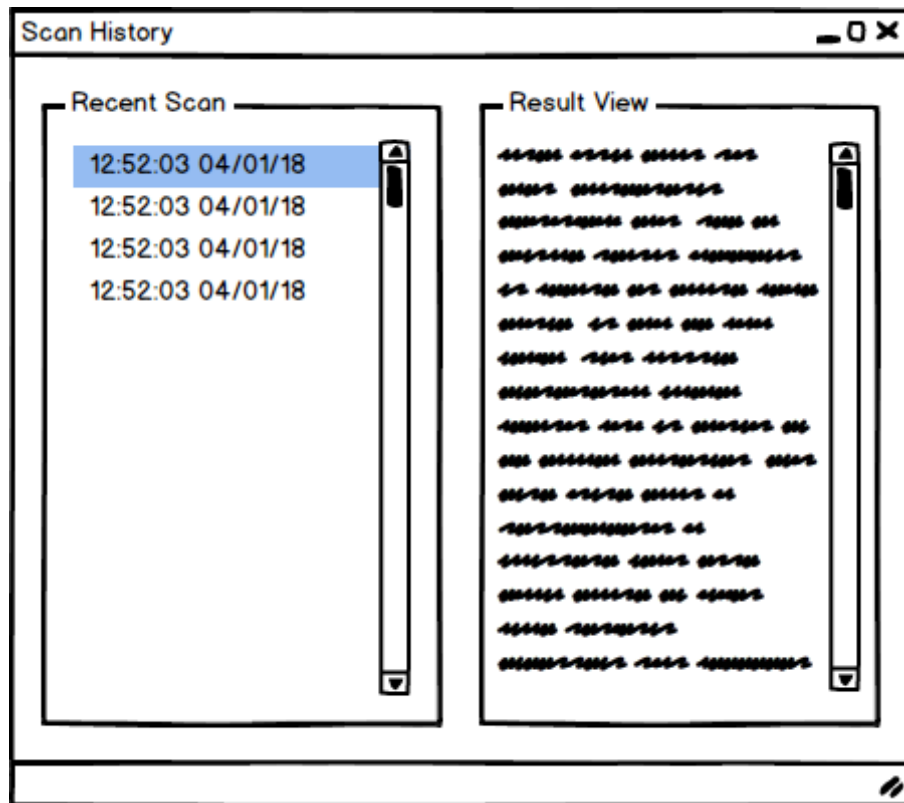
The above image shows the scan history window, which displays all the users scans based on the time the scan was performed. Selecting a scan item on the left panel brings up the scan results for that item.

1. On a scale from 1 to 10, how clear would you consider the above interface to be?

Unclear   Clear

  1   2   3   4   5   6   7   8   9   10

2. Are there any improvements (additions, changes, removals, etc.) you would make to the current design?

B. Implementation Appendix

B.1 Example SSH Scan Log

Quick scan on network: 127.0.0.1/30

Quick Scan Specification

Start Time: 16 Mar 2018 at 12:42:42 Scanning Mode: SSH IP Address Range: 127.0.0.0 - 127.0.0.3 Changing device passwords? - No Notify device about vulnerability? - Yes

Scan Results

Trying SSH for IP "127.0.0.0" ... - Could not connect to device via SSH! Trying SSH for IP "127.0.0.1" ... - This device is vulnerable! - The device was alerted about the vulnerability. - This device's password was not changed. - Closing device... Trying SSH for IP "127.0.0.2" ... - Could not connect to device via SSH! Trying SSH for IP "127.0.0.3" ... - Could not connect to device via SSH!

1 vulnerable device was found!

C.   Usability Evaluation Appendix

C.1   Usability Study Consent Form

<div align="center">

Usability Survey Consent Form

Participant No.

</div>

Project: Building and testing a white-hat botnet checker for evaluating the effectiveness of Mirai against other botnet malware

The objective of the experiment is to provide useful usability information of Miraihilate, a botnet checker program. The program is primarily used to scan a range of IP addresses to find devices that are vulnerable to Mirai malware. Prior to the study, you will be given a question sheet asking brief questions about your technical competence. In the study, you will be given the Miraihilate prototype program to conduct four scenarios given to you and answer questions about them.

All results will be held in strict confidence, ensuring the privacy of all participants. There will be no record kept that would allow your results to be tracked back to you. All data will also be held securely in a locked office. A feedback sheet (containing summaries of the data mentioned above) will be sent to all participants who request it, after the data has been analysed. Your participation in this experiment will have no effect on your marks for any subject at this, or any other university.

You may withdraw from the experiment at any time without prejudice, and any data already recorded will be deleted.

Project Student:	Project Supervisor:
Ryan Shah	Mike Just
Email: rs10@hw.ac.uk	Email: m.just@hw.ac.uk

Name:

Signed:

Date:

Email Address:

I would like to receive information on the results of this study (circle **one**): **Yes / No**

C.2 Pre-Study Question Sheet

Usability Survey Pre-Study Questionnaire

Participant No.

Below is a series of questions aimed to gather information about your technical competence, please answer all questions.

1. On a typical day, how much time do you spend on a computer? (Circle one)

  None   > 30 mins   30-60 mins   1-2 hours   2-4 hours   > 4 hours

2. Have you ever used a unix-based operating system, such as Linux or macOS?   Yes / No

3. If yes, on a scale of 1-10 how confident are you with using a unix-based operating system? (If no, proceed to question **4.**)

 Not Confident   Very Confident
   1   2   3   4   5   6   7   8   9   10

4. On a scale of 1-10 how confident are you with troubleshooting (fixing a problem) a computer program?

 Not Confident   Very Confident
   1   2   3   4   5   6   7   8   9   10

5. Have you had any experience with using a command-line interface, such as Command Prompt or Terminal?   Yes / No

6. If yes, how would you rate your experience with a command line interface? (If no, proceed to question **7.**)

Inexperienced   Experienced

    1   2   3   4   5   6   7   8   9   10

7. Have you had any experience with scanning programs, such as an antivirus?   Yes / No

8. Have you had any experience with network tools (i.e. network scanners, command-line network utilities)   Yes / No

C.3   Scenario One

In this scenario, you will be setting up Miraihilate. This program will ask you to enter details about yourself to setup your own user for the system, which will be used for the rest of the study. Please follow all the steps below.

1. The terminal window will already be open for you in the correct directory, so run the following command:

    ```
    python3 setup.py
    ```

2. Follow the remaining steps on the command-line interface.

3. After finishing entering your details, you will be given a securely generated password. Make a note of this, as this will be used for the rest of the scenarios.

   Note: You will see that the Miraihilate client program will be downloaded in the same directory.

After completing the above scenario, please answer all questions below.

1. Was it easy to setup Miraihilate using the command-line interface?

Very Difficult   Very Easy

   1   2   3   4   5   6   7   8   9   10

2. Would it be more preferable to have a graphical interface rather than using the command-line to setup the program?   Yes / No

3. Are there any other improvements or changes you would make to the current setup utility?

C.4   Scenario Two

In this scenario, you will be performing a quick scan and viewing your recent scan history. Please follow all the steps below.

1. Run Miraihilate by double clicking the executable file named *miraihilate.jar*

2. You will be asked to first login, so use the email address you entered and password generated for you from the first scenario.

3. Spend a brief amount of time looking at the features of the main interface you see.

4. Locate the quick scan panel and fill in the field with the following details:

   ```
   IP Address: 127.0.0.1
   CIDR: 30
   Change Password: Unticked
   Notify Device: Ticked
   ```

   Press the help button if you want more information on what these things mean.

5. Press the quick scan button.

6. Open the "Scan History" window.

7. Check that you can see your scan.

After completing the above scenario, please answer all questions below.

1. On a scale of 1-10 how easy was it to login to Miraihilate?

Very Difficult   Very Easy

    1   2   3   4   5   6   7   8   9   10

2. Are there any improvements or changes you would make to the login?

3. Looking at the main interface after logging in to Miraihilate, are there any improvements or changes you would make to improve the layout and usability of the main interface?

4. On a scale of 1-10 how easy was it to perform a quick scan?

Very Difficult   Very Easy

1   2   3   4   5   6   7   8   9   10

5. Are there any improvements or changes you would make to the quick scan panel to improve its usability?

6. How easy was it to view your scan history?

Very Difficult   Very Easy

1   2   3   4   5   6   7   8   9   10

7. Are there any improvements or changes you would make to the scan history interface?

8. Ultimately, how confident were you performing a quick scan on a network using Miraihilate?

Not Confident   Very Confident

    1   2   3   4   5   6   7   8   9   10

9. Do you have any final comments about the tasks or the interfaces used in this scenario?

C.5   Scenario Three

Participant No.

In this scenario, you will be editing your profile to change your password. You were given an auto generated password, which is cryptographically secure, but have the option to change it to one more suitable to you if required. Please follow all the steps below.

1. Open the "Edit Profile" window.

2. Enter a password of your own choice and press the Change Password button.

3. Logout of Miraihilate and start up the program again to login with your new password.

After completing the above scenario, please answer all questions below.

1. On a scale of 1-10 how easy was it to change your password?

Very Difficult   Very Easy

    1   2   3   4   5   6   7   8   9   10

2. Did you feel like the previously generated password was secure?   Yes / No

3. Would you prefer to have the option to use your own password in the setup stage?   Yes / No

4. Are there any improvements or changes you would make to improve the layout and usability of the main interface?

C.6   Scenario Four

In this scenario, you will be performing an advanced scan. The procedure is similar to performing a quick scan, with the added functionality of providing a timeout and extra commands to the device. Please follow all the steps below.

1. Open the "Advanced Scan" window.

2. Locate the quick scan panel and fill in the field with the following details:

   ```
   IP Address: 127.0.0.1
   CIDR: 30
   Change Password: Unticked
   Notify Device: Ticked
   ```

   Press the help button if you want more information on what these things mean.

3. Choose a timeout value under 8 seconds and enter one or more commands from the following list:

   ```
   # Create an alert with Hello World
   - notify-send 'Hello World'


   # Create a directory in the root user home
      directory called hello
   - mkdir hello


   # Show who is logged on and what they are doing
   - w


   # Create a directory called study and create a file
      called test.txt inside it, containing "Hello
      World"
   ```

```
- mkdir study; cd study; echo "Hello World" >
   test.txt


# List items in the root user home directory

- ls
```

4. Pressed the Perform Advanced Scan button.

5. Go back to the main interface and see your scan in the "Recent Scan" panel.

After completing the above scenario, please answer all questions below.

1. On a scale of 1-10 how easy was it to perform the advanced scan?

Very Difficult   Very Easy

   1   2   3   4   5   6   7   8   9   10

2. What timeout and commands did you enter in the advanced scan?

3. Are there any improvements or changes you would make to the quick scan panel to improve its usability?