IOT Honeynet Framework

Timothy McCann
Jason Swords
Raigridas Bartkus

Department of Informatics, School of Informatics and Engineering,
Technological University Dublin

Submitted to Technological University Dublin in partial fulfilment of the requirements for the degree of

B.Sc. (Hons) Digital Forensics and Cyber Security

Supervisor:
Mark Cummins

20/05/2019

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of Honours B.Sc. in Digital Forensics and Cyber Security in the Technological University Dublin – Blanchardstown Campus, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed:_________________________	Dated: ____/_____/_____
Signed:_________________________	Dated: ____/_____/_____
Signed:_________________________	Dated: ____/_____/_____

Contents

Declaration 2

List of Figures 7

Abstract 9

Project Overview 10

Chapter 1: Introduction 11

1.1 Problem Area 11

1.2 Research Objectives 12

Chapter 2: Literature Review 13

2.1 The Modern Cyber-Threat Landscape 13

2.1.1 Cyber Attacks 13

2.1.1.1 Attack Factors 13

2.1.1.2 Types of Attacks 14

2.1.1.3 Syntactic Attacks 16

2.1.1.4 Semantic Attacks 17

2.1.1.5 Attackers 17

2.1.2 The Internet of Things 19

2.1.2.1 IoT Device Security 19

2.1.2.2 The Future of IoT Devices 22

2.2 Honeypots 22

2.2.1 Types of Honeypots 23

2.2.2 Honeypot Interaction Levels 23

2.2.3 Honeynets 24

2.2.4 Risks and Legal Issues 26

2.3 Related Work 27

2.3.1 Modern Honeypot Network 27

2.3.2 Tpot 27

Chapter 3: Methodology and Design 29

3.1 Concept 29

3.2 Features 30

3.3 Problems 31

3.3.1 Emulating services 31

3.3.2 Port issues 32

3.3.3 SSH port 32

3.3.4 Visualisation of data 32
3.4 Addressing challenges 33
3.4.1 Emulating services 33
3.4.2 Port Issues 34
3.4.3 SSH Port 34
3.4.4 Visualisation of Data 35
3.5 Hosting Service 35
3.6 Understanding Amazon EC2 Features 36
3.6.1 Setting Up Amazon EC2 36
3.6.3 AMI's and Instances 37
3.6.4 Regions and Availability Zones 38
3.6.5 Amazon EC2 Key Pairs 38
3.6.6 Security Groups 39
3.7 VPC Flow Logs and CloudWatch 39
3.7.1 VPC Flow Logs 40
3.7.2 Amazon CloudWatch 41
3.7.3 AWS IAM Policies and Roles 42
3.8 Docker 42
3.8.1 Building a Docker container 43
3.9 Python Socketserver 45
3.9.1 TCP Server 45
3.9.2 UDP server 46
3.9.3 Threaded programming 46
3.10 Bash 47
3.11 NMAP 48
3.12 Web Framework Design 49
3.12.1 Flask vs Django Comparison 49
3.12.2 Web Server 50
3.12.3 Routing System 50
3.12.4 Key Objectives 50
3.12.5 Framework Structure 50
3.12.6 Technical Specifications 52
3.12.7 Non-Functional Requirements 52
Chapter 4: Implementation 53
4.1 Scanning the IOT device 53
4.2 Creating the Clone Script 54

4.3 Emulating services 58
4.3.1 High Interaction method 58
4.3.2 Bash Method 59
4.3.3 Python Method 60
4.4 Clone Deployment 62
4.4.1 Building and running the Docker Image 62
4.4.2 Transferring files 64
4.4.3 executing the scripts 66
4.5 Deploying A Honeypot to the Honeynet 66
4.5.1 Requirements and Setup 67
4.5.1.1 Setup IAM User and Environment 67
4.5.1.2 Creating a key pair 70
4.5.1.3 Create a Log Group 71
4.5.1.4 Create a Role and IAM Policy 71
4.5.1.5 Create a Flow Log 74
4.5.2 Creating a Honeypot 75
4.5.2.1 Managing EC2 Instances 81
4.5.2.2 View Logs 82
4.6 Setting up Flask 83
4.6.1 Templating and Bootstrap 84
4.6.2 Route Functions 85
4.6.3 WTF Forms and Form Handling 86
4.6.4 Data Modelling with SQL Alchemy 88
Chapter 5: Analysis 90
5.1 Framework Features 90
5.1.1 Cloning Device 90
5.1.2 Deploying and Managing Honeypots 91
5.1.3 Visualisation Data 91
Chapter 6: Conclusions and Results 92
6.1 Conclusions 92
6.1.1 IOT Honeynet Framework 92
6.1.2 The Potential of Containers and Cloud Computing 93
6.1.3 The Future of IOT Security 93
6.2 Future Work 94
6.2.1 Complete Platform Independence 94
6.2.2 Improve Honeypot Design 94

6.2.3 Visualisation of Data 95
6.3 Closing Remark 95
Bibliography 97

List of Figures

Figure 2.1 Honeynet Architecture [35]. 25

Figure 3.1 Sample Flow Log Record 40
Figure 3.2 Docker build process 43
Figure 3.3 Python image [62] 45
Figure 3.4 TCP Three Way Handshake [56] 46
Figure 3.5 User Datagram Protocol [55] 46
Figure 3.6 WireFrame of the Main Web Framework 51
Figure 3.7 WireFrame of the Web Framework's Login Function 51
Figure 3.8 WireFrame of the Web Framework's Settings Page 52
Figure 3.9 WireFrame of the Web Framework's Sidebar for all Web pages except the Home (no sidebar) and Settings (custom sidebar) page 52

Figure 4.1 Clone.sh Script 58
Figure 4.2 Netcat fails to return correct message. 59
Figure 4.3 simple Python3 TCP server. 60
Figure 4.4 Multithreaded Python TCP socket server. 61
Figure 4.5 Reading in a text file. 62
Figure 4.6 Basic login prompt. 62
Figure 4.7 Dockerfile creation function 63
Figure 4.8 Dockerfile completed 63
Figure 4.9 IAM Navigation Menu. 67
Figure 4.10 IAM Add User. 68
Figure 4.11 IAM Username and Access Type. 68
Figure 4.12 IAM user permissions. 69
Figure 4.13 Review IAM User Details. 69
Figure 4.14 IAM User Created Successfully. 69
Figure 4.15 Script creates keypair. 71
Figure 4.16 Create Log Group. 71
Figure 4.17 Checks if role name is occupied. 72
Figure 4.18 Trust Relationship. 72
Figure 4.19 Create a Role 72
Figure 4.20 Flow Log IAM Policy. 73
Figure 4.21 Creates a policy, attaches policy to Role, then attaches role to IAM User. 73
Figure 4.22 List IAM users. 74
Figure 4.23 List Roles. 74
Figure 4.24 List Log Groups. 74
Figure 4.25 List VPC's. 75
Figure 4.26 Example resource and client session objects. 76
Figure 4.27 Stores absolute path of Scripts directory. 76
Figure 4.28 Store clone name. 76
Figure 4.29 Function checks ports.txt and protocols.txt for port 22 specifications. 77
Figure 4.30 Sub Process module passing form data to clone bash script. 77
Figure 4.31 Create ingress rules for security group. 78
Figure 4.32 Creating security group and adding ingress rules. 79
Figure 4.33 Displaying Keypairs to user. 80
Figure 4.34 Create Instance. 81

Figure 4.35 Python Scripts that Manage Instance States. 81
Figure 4.36 Display Honeypots on Framework. 82
Figure 4.37 List Log Groups and Honeypot Private IP addresses. 82
Figure 4.38 Return Filtered Logs. 83
Figure 4.39 template.html using Bootstrap, expressions and control flow statements...................................... 85
Figure 4.40 Navbar displayed on the Web Framework... 85
Figure 4.41 routes.py app.route decorator. 85
Figure 4.42 run.py run method. 86
Figure 4.43 __init__.py using a SECRET_KEY and SQLALCHEMY_DATABASE_URI configuration variables. 86
Figure 4.44 forms.py class with validators. 87
Figure 4.45 routes.py function using GET and POST methods and using the validate_on_submit () method. ... 88
Figure 4.46 Registration Form displayed on the Web Framework. 88
Figure 4.47 routes.py using db.session.commit () method. 89
Figure 4.48 Users committed to the database. 89

Abstract

The exponential increase of interconnected devices that were traditionally isolated has motivated attackers to develop new threats targeted at vulnerable devices and critical service infrastructures. As a result, the internet is becoming an increasingly hostile environment with a constantly evolving threat landscape. These threats are continuing to grow in sophistication and creating a larger gap between the threats themselves and the current security solutions in use to mitigate them. This research focuses on providing enhanced security by developing a system that is used to deploy and manage a honeynet of IOT devices, which will aid in the promotion of honeypots as a means to mitigate security risks and provide valuable threat intelligence.

In particular, much of the research focuses on developing a system that can feasibly provide the ability to clone, deploy and manage honeypots in a holistic approach. Special efforts were made to ensure that the solution operated efficiently and could be maintained effectively. Leveraging the advantages of containers and modern hosting services to deploy security solutions such as honeypots is an area with huge potential and was explored within the scope of this research.

The novel element of this solution is the ability for the user to create a honeypot by cloning an existing device and emulating its services. While research and industry projects have explored the ways in which honeypots can be created and deployed, this research distinguishes itself by providing a functional web framework that has the unique feature of being able to clone and emulate existing services on devices and then deploy the clone as a low-level interactive honeypot that is attached to a honeynet. This IOT Honeynet Framework can be used in a production role to detect and defend against threats targeted at an organization or in a research role to help model the modern threat landscape and provide valuable research on modern security threats.

Project Overview

The project overview provides an overall look at the structure of the thesis and its contents. *Chapter 1* introduces the reader to the problem area of the thesis that is explored and the research objectives of the thesis which looks at what motivates the research that was undertaken and the desired result of the overall thesis.

Chapter 2 entitled as "*Literature Review*", provides a background to the area in which the research is focused on. The modern cyber threat landscape is explored with attention payed to the different cyber-attacks used and what role IoT plays in the current cyber landscape. The role of Honeypots is explored with special attention focused on the types of interaction levels honeypots have, what honeynets are and the legal issues that are encountered when working with honeypots. Finally, some closely related work that is relevant to this project is evaluated.

Chapter 3 entitled as "*Design and Methodology*", focuses on the design and methodology used to build the IOT Honeynet Framework. It focuses on briefly covering the overall concept envisioned for the IOT Honeynet Framework and the associated problems encountered during the initial design of the framework. The solutions to addressing these problems are explored in this section, which covers the technologies used to accomplish the various design goals for the framework.

Chapter 4 is entitled as "*Implementation*" and elaborates on the steps taken to implement the desired concept for the framework that was envisioned and designed in *Chapter 3*. It covers the overall implementation of the features in detail making use of code samples to reinforce the understanding of how the features work.

Chapter 5 is entitled as "*Analysis*" and analyses the final product developed after the implementation chapter. The chapter evaluates framework in terms of what it is capable of and all the features that were successfully implemented for the framework.

Chapter 6 is entitled as "Conclusions and Results" and is the final chapter in the thesis. This chapter provides a summary of the final product and the final results gained from conducting the thesis. There are some final closing remarks on some future work that could be used to continue the project and enhance the final solution.

Chapter 1: Introduction

This research aims to show the potential of an IOT Honeynet Framework that allows the seamless process of scanning, cloning and deploying a honeypot that can be used to help individuals and organisations against emerging cyber security threats in the modern cyber landscape. *Section 1.1* outlines the problem area that the research focusses on and tries to address these problems. This section lays a foundation to highlight the importance of the research to provide security solutions to combat new and emerging cyber security threats. *Section 1.2* discusses the objectives of the research which are motivated on addressing the problems stated earlier in *Section 1.1*. These objectives include the development of an IOT Honeynet Framework that can scan, clone and deploy devices on a network as a honeypot which is part of a honeynet.

1.1 Problem Area

Devices are being connected to the internet at an ever-increasing rate due to the reliance societies and organizations place on interconnected services, which is steadily growing. These devices are becoming increasingly operated by individuals who have little technical background and even less knowledge on security threats. This has created a fertile environment for hackers, who can attack, exploit and take advantage of such vulnerable devices to devastating effect.

In recent years, these attacks have become increasingly targeted against critical infrastructure such as communication services, transport systems and manufacturing facilities. These services are mostly controlled and monitored by using Industrial Control Systems (ICS) which incorporate off-the shelf software [1]. This can be exemplified in the maritime sector of Europe, which is critical for European society as there is an increase in the dependency for maritime transport. Like other important economic sectors, maritime increasingly relies on information and communication technology (ICT) systems to optimize and increase the efficiency of operations in the sector from the management of cargo to the traffic control communications [1]. With a greater amount of cyber-attacks being targeted against critical infrastructure and services, operators of these services are now facing growing uncertainty about the security of these systems.

It is also possible to see a growing change in the way that infrastructures are being deployed as organizations seek to reduce costs by outsourcing the management of such infrastructure to third parties. The decoupling of services by deploying them as microservices can be seen with service providers such as Amazon Web Services has allowed customers to benefit from greater scalability, agility and speed of service deployment. How to implement proper security solutions when performing such deployments is essential and must keep up as organizations continue to migrate to these types of deployments.

1.2 Research Objectives

The research conducted in this thesis is motivated by the growing reliance of interconnectivity between devices within society, the growing threats against infrastructure and critical services and the way in which the deployment of such critical services and infrastructure is changing. By observing these challenges, it is becoming glaringly obvious that new security solutions need to be investigated and designed, that can deter and protect against malicious attackers. Developing a solution that can be used to not only detect and defend against attacks but also help in identifying new threats and enabling analysis of such threats is the solution to these rising challenges.

This research investigates an approach to providing a practical and feasible solution to defend against emerging threats that are becoming increasingly more sophisticated and are targeted against devices and critical infrastructures and services. Establishing an open source IOT Honeynet Framework that can seamlessly clone and deploy devices on a network as part of a honeynet is ideal in achieving this objective. The novelty of this approach is the use of a script that scans a device on a network and produces a low-level interactive honeypot clone of the device, the deployment of these low-level honeypots on a honeynet and a framework that provides a holistic approach to performing the two functions just mentioned.

The level of cyber-attacks against exposed devices on the internet continues to grow and evolve, fuelled by society's and businesses reliance on interconnected devices, the increased threats to critical service infrastructure and the way infrastructure is being deployed. The proposal of a novel security solution to effectively and feasibly address these challenges is the aim of this research.

Chapter 2: Literature Review

2.1 The Modern Cyber-Threat Landscape

The modern cyber landscape is not a static or fixed plane, but a constantly shifting environment that continues to change and adapt. Cyber incidents and threats are starting to adapt in response to the changes in how organizations and wider society use critical service infrastructures and devices on the internet. The growth in sophisticated attacks has prompted the development of new security mechanisms to defend against these threats, which continue to evolve.

2.1.1 Cyber Attacks

The occurrence of cyber-attacks continues to grow and the means to perform these attacks is becoming more freely available to everyone through the Internet. The level of knowledge and skill required to perform cyber-attacks has reduced due to automation and a new type of attacker group has been formed due to this. Cyber-attacks are often successful only because known and preventable vulnerabilities haven't been addressed. The success of cyber-attacks are also guided by victims lack of awareness and failure to secure their own device.

2.1.1.1 Attack Factors

There are three main factors that contribute to why cyber-attacks are launched against organisations or individuals. The three factors are fear, spectacularity and vulnerability.

Fear is a big factor in cyber security. It is reported that there is a cyber-attack every 39 seconds, which leaves organizations thinking that they are next. The frequency of cyber-attacks is on the rise due to artificial intelligence (AI) and automation which further adds doubts to a company's belief in being able to defend against and mitigate the risk of cyber-attacks. The high cost of recovering from cyber-attacks and the frequency of which such attacks occur leaves companies sceptical of being able to prevent attacks by implementing proper security mechanisms and hiring professional cyber security specialists or outsourcing cyber security to specialists' organizations, resulting in nearly 60% of company's not implementing proper cyber security solutions simply because they are outside of their budget [2].

Small-to-mid-sized companies will struggle the most in particular when it comes to hiring the right cyber security teams. A lot of small or mid-sized companies are not going to have a cyber division established within their company depending on their business model and might not necessarily feel confident in trusting third-party security organizations with their confidential data [2]. Following a cyber-attack, organisations and companies are reluctant to reveal any information about the occurred cyber-attack or even mention that a cyber-attack existed in the first place due to the fear of being publicly denounced and shamed as well as losing customer trust.

The spectacularity factor is the measure of the damage caused by a cyber-attack. Damage caused to a company or organisation can be measured through direct losses such as the loss of availability, loss of income and the loss of public trust and favour which was mentioned when discussing the fear factor. The cost of recovering from cyber-attacks can vary from company size and how valuable the company's assets are. A small-to-mid-sized company (SMB) will on average experience a cost of $50,000 from cyber-attacks, where a larger company can suffer costs up to $500,000 or more depending on the severity of the cyber-attack. This raises a question of how much investment should be made on cyber security defence if the damage caused from a single cyber-attack can cost a company so much. This again would vary from company's scale and size, but large multinational global company's like Accenture spend a massive $11.7 million annually on cyber security defence [2]. Accenture approach and investment on its organization's security solutions would not be feasible for small-to-mid size organizations even though their confidential data could be just as devastating if it's stolen or destroyed.

The vulnerability factor deals in measuring how vulnerable an organisation or individual is to a cyber-attack. Host and network devices may be out dated and running older versions of operating systems or updates which leave these devices susceptible to cyber-attacks. Therefore, maintenance of such devices should be highly prioritized and not overlooked. Previously, networks were shaped differently and were more closed off to the public making them far more defensible. Nowadays most networks are exposed to the public and encourage things like bring-your-own-device (BYOD), making the network harder to secure. Unfortunately, human error can also be seen as a vulnerability when measuring organisations security. The use of deceptive tactics like email phishing attacks against individuals to exploit the entity of trust and gain unauthorized access to individuals of organisations are not uncommon and the concept of doing so is known as social engineering. An organisations cyber security defence is as strong as their weakest point which can be exploited and bypass the whole security mechanism put in place [2] [3].

2.1.1.2 Types of Attacks

Cyber-attacks can be broken down into two categories, active and passive. Active and passive attacks can target a whole network or an individual host, depending on the target different attacks can be deployed in each scenario. An attacker can use a combination of both attack techniques to gain access to a system, network or data. Often a passive attack is launched prior to an active attack to perform a reconnaissance of the target and gather information that can reveal the vulnerabilities and weaknesses of the target.

An active attack is an attack that intercepts and modifies the information gathered during an attack phase. These types of attacks are often aggressive, and victims are usually aware when this attack is taking place. Active attacks are highly malicious in nature, often locking out users, destroying memory or files, or forcefully gaining access to a target system or network [4]. Syntactic attacks are examples of an active attack and are discussed further in 2.1.1.3. Other examples of an active attack include man-in-the-middle attacks, buffer overflows and Denial of Service attacks.

- A man-in-the-middle (MITM) attack, also known as a Janus attack is an "active form of eavesdropping in which the attacker makes independent connections with victims and relays messages between them making them believe that they are in contact privately" [5].
- Buffer overflow is a well-known attack dating back as far as 1988 when it was accidentally discovered by a graduate student. A buffer overflow attacks work by overrunning a buffers boundary and overwriting the adjacent memory locations causing the system to crash or perform in an unpredictable way. "Overflow attacks exploit a lack of bounds checking on the size of input being stored in a buffer array" [6].
- This attack focuses on crashing a system or making a system unusable or unavailable to legitimate users [5]. The attack exploits weaknesses in TCP/IP (Transmission Control Protocol/Internet Protocol) protocols and can be launched with minimum effort and can be very difficult to trace back to the attacker. Denial of Service attack can also be used to corrupt or in some cases delete data [7].

"An attack in which an unauthorized attacker eavesdrops on the communication between two parties in order to steal information stored in a system by wiretapping or similar means" [5]. The eavesdropper however does not make any changes to the data gathered and it is this feature that separates a passive attack from an active attack. Passive attacks are often viewed as non-disruptive methods of gathering information about a victim or a company who most of the time are unaware that the passive attack is even taking place. The goal of a passive attack is to collect data while remaining anonymous and silent [4]. Examples of a passive attack include port scanning using tools like Nmap and key logging by installing some sort of malware on the victim's system.

- Wiretapping or passive wiretapping refers to the monitoring or recording of data as its being transmitted over a communication medium, without altering or changing that data [8].
- Is a type of a Reconnaissance attack in which an attacker probes a network or a host to learn which ports are available and the services associated with the network or the host [5]. Ports found can be both closed or open and the goal of a port scan is to find an open port that is vulnerable to an exploit. A common tool used to perform port scanning is Nmap or a GUI version of this tool known as ZeNmap.
- Key logging represents a serious threat to the privacy and security of today's systems. A key logger is a malicious program that runs stealthily in the background on a user's computer and collects the sensitive information about that user, such as the user passwords, credit card details and any other personal information. Many anti-virus software's fail to detect a key logger running on a user's system and a user has no way to determine if their input on the keyboard is being recorded, often resulting in the user becoming a victim of identity theft and fraud [9].

2.1.1.3 Syntactic Attacks

"Syntactic attacks use virus-type software to disrupt or damage a computer system or network" [10]. Syntactic attacks are commonly referred to as "malware" because they are considered as malicious software. These attacks may include viruses, worms and Trojan horses.

Viruses are malicious executable files that can copy themselves and infect a computer through a removable medium like a USB key or through email files. However, viruses need user intervention to execute. A virus can hide in locations least accessed by a user, in the memory of a computer system to establish persistence and can attach itself to whatever file it wants to insert its own code into that program [11].

Worms are self-sustaining and replicating programs that can independently exploit vulnerabilities in a network and requires no user intervention to execute. A worm's main goal is to replicate itself across the whole system and establish persistence and a worm often uses network vulnerabilities to achieve this. All worms are harmful and have the ability to corrupt, delete or encrypt files creating the possibility for further attacks such as ransomware, if a worm is not configured with a payload it can still cause damage by consuming the bandwidth of a network. An example of this is the Morris worm [11].

Trojan Horses are malicious programs that appear legitimate and mislead the user into thinking that it will perform a desirable function but in fact facilitate unauthorised access to the user's computer.

Trojan horses often aim at exploiting authorization and granting the attacker access to the victim's computer where the attacker can then install additional malware on the victim's system. Trojan horses are unlike viruses and worms as they do not try to insert themselves into files, and don't focus on propagation. Nevertheless, a Trojan horse has some similarities to the previously discussed malware, for example it disguises itself as a useful program in hopes that it will be executed by a user and therefore requires user intervention which makes it similar to viruses [11].

2.1.1.4 Semantic Attacks

"Semantic attacks involve the modifications of information or dissemination of incorrect information" [10]. Modification of information has been around long before the invention of computers, but computers and networks have open up new opportunities to achieve this. The use of incorrect information can be used to harm, mislead or manipulate the individual target and damage their credibility. Semantic attacks are often used to perform cybercriminal activities such as propaganda and stock manipulation using a semantic hacking technique known as pump 'n' dump [10].

2.1.1.5 Attackers

Cyber attackers are individuals or groups who aim to exploit network vulnerabilities in IT systems and infrastructures. Cyber criminals are not limited to targeting organisations or businesses and the world of cyber security is utterly limitless. Cyber attackers often target individuals to steal personal data and can often use the information gathered from an individual as means of exploiting an organization or business [12].

In today's world, where information is easily accessible online to anyone, it's easy for individuals to gather knowledge and resources about IT vulnerabilities and exploits. Various open source tools are constantly being developed by the cyber security communities or individuals which are free and accessible to anyone and often require very little theoretical knowledge about hacking to operate. Attackers can be categorized into different groups based on technical background, motivation, the type of threats that they pose and many more. Below are some categories of attackers that are discussed.

Sophisticated attackers are cybercriminals that have great knowledge of cyber security and have both the practical and theoretical skills. Defending against these types of attackers can prove very difficult as their attacks are highly sophisticated and coordinated. These hackers are mostly motivated by the idea of financial gain [13].

Script-Kiddies are seen as the lowest tier of hacker and are often disregarded on an international cyber level. These hackers write scripts and programs to perform mischievous activities without fully understanding the exploits and vulnerabilities of a system. Script-kiddies are not focused on financial gain instead they are concerned about making themselves known to others and gaining a sort of reputation or fame [14].

Hacktivist is a person who uses hacking to influence political or social change. The term hacktivist originates as far back as 1994 from a hacker group known as the "Cult of the Dead Cow". Hacktivists also steal data and money to fund their agenda, but unlike a typical hacker who steals from anyone, hacktivists have an ideology to behave like Robin Hood who steals from the rich to help the poor. Hacktivists can be seen as vigilantes who used their cyber knowledge to enact social justice [15].

Bots also known as zombies are nothing more than compromised computers controlled by a human operator remotely. Bots can belong to a botnet which can consist of multiple bots. These botnets can be used to execute malware and launch various cybercrimes such as DDoS attacks, phishing and click fraud. Bots can additionally be used to infect other computers and turn them into bots [16].

Insiders are the individuals within an organisation that bypass the organisations security as trusted individuals but have the potential to cause a great cyber threat to the organisations network [17]. A report from the Computer Security Institute (CSI/FBI) stated that nearly 66% of all cyber breach incidents were in fact caused by authorized insiders [18]. There are three different attacks that and insider can perform on an organisation which are, misuse of access, defence bypass and access control failure [17]. All of these attacks are discussed respectively below.

- The misuse of an individual's access and privilege within a company's network is among the most difficult form of attack to detect and prevent. In this form of attack the insider uses their privileges and access rights to the company's resources in a wrongful manner. As mentioned previously it's very difficult to prevent or detect an individual's misuse of the company's resources, especially by examining the individual's behaviour through technical means. Technical analysis of data can be performed by checking for unusual patterns, quantities of requests or by inspecting the users log information for any suspicious activity [17].
- Insiders naturally have a far superior advantage over outsiders, as an insider has inside knowledge of an organisation's infrastructure, physical security and other defence layers. Insiders are essentially inside of a firewall and therefore are not blocked by it to some extent, and usually have some sort of a login access for a system which allows them to perform local based attacks rather than targeting the organisations network. Much like the previous attack discussed it is also difficult to prevent by depending on technical analysis alone, but counter

measurements can be put in place to examine anomalous behaviour and detection mechanisms can be configured to recognize known attacks on nominally-protected systems [17].

- Compared to the two above attacks, access control failure represents a technical problem within the organisations system. This can be a case of system or access control mechanism misconfiguration which if overlooked and can grant insider access to parts of the systems that they shouldn't be able to access or lead to more severe security threats [17].

2.1.2 The Internet of Things

Internet of Things (IoT) is an idea of forming an intellectual network with numerous devices, systems and equipment's. These devices are referred to as IoT devices and can vary from smart homes, smart cars, smart televisions, and smart light bulbs. All of these IoT devices use the word "smart". A smart IoT device requires minimal human intervention and has the intelligence to operate adaptively in different situations. IoT devices also have a key feature of automation, allowing them to gather and process data, collaborate with other IoT devices on the same network and make decisions based off this data. Apart from being intelligent and automated these devices are also very portable and support something called plug-and-play feature which minimizes the configuration and setup processes of these devices [19].

2.1.2.1 IoT Device Security

Internet of Things (IoT) achieves greatness by intercontinentally connecting devices across the globe from smart phones to wearable devices to home appliances. IoT devices are susceptible to major security threats and vulnerabilities that cannot be neglected. In this section IoT device security will be examined under the following headings, IoT infrastructure, authentication, authorization cryptosystems, privacy, software vulnerabilities and malware.

1. **Object Identification and locating in IoT**
   Identifying an IoT device uniquely is a challenge of its own and can be a problem before the actual security issues arise. To achieve this a proper object identification mechanism needs to be put in place that can identify each IoT device uniquely and reflect its properties. Domain Name System (DNS) is a good identification method that when combined with Fully Qualified Domain Name (FQDN) naming policy provides a unique identifier and reflects devices

properties. Object Name Service (ONS) was developed which combined object identification and device property location and is a structure that could now be applicable to IoT devices. Object identification on a network is also important since all IoT devices are connected to a network and the means to locate them geographically based on a device IP address is a problem that needs to be addressed. IoT devices can be located by using IPv4 and IPv6 locating methods, but the future of IoT device IP addressing remains to be seen as future candidates such as Named Data Networking (NDN) can replace the current IoT infrastructure [20].

2. **Authentication and Authorization**

   Authentication and authorization are vital components of security. Authentication refers to identifying a user while authorization decides what that permissions a user has on system. Authentication can be achieved by traditional methods such as using a passwords, pre-shared keys or public and private keys. However due to the complexity of IoT device networks, the standard authentication and authorization methods may not suffice or may not be applicable. Using cryptographical pre-share keys for IoT device authentication and authorization is an example why standard authentication and authorization methods are not applicable as the management of keys on a constantly growing network of IoT devices could prove rather difficult and ineffective [20].

3. **Privacy in IoT**

   Data privacy has become an ever-growing concern in today's world, as data leaks with sensitive user data have experienced growth in the recent years. Its no secret that web browsers and search engines collect user information to enhance the overall experience of that user but as for IoT devices the collection of information exceeds that of only collecting search history. IoT devices are capable of collecting information about user's daily routine, their interests and essentially have the ability to construct a profile for that user. Collection of this information may help by providing the user with relevant content and news, but this is not to be confused with the storage of this data and who has access to it. If the stored user's data is not securely protected, then this can lead to personal information mis usage [20].

4. **Cryptosystems and Security Protocols**

   IoT devices can vary from mobile phones, to wearable devices and smart sensors. IoT devices are known to be lightweight and versatile and this means that some of these devices will have certain constrains in terms of processing power and battery life and this can be a security issue. While there are suitable cryptosystems and security protocols to protect against these security threats, they are sometimes resource intensive. Therefore, some of these security features wouldn't be implementable on resource-constrained devices and would leave them vulnerable [20].

5. Software Vulnerabilities and Backdoor Analysis in IoT

Software vulnerabilities are generated from bugs during the development stage of a software. When a bug is discovered in a software and is exploited on the first day of its release, it is known as a zero-day. This exploit caused by this vulnerability can be severe and unknown. Software vulnerabilities are caused by security unawareness of software programmers which can be avoided by proper training. This is however easier said than done, as IoT devices have diversified hardware platforms, complex software and customized operating systems making it difficult to raise the security awareness and the aspect of secure programming [20].

Program analysis can be performed to discover software vulnerabilities before the software is launched into public. Program analysis can be broken down into static and dynamic analysis, static analysis focuses on examining a program that has not been executed yet, while dynamic analysis is the opposite and examines the state of a software that is running in a controlled environment. Dynamic analysis uses many advanced analysis techniques and tools that are highly intensive in terms of computational power and due to IoT device resource-constraints and limitations this analysis is often not possible to perform [20].

Backdoors are caused by software vulnerabilities that can be used for malicious intents by attackers such as having access and control over the victim's device. This malicious activity can be detected by the likes of an Intrusion Detection System (IDS) or prevented by an antivirus, however these security mechanisms require a certain amount of computational power that most IoT devices lack. Other backdoors are built into the software by the developers for testing or other purposes, these backdoors can be discovered by the use of reverse engineering and hijacked by attackers. Patches and updates applied to the software would cause the attacker to perform reverse engineering on that software again until a backdoor is discovered and this is often time consuming and hard to achieve [20].

6. Malware in IoT

IoT devices don't have immunity to malware too with the first IoT malware being discovered in 2013 by Symantec. This confirms that IoT devices are susceptible to IoT malware which is another security issue associated with IoT devices. IoT devices are known to exist in a network of other devices and therefore any IoT device infected by malware is highly contangoes to other devices on that network. This opens up a huge security risk as most devices are developed to have resilience to attacks from outside the network and not from within the network, end-devices such as surveillance cameras are a perfect example of where this would be an issue as they rarely contain strong security defences and can be used for profiling a victim's residence further leading to theft or blackmail. This is a serious violation of privacy and threat to IoT devices that awaits to be addressed [20].

2.1.2.2 The Future of IoT Devices

The future of IoT devices is versatile and holds great potential for personal, commercial and economic benefits. However, IoT devices also have a lot of challenges and possibly plenty more in the future. The technology is advancing at a rapid pace and developers are in constant gaze for new ideas and inventions. This puts a strain on the security side of IoT devices leaving them vulnerable and exposed. IoT devices face many privacy and security challenges such as data confidentiality, network security, encryption and information privacy. Many IoT devices use sensors to measure and transport data to the information processing unit, therefore it's important that these devices have proper encryption mechanisms implemented into them to ensure data integrity as well as proper authentication methods to ensure that the data is only accessible to the privileged users. The transmission system should be equipped well enough to handle large quantities of incoming data from sensors and ensure there is no data loss in the process. As well as considering the above points, it is important to protect the actual users of the IoT devices and prevent any privacy and confidentiality leaks [21].

2.2 Honeypots

Honeynet Project founder Lance Spitzner defines honeypots as a "security resource whose value lies in being probed, attacked, or compromised" [13]. Explained simply, a honeypot is a resource such as a smart bulb or router, that when setup, is expected to be probed, attacked and exploited. The value of the resource lies in the goal of it being noticed and attacked. If the honeypot is not probed or attacked, then it has very little to no value since it does not achieve its goal [13].

A honeypot functions by monitoring and logging the activities of any entity that interacts with the honeypot. Since a honeypot is a resource that has no impact on other services and provides no services, no entity should be theoretically communicating with them. This means any interaction with the honeypot is considered suspicious [14].

2.2.1 Types of Honeypots

Honeypots can be generally broken into two categories which include research honeypots and production honeypots.

A research honeypot aims to gather information on the hacking community. A research honeypot adds no value to an organization and is instead concerned with collecting data to help identify and investigate current threats, discover what kind of tools are being used, where these tools originated from and observing what systems are being targeted. A research honeypot must offer more functionality for a potential attacker to interact with and therefore is more demanding in resources and administration. The primary objective is to use the data collected to identify the kind of threats that are facing an organization and help in better protecting against these threats [13].

A production honeypot seeks to mitigate risk in an organization by trying to detect and defend against threats. A production honeypot adds value to an organization by mitigating the risk posed by attackers and is focused on preventing attacks from occurring rather than collecting detailed information on the attack itself. A production honeypot is not interested about what kind of tools that are being used or learning more about what an attacker does after gaining access to the honeypot and therefore tends to collect less information then a research honeypot. This also means that there is less functionality in production honeypots compared to research honeypots, since research honeypots are more interested in detailed information and therefore require more functionality for the attacker to interact with [13].

The distinction between these two honeypots is based on how the information collected by the honeypot is used rather than its implementation. The information collected can be used by an organization to detect and defend against attacks, which would make the resource a production honeypot. An organization could also use the information collected to better understand the tools used by attackers and the activities conducted in the aftermath of the attack, which would make the resource a research honeypot.

2.2.2 Honeypot Interaction Levels

Apart from the categories production and research, a honeypot can be categorized based on the level of interaction allowed between an attacker and the honeypot system. These interaction levels are broken down into low-level, medium-level and high-level. The level of interaction a honeypot system offers is indicative towards the desired function of the honeypot.

A honeypot with a low-level of interaction is designed to emulate a service and provide a façade for the attacker to work against. It poses the least amount of risk since there is little in the way of interaction between an attacker and the honeypot, but it also provides very little information on the attack conducted by the intruder due to having no functionality [15]. These honeypots are easy to deploy and maintain due to their lack of functionality and would fit the description of a production honeypot.

Honeypots with medium-level interaction are step above low-level honeypots by providing greater interaction between an attacker and the honeypot. There is a higher technical level behind the services emulated in medium-level honeypots which helps provide more information on the interaction between the attacker and the honeypot, but this also means the honeypot is harder to deploy and maintain due to greater functionality [15]. These types of honeypots would not provide enough functionality to be considered a research honeypot but would also be more technical than necessary to be a production honeypot. This again comes down to how the information is used rather than how the honeypot is implemented.

Finally, a honeypot with a high-level of interaction provides a large amount of functionality and is running its own services, operating system and even firmware. The level of functionality means that this honeypot is the most difficult to build and maintain. A honeypot with a high-level of interaction also provides the greatest amount of security risk due to more resources being available to the attacker, potentially causing the honeypot to become a security hole [15]. However, these honeypots can provide a wealth of information on the attacker and their activities after gaining access to the honeypot, which makes these highly interactive honeypots fit the category of research honeypot.

2.2.3 Honeynets

According to Lance Spitzner in his book "*Honeypots: Tracking Hackers*", a honeynet is the concept of building an entire network of production level resource honeypots that are then placed behind an access control device such as firewall. An attacker can then probe, attack and exploit any system in the honeynet [13]. The primary object of a honeynet is based on research. Honeynets can be used in the same role as production honeypots but due to their high functionality, the maintenance and administration cost is quite high. A honeynet offers a level of depth in information on attackers that provides a tremendous amount of value to researchers who wish to learn more about what kind of attacks are being used, what tools are employed, and what motives are behind an attack, to name just a few [13]. The key difference between a honeynet and a honeypot is that unlike the latter that only uses a single system, a honeynet is a physical network with multiple systems.

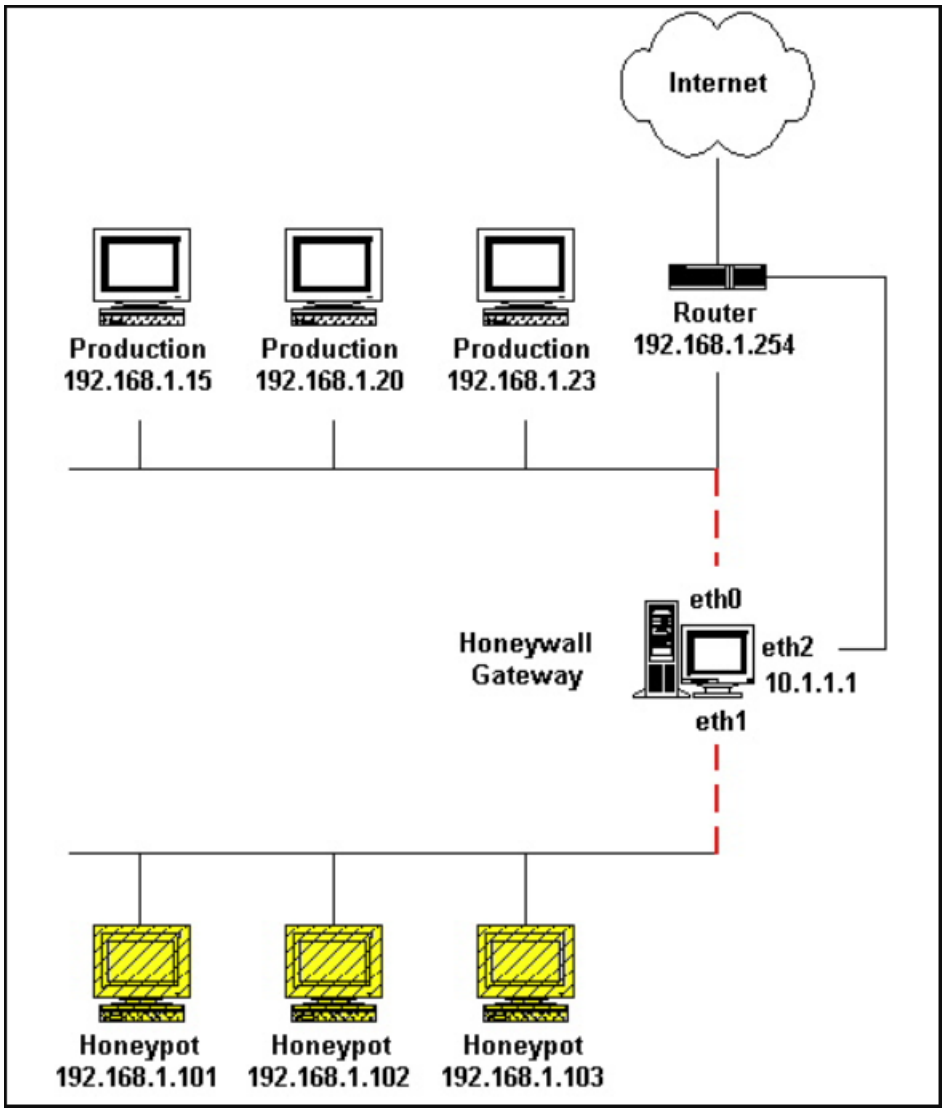


Figure 2.1 Honeynet Architecture [16].

According to document created by the Honeynet Project titled "*Definitions, Requirements and Standards*" honeynet architecture has several key requirements the most of important of which in relation to this thesis include Data Control, Data Capture and Data Collection [16]. While there is no specific way to implement a honeynet, it is important that the above requirements are met when implementing a honeynet.

Data control is used to mitigate risk and refers to containing the attacker's activity. The aim of data control is to contain the attacker's activity within the honeynet and not allow malicious code or activities to effect non-honeynet systems [16]. The challenge with data control is ensuring that an attacker does not detect it. An attacker must have a certain amount of functionality available to interact with and allow researchers to learn as much as possible from the attacker's activity. This essentially comes down to the level of freedom an organisation is willing to allow an attacker to have. It is recommended to implement data control in a layered approach such as enabling bandwidth restrictions and counting outbound connections, to ensure there is no single point of failure in the data control solution implemented [16].

Data capture involves monitoring and logging all the attacker's activity inside the honeynet, which is then analysed to learn more about the attacker's tools, motives and tactics [16]. Like data control, the data capture mechanism used needs to ensure that an attacker doesn't discover that their activities are being logged. The data captured is also normally encrypted since attacker's activity would be conducted over encrypted channels such as SSL. It is important that this is considered when deciding on the data capture mechanism. Another key element is to ensure that the activity that is collected from an attacker is not stored on the honeypot itself and is instead stored in a different location. This prevents data from being tampered with by the attacker or being completely deleted by a potential intruder [16].

Data collection involves collecting and aggregating the data a honeynet collects into a format that can be used by the organization and provide the ability to properly investigate the data collected [16]. This is not considered to be a requirement for standalone honeynet deployments and is meant to be used when there is more than one honeynet being deployed by an organization. The purpose of data collection is to centrally capture and aggregate all the information multiple honeynets gather, which could be sourced from disparate honeynets spread over different geographical areas. This can greatly increase the value of the information gathered by organizations if it is correlated correctly to the organizations needs [13].

2.2.4 Risks and Legal Issues

There are some risks and legal issues that are associated with the use of honeypots. Most of these legal issues are far too complex to elaborate in a single section and are usually dependent on the country in which the honeypot is deployed. However, it is useful to identify some of the legal pitfalls that can be associated with honeypots as well as some of the risks that are present. Mokube et al. discusses these ethical issues in detail in their paper [15]. Since the honeypots being used in this project will be deployed in America, it is useful to be aware of some of the major legal issues that need to be considered.

Entrapment is a concern when deploying honeypots since the honeypot is designed to entice an intruder to attack it. In this situation it could be argued that the honeypot was designed and implemented in such a way to cause the intruder to perpetrate the attack [15]. In a scenario where an organisation is using honeypots to not only detect and defend against attacks but to also use the evidence collected from the honeypot to pursue legal action, this could be an issue. However, this is unlikely to be used as a defence against a pure honeypot implementation as any activities against a honeypot is suspicious and would not be considered normal activity.

Privacy laws are another legal issue that can hinder a researcher’s ability to monitor the data collected from a honeypot. Countries may have restrictions that limit the monitoring of user activities on a system. This can range from government laws to organization policies. Breaching these privacy laws can result in criminal charges and should be considered before deploying honeypots with the intent to pursue legal action or retain information of activities for further research [15].

Liability risk is another legal issue that should be considered when deploying a honeypot. An attacker can gain access to a honeypot and use it to conduct malicious activities such as using it as a zombie in a botnet to perform denial of service attacks or as a drop point for contraband. This can lead to lawsuits from victims. Therefore, it is important to mitigate liability risk by regularly monitoring a honeypot and observing its activities [15].

2.3 Related Work

The closely related projects described in this section are explored from the point of view of their most interesting characteristics rather than the achievement of their respective research objectives. They mainly served as a source of inspiration and to see how they went about implementing their respective products.

2.3.1 Modern Honeypot Network

The Modern Honeypot Network is a free open source software which supports external and internal honeypot deployments at a large and distributed scale. The software seeks to simplify the deployment and management of honeypots for organizations, making it easier and more feasible to deploy honeypots for research and defensive purposes. The product provides a means to deploy honeypots and aggregate the data collected from these honeypots for easier analysis. A variety of different honeypots are supported such as the popular Dionaea honeypot and the Snort Honeypot [3].

2.3.2 Tpot

The Tpot project was developed by Deutsche Telekom, in which they use in their production networks for early threat detection and defence. Tpot provides Dockerized versions of a variety of honeypots like Dionaea and Cowrie. These honeypots are actively maintained and target a variety of different interactivity levels for attackers. All container images for the honeypots are built based on the Alpine Linux31 container image, which has a particularly small image size meaning that its storage footprint is minuscule. A point of note drawn from Tpot is the potential benefits of leveraging containers to host honeypots is an intriguing one, particularly since the difficulties of deploying and maintaining honeypots have been a huge barrier to their widespread adoption to-date [22].

Chapter 3: Methodology and Design

This chapter outlines the design of the IOT Honeynet Framework and elaborates on the methodology used when designing the framework. This chapter is broken down into different sections which includes deciding what design features the framework should support, issues that were encountered when designing the framework and a short summary on the final design choices made for the framework.

This chapter aims to provide a design for the IOT Honeynet Framework that solves the problems posed in chapter 3. An initial system design was created to accommodate all the objectives of the framework.

3.1 Concept

The outset of this project is to develop an extensible framework that can automate the cloning, emulation and deployment of an IOT device. With the vast number of IOT connected devices currently available worldwide, this project aims to produce a framework on which security research can be conducted specifically on the IOT sector.

The cloned device must replicate the services of the original device as precise as possible and behave in a similar manor. The clone must also present as an exact network copy of the original device when scanned with a port scanner. The framework will manage the deployment of the clone to a remote host as a member of a honeynet cluster.

Each member of the honeynet cluster will represent a node inside a fictitious LAN (Local Area Network) network. Inside the cluster, network monitoring methods will be implemented to capture network traffic from each cloned device. This traffic will be redirected back to the framework to be displayed visually.

To simplify the setup and deployment, a graphical interface will be utilized with intuitive options to simplify initial setup. Some guidance may be required on how the system operates and additional setup processes to be carried out.

On a broader term, the framework will clone any given device effectively to allow accurate emulation inside a honeynet cluster. All processes will be automated where possible, minimising user interaction and mitigating possible problems.

3.2 Features

The framework must be user friendly, informative and guided where necessary. The use of intuitive menus, clearly labelled buttons and a logical flow of data input should be adhered to. Many other features are required to complete this project and allow it to function as intended. This includes:

- Determine the platform.
- Automation of the scanning process.
- Storing the scan results.
- Segregation of important information required to build the clone.
- Automate building the clone.
- Deploy the clone to the intended host.
- Include a method of monitoring network traffic.
- Redirect network traffic back to the framework.
- Analyses the network traffic.
- Visually display a reformatted version of the network traffic.
- A method of controlling the clones remotely (Stop, Start, Delete).

Many issues are noticeable at this initial stage. Each problem will be addressed in a logical order, from the first stages of the project to destroying running clones nearing the final stage.

The platform, or host OS (Operating system) should be carefully decided. The OS should have good functionality and allow the user easily setup any additional dependencies that the framework may require. The three most popular OS systems, Windows, Mac and Linux all have their advantages and disadvantages.

To clone a device, some type of port scan is required to determine what ports are opened and establish the services running on those ports. Other information may also be required, such as version numbers or login prompts for terminal based sessions. Tests must be conducted to establish the most suitable method of recovering this information from the target device.

Data storage will be required at a certain stage to store scan results returned from the previous step. These files, along with other scripts, will be important to the rest of the project and must be stored in a suitable location. The location will be determined mostly by the framework file hierarchy setup.

Segregation of important files will be important so as to not interfere with framework functionality in any way. Scripts will most likely be developed to automate certain areas and will require execution at key stages. This must also be segregated from framework files and any files created as a result of the script execution must also be controlled appropriately.

Automation will greatly increase the productivity of the framework and reduce the user interaction. The method of automation will depend on the host OS chosen. Microsoft Windows use a command line scripting language called PowerShell, whereas Linux and Mac use a command line scripting language called Bash. Further research will expose any favourable tendencies that can simplify the framework operation.

3.3 Problems

This section elaborates on the problems encountered during the design of the IoT Honeynet Framework.

3.3.1 Emulating services

The IOT sector includes billions of connected devices with an aim to connect objects to computerised devices to automate simple tasks. Billions of these devices exist and each one employs similar services to communicate with internet bound devices. These are the services that this framework aims to emulate on each device. These services produce unique signatures that port scanners can accept and determine what service is running.

After successfully acquiring the version of service running on the target IOT device, this service must then be emulated on the cloned device. To achieve this in a low interaction honeypot type system, the banner messages must be displayed when connection requests are made. Banner messages are strings of text containing the version type of the service running on a given port. This gives any connecting device the information required to connect successfully. To emulate these banner messages, they must be captured and stored along with the port number associated with the banner message.

Emulating the services of an IOT device can prove troublesome at best. The level of interaction is an important determination at this stage. High interaction honeypots require a high level of interaction which includes login prompts for Telnet services and actual services running on other ports to interact with. Low level honeypots are a much simpler affair. They require minimal interaction. This also means that no login areas or actual services need to be running. To achieve a low-level honeypot, banner messages are displayed when a connection request is made to a particular port.

To return these messages when a connection request is made, a service of some type must be running to accept the connection and return the banner messages. This emulated service must run repeatedly until the service is explicitly terminated. Although many methods exist to accomplish this task, determining which one would be more suitable for this framework will depend on what features are decided upon. Many features implemented will depend greatly on which OS system is decided upon. This alone will determine which selection of software tools are at the disposal of the framework.

3.3.2 Port issues

All honeypots, no matter the deployment environment, requires some degree of networking to allow the honeypot to be recognised on the network and to capture network related interactions between honeypot and the internet. The most valuable information retrieved from any honeypot is the network traffic. The captured traffic identifies the origin of the attack, the destination port the attack is aimed at, and with high interaction honeypots, the type of attack used, files that may be downloaded by the attacker and most importantly, the behaviour of the attacker while inside the honeypot.

Although this information is very important, with a honeynet cluster, multiple devices will be setup on a single host to simulate a LAN configuration. This serves many networking issues. The most important and concerning is port allocation. Each cloned device will be required to open ports based on the target device. If the targeted device has an identical port as a previously cloned device, the port cannot be opened again. Opening similar ports on multiple devices will cause a port clash. A port cannot have multiple instances opened on a single machine. This scenario will cause fatal errors when running the second device and will need to be addressed.

3.3.3 SSH port

Port 22 (SSH port) is an important port required by almost all servers as a means of remote login. This port can be reallocated to another port number if a cloned device is required to use port 22 open. The issue resides with having an additional port open that is not opened on the cloned device. When a scan is completed of the cloned device, an additional port will be identified from the scan, which might forewarn the most attentive of attackers. The biggest issue is this port must remain open to allow remote login to control, install, upgrade and deploy any devices running on the remote host.

3.3.4 Visualisation of data

This framework provides the user with a mean of cloning any IOT device that they can scan and deploy the device on a remote host to begin capturing network traffic from potential intruders from the world wide web. On the Internet, many automated scans and scripts run continuously, attempting to identify possible vulnerabilities that can be exploited. This generates large amounts of network traffic on each device that will need to be filtered down to what is useful, and what is not. Traffic originating from within the host device should also be removed.

When the filtering of unimportant traffic is complete, a method of displaying the results inside the framework that allows the user to easily understand what is being displayed. Valuable information, such as source addresses and destination ports should be clearly displayed. This display may also have additional filtering to allow the user control what data they would prefer be displayed.

3.4 Addressing challenges

This chapter focuses on the problems encountered throughout this research while trying to develop the IOT Honeynet Framework and how each major issue is addressed.

3.4.1 Emulating services

The challenge is to emulate the services of an IOT device effectively enough to fool unwitting attackers into thinking it's a real device. This poses many challenges. The majority of these challenges revolve around successfully displaying the correct services on the correct port.

In order to achieve this, a mechanism must be developed to resemble server characteristics. This will require one of a couple of approaches.

The host operating system can be manipulated in a way that it resembles the functionality of an IOT device. This approach will require the installation of correct software versions that the scanned device is running, and if possible open the ports that the scanned device has open while closing all other currently open ports. This approach will have its own issues when attempting to alter the open ports. The OS will have ports open for internal services that will be difficult to close without compromising the OS itself.

Another approach will involve using a software developed server which can send and receive data across the network. Most modern languages have modules that handle network sockets and interconnected functionality. A software approach will give the versatility needed to open a varying number of ports each time the sever is run. Additionally, this would provide the functionality and flexibility to use any files created from the scanning process and integrate the results into how the server will function. With most modern programming languages, threaded programming is available to increase functionality in regard to the number of connections the server can accept at any given moment. Many parallel connections are expected and should be adequately handled by implementing some type of threaded processes.

3.4.2 Port Issues

Port clashing is a large concern and so needs some attention to adequately mitigate any chance of clashing. With no way to run multiple instances of each port, a method of applying an IP address to each clone is seen as a viable option. If each clone is separated by IP, the clone will have the freedom to open any port needed without any possibility of clashing. Each IP address can have any number of ports open and this is separate to another IP address. Similar to an internal LAN setup, this will mean the need for a DHCP (Dynamic Host Control Protocol) server to assign IP's to each device. The DHCP server assigns IP addresses to each device from a preconfigured pool of addresses. The DHCP server will ensure no devices are assigned the same IP as any other device. However, this will have the honeynet cluster behind a type of router and therefore harder to expose directly to the internet. The router would need to be intentionally vulnerable to allow attackers pass through without much resistance and access the honeynet cluster.

A variation of the previously mentioned implementation could also work. Cloud hosting vendors allow multiple instances to be created on each account. This means that when signed up, a user can have multiple OS's running on a separate instance. An instance is when multiple, separated OS's run on one piece of hardware. Each instance is self-sufficient and cannot interfere with another instance. Most importantly, each instance is assigned its own dedicated IP address. This will benefit this project as a new instance can be created for each cloned device. Although each instance has its own OS, the ports and services of the OS will need configuration.

Many technologies exist to aid the deployment of software bundles across multiple platforms. Containers are a software created container that contain all the code and dependencies required to run an application. This type of implementation will allow the software server to be platform independent and help remove some platform dependant issues. The networking of containers are slightly different. Each container is assigned a separate IP address when created and requires additional settings to map container ports to host OS ports.

3.4.3 SSH Port

The SSH port is an important element of remote servers. Each remote server requires a means of remote login for configuration and setup purposes. This port should not be publicly viewable and needs to be hidden from any scans that may be conducted. Furthermore, to establish a realistic clone environment, the port needs to be hidden from any scans conducted. The most effective method of achieving this is using port knocking. Port knocking is a method of hiding the port from any scans conducted on the network. This is accomplished by closing the SSH port from being accessed from the outside network. To open the port, a daemon running on the server is listening for a well-defined sequence of knocks (connection attempts) on specific port numbers in a specific sequence. If the sequence matches exactly the predetermined sequence, then a set port will open, if the sequence does not match, the port will remain undisclosed. The sequence can be of any number of knocks on any port number.

3.4.4 Visualisation of Data

Another equally important feature of the framework is how the results are visualised back to the user. As with many honeynets, the network traffic is the most important part of the project and should be carefully displayed, containing as much information as possible. The user must be able to control what is displayed and what is unnecessary in the scope of their use. With the large amounts of data that is returned on a continuing basis, storing the data will be avoided if possible and live filtering will be used.

In order to display the information in a manor beneficial to all users, a means of filtering the display would allow users to specifically choose what they wish to see. This type of layout will allow the user to view more relevant traffic data and filter out nonessential data. The honeynet traffic will be filtered to exclude generally irrelevant information such as account id but allow the user to decide what additional information is useful by displaying remaining categories. With the ability to run numerous clones at any given time, the data relating to each clone will be displayed on individual pages to ensure adequate room for all the information.

3.5 Hosting Service

In this section, the decisions made for the hosting service selected in relation to its design is explained. It was decided to use Amazon Web Services (AWS) as the Hosting platform. Amazon Web Services (AWS) is a subsidiary of Amazon Inc. and describes itself as a “secure cloud services platform, offering compute power, database storage, content delivery and other functionality” [23]. The project makes use of Amazon EC2, which describes itself as being able to provide “scalable computing capacity in the Amazon Web Services (AWS) cloud” [24]. Amazon EC2 allows developers to deploy as many or as few virtual servers as is needed, configure their networking and security, and manage their storage.

Amazon EC2 was chosen due to its flexibility, making it possible to deploy and manage virtual servers on demand. Other hosting providers such as Digiweb, Microsoft Azure and DigitalOcean were considered but Amazon EC2's flexibility and prestige as being an industry leader in providing cloud hosting services resulted in it being chosen for this project.

Another reason for choosing Amazon for this project was that Amazon has its own SDK called boto3 which allows system administrators to automate a wide variety of tasks. The boto3 library is used to aid in providing a means for the user to interact with their AWS account through the web framework used by this project.

3.6 Understanding Amazon EC2 Features

As previously mentioned, Amazon EC2 describes itself as being able to provide scalable computing capacity in the Amazon Web Services (AWS) cloud. Amazon EC2 instances can be provisioned under the AWS 12-month free tier to facilitate the frameworks need to deploy multiple instances. The features that make up Amazon EC2 and their relevance to the project are explored in further detail in the following sections. These features influence the type of hosting service the framework uses and how a user goes about deploying and managing their honeypots on a public network.

It is important to understand the different features provided by Amazon EC2 to comprehend the design of the IOT Honeynet Framework and all of its associated features which help researchers and users to seamlessly scan devices and deploy them as honeypots quickly and efficiently. There are a number of requirements that must be satisfied in order to allow such an IOT Honeynet Framework to function properly. It is for this reason, that the features of Amazon EC2 must be explored and understood.

3.6.1 Setting Up Amazon EC2

Using a free tier EC2 instance first involves setting up an AWS user account. Once created, an account has access to all AWS services automatically, including Amazon EC2 [25]. Once an account is setup, the next step is to create an IAM user. Services in AWS, like Amazon EC2, require users to provide credentials when accessing them so it can be determined whether a user has permission to access that service or not [25]. It is not recommended that an AWS service be accessed using the default AWS account login credentials but rather by using the AWS Identity and Access Management instead, which is necessary to allow services to be accessed programmatically [25].

3.6.2 Virtual Private Cloud (VPC's)

In Amazon, a Virtual Private Cloud (VPC) is defined as "creating a virtual network in your own logically isolated area within the AWS cloud, known as a virtual private cloud (VPC)" [26]. Amazon EC2 instances can be launched in a VPC which acts similarly to a traditional network but with the added bonus of being able to use scalable infrastructure from AWS [26]. When an AWS account is created, a default VPC is created in the region specified when setting up the AWS account, enabling users to instantly launch instances.

It's important to understand the concepts of VPC as it is the networking layer for Amazon EC2 instances. Amazon provides every user who sets up an account with a default VPC which is configured and ready for use. The default VPC conveniently includes an internet gateway, and each default subnet is a public subnet [27]. Each EC2 instance that is launched on a default subnet has a private IPv4 address and a public IPv4 address. These instances can communicate with the internet through the internet gateway. An internet gateway enables your instances to connect to the internet through the Amazon EC2 network edge [27].

It is possible to create nondefault VPC's, but it was decided to work of the default VPC since it satisfies the requirements needed by the IOT Honeynet Framework, which is to allow honeypots access to the internet and to collect logging information, something that will be discussed in more detail later. This reduces the amount of setup and configuration required by the user and mitigates any errors in configuration that could hinder the overall operation of the framework.

3.6.3 AMI's and Instances

According to Amazon, an Amazon Machine Image (AMI) "is a template that contains a software configuration (for example, an operating system, an application server, and applications)" [28]. It is from this image that an instance is launched, which is a copy of a virtual server running in the cloud. There are multiple AMI's that can be selected, with some being a part of the AWS 12-month free tier, while others being more expensive.

An important consideration to make when launching an instance is the type of instance that is being launched. The instance type that is specified determines the hardware of the host computer used for your instance, offering different compute, memory, and storage capabilities [29]. These instance types are often grouped based on their capabilities and can dramatically impact the performance of the application or software being deployed on the instance.

For the base operating system of each honeypot, it was decided to only use the free tier AMI's and allow the user to select from a pre-determined list, which operating system they would like each honeypot to run. This approach is used in response to the fact that depending on the region in which a user has setup their AWS account, certain AMI's may not be available and are unique to specific region. This approach allows the user more options while also keeping in line with the AWS 12-month free tier contract. The different instance types that were allowed for a user to select from were kept to a small t2.micro general-purpose instance type. This was chosen based on the minimum resources that were required for each honeypot to operate in the Honeynet framework, meaning more honeypots could be deployed without unnecessarily using resources. This aimed to incorporate efficiency and cost-effective elements to the framework.

3.6.4 Regions and Availability Zones

According to Amazon, Amazon EC2 can be hosted over multiple locations world-wide, with these locations being composed of Regions and Availability Zones [30]. Each Amazon region is designed to be completely isolated from other Amazon EC2 Regions. Each region is then comprised of isolated locations known as Availability Zones [30]. This is meant to achieve fault tolerance and stability, two very important requirements for the hosting service of a Honeynet as any downtime could result in valuable data being lost for researchers.

The choice of what region to deploy the Amazon EC2 instances could greatly impact the type of results and data gained by a researcher. It is an important consideration to make when determining what region to deploy Amazon EC2 services in. The goal of the IOT Honeynet Framework is not to make these decisions for the researcher but to rather support the ease of deploying and managing instances for the researcher. It is with this in mind that it was decided to allow the user to make this decision when initially setting up an AWS account. The framework would provide a list of AMI's to select from based on which region the user chose.

3.6.5 Amazon EC2 Key Pairs

A key pair refers to the public and private key used to encrypt and decrypt data, which is used by Amazon EC2 to encrypt and decrypt login information [31]. This enables secure remote access into an EC2 instance. Amazon stores the public key while the private key is kept by the user which is why it is important to keep the private key secure as anyone with access to the key can decrypt the login information of any instance associated with that particular private key. The keys that Amazon EC2 uses are 2048-bit SSH-2 RSA keys and it is possible to have up to five thousand key pairs per Region [31].

It is important to create a key pair to securely access a honeypot instance from a remote location once it has been created. Once created it is necessary to change the mode of the key pair file to read-only otherwise it will be denied access. It is possible to create key pairs in the IOT Honeynet Framework which are stored in a file with a .pem extension and then later used when creating the instance. This will be demonstrated programmatically in the Implementation Chapter, but it is necessary to understand the importance of the EC2 key pair for logging in securely to newly created instances and is instrumental for allowing the transfer of files necessary for the setup of the python server used for the honeypot.

3.6.6 Security Groups

Amazon describes how a security group “acts as a virtual firewall that controls the traffic for one or more instances” [32]. When an instance is launched, one or more security groups can be assigned to it. A security group contains rules which dictates what traffic is allowed to and from an instance. Security groups are associated with network interfaces and when changing an instance's security groups, it changes the security groups associated with the primary network interface (eth0) [32].

The security group rules dictate what inbound traffic can reach an instance and what outbound traffic can leave an instance. There are a number of key characteristics that a security group has which are important in acknowledging when deploying an instance:

1. Security Groups allow all outbound traffic [32].
2. It is not possible to create rules that deny access [32].
3. Security groups are stateful meaning it tracks the operating state and characteristics of network connections traversing it. The firewall is configured to distinguish legitimate packets for different types of connections [32].
4. Rules can be added and removed at any time [32].

A rule in a security group is created with several parameters. When the IOT Honeynet Framework creates the security group, the protocol type, source and destination port, and the source address parameters are used. This will be demonstrated further in the Implementation Chapter. It is essential to know how to configure a security group to open ports for the honeypot based on what scan results are provided by the user [32]. The security groups and the ingress rules of the security groups themselves will be generated automatically based on information that is retrieved from the scan.

3.7 VPC Flow Logs and CloudWatch

VPC Flow Logs and CloudWatch are separate topics from Amazon EC2 and deserve their own section to better elaborate on their importance to the IOT Honeynet Framework. Logging is an important and vital function for any Honeynet Framework service. For this thesis, it was evident that if Amazon EC2 was being used by the IOT Honeynet Framework then VPC Flow Logs and Amazon CloudWatch were going to be used for collecting log information from each honeypot. It is therefore important to understand the process to setup each feature and how both relate to each other. It is also essential to be able to understand how to read the information that is being stored, hence why it is necessary to provide a separate section for both VPC Flow Logs and CloudWatch.

3.7.1 VPC Flow Logs

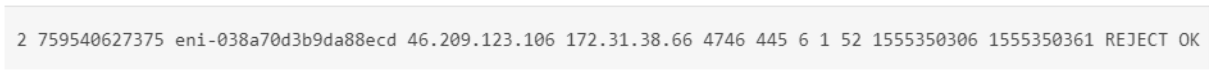
VPC Flow Logs is a feature that captures information about the IP traffic going to and from network interfaces on a VPC. This data is then published to either Amazon CloudWatch Logs or Amazon S3 [33]. Setting up a flow log involves specifying a resource to capture data from, the type of traffic to capture and where to send the flow log data once it is collected.

An important point to note is that if a flow log is setup for a VPC with multiple instances on it then each network interface on the VPC is monitored and stored in that flow log [33]. This data that is collected is stored in what is called flow log records which consists of fields that describe the traffic being monitored on a network interface.

The structure of the flow log is quite detailed and so requires users to understand the different fields when examining a log record. A flow log record represents a network flow in your flow log and is a space-separated string that has the following format [33]:

<version> <account-id> <interface-id> <srcaddr> <dstaddr> <srcport> <dstport> <protocol> <packets> <bytes> <start> <end> <action> <log-status>

Amazon [33] provides a table containing the description for each field. This format for a log record can appear to be quite complex and detailed at first but it provides a wealth of information to a user who wishes to see what traffic is being sent to their honeypots. A sample log that was stored for one of the test honeypots used during the development of the IOT Honeynet Framework can be used as an example to showcase what information is being gathered. Below we can see a sample log:



```
2 759540627375 eni-038a70d3b9da88ecd 46.209.123.106 172.31.38.66 4746 445 6 1 52 1555350306 1555350361 REJECT OK
```

Figure 3.1 Sample Flow Log Record

The first three fields relate to account information and displays the VPC Flow Logs version (2), the AWS account ID for the flow log (`759540627375`), and the ID of the network interface for which the traffic is recorded (eni-038a70d3b9da88ecd). This information doesn't serve much purpose for researchers, but it is was deemed necessary to be aware of these fields in order to be thorough in explaining flow log records.

The following fields are of more interest to researchers, which contain the vital data that can be used later to determine what kind of attacks are being used and where these attacks are coming from. The source address in this flow log record is 46.209.123.106. A WHOIS lookup conducted using the website BigDomainData [34] showed that the address was from the geolocation of the Islamic republic of Iran. The next field shows the private IP address (172.31.38.66) of the honeypot that received this traffic. The next two field are the source port (4746) and destination port (445) respectively. The next field shows the IANA protocol number 6 which represents the type of protocol used, which in this case according to iana.org is the TCP protocol [35]. The following field represents the number of packets sent, which according to this log is one. This is followed by the byte size of the packet which was 52. The log also provides the start and end times of the capture window in Unix seconds. Finally, the action and log-status are recorded by the log which shows that the action taken against the packet was to reject it, and then store the log successfully (denoted by 'OK') to its intended destination with the accompanying data.

For the IOT Honeynet Framework, it was debated on how to display the logs to the user. As an essential component of the IOT Honeynet Framework, it was decided to present the log records as provided by Amazon Flow Logs to the user. This would allow users to observe and work with all provided information in a log. This avoided the possibility of hiding or withholding information that a researcher/user may want access to. However, this could also hinder a user's ability to interpret the log information and cause relevant data to be obscured by irrelevant data. In the end the decision to provide all given information was taken to ensure all log information was provided to the user and could be used at their discretion.

3.7.2 Amazon CloudWatch

As mentioned earlier, when setting up VPC flow log there are a number of steps taken, one of which is to choose where to send the flow log data once it is collected. There are two options available when considering where to send the flow log data once it is collected which include Amazon Simple Storage Service (S3) and Amazon CloudWatch.

Amazon S3 is designed to allow a user to store and retrieve any amount of data using a web interface from anywhere on the web [36]. Amazon S3 provides a means for customers of any size in the industry to store data and then take advantage of easy-to-use management features to organize the stored data and finely tune access controls [36].

Amazon CloudWatch is promoted by Amazon as "a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers" which "collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications and services that run on AWS, and on-premises servers" [37].

For this thesis, it was decided to use Amazon CloudWatch over Amazon S3 due to Amazon CloudWatch being designed to visualize logs, troubleshoot issues and analyse system metrics. This is in comparison to Amazon S3, which focuses on storing data from applications which can then be used in a variety of use cases such as backup and recoveries or big data analytics [36]. The process for implementing CloudWatch with VPC Flow Logs will be explored in more detail in the Implementation chapter.

3.7.3 AWS IAM Policies and Roles

IAM Policies as described by AWS documentation "is an entity that, when attached to an identity or resource, defines their permissions" [2]. These policies provide permissions that can be attached to users, roles, and resources which allows them to perform certain tasks. In terms of the IOT Honeynet Framework, permissions are important as they allow an IAM user setup by the account holder to manage AWS services. This is essential for the framework which uses the access credentials provided by the user to perform tasks such as deploying instances or when it comes to logging, assign permissions to the VPC and IAM user to allow them to store the logs in what is called a log group, which will be mentioned again later in the implementation chapter.

An IAM role is an entity that defines the permissions for making AWS service requests [38]. IAM roles are not associated with any user or group but instead are assumed by entities such as IAM users, applications and AWS services like EC2. These roles can have policies attached to them which are then used to define the permission of the role. These roles are used to assign logging functionality to services and users as well as allow administrator access for the user. It was decided for this project to allow the user to configure the administrator role for their IAM user but the policy and role for logging are handled through the framework. The user needs to provide the administrator IAM users details in order for the framework to function hence why this is handled by the user and not through the framework. The policy and role for logging can be handled through the framework, once the IAM user credentials are set.

3.8 Docker

Docker is a software platform that creates a virtual environment for Docker containers to run within. This allows for quick testing and deployment of applications across multiple operating systems without the need for additional changes to accommodate for each operating system (OS) the container will run on. Docker acts as the OS for each container and provides a simple method of building, running and stopping containers, much like a virtual machine.

Docker uses containers to package all necessary resources into an environment, that can then be deployed cross platform [39]. Each container is built with all resources the application may need installed inside the container and can be a completely self-sufficient.

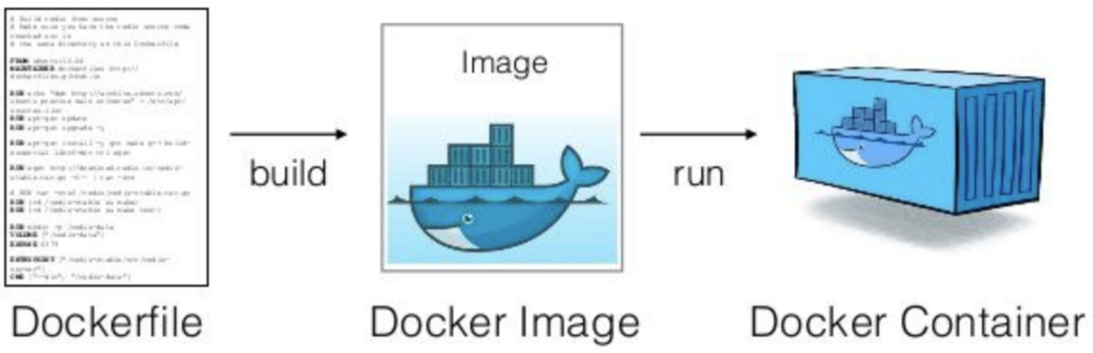


Figure 3.2 Docker build process

Working with Docker involves understanding a few key concepts. To deploy a container, a Docker image first needs to be created using a Dockerfile. A Dockerfile is a text file containing all the necessary commands required to create a Docker image [40]. A Dockerfile contains many commands, including FROM, COPY, RUN, CMD.

Once a Dockerfile is defined, a Docker image can be created. A Docker image is an executable package, created after execution of a Dockerfile using resources specified in the Dockerfile. The images contain all the necessary code required to complete the task the image was created for [41].

With a Docker image now created, it is possible to deploy a Docker container. A Docker container is “A standardized unit of software” [42]. Each container is an instance of a Docker image. Multiple instances can be created from a single Docker image.

3.8.1 Building a Docker container

The Docker build process consists of multiple steps to create a Docker container. A file containing optional parameters, called a “Dockerfile” is created first. This text file contains all the necessary instructions required to build the image, install/update any additional software and determine where the entry point should be [43].

The ENTRYPOINT command is used to determine what script should be run when the container is launched.

The COPY command can copy any needed files into the image.

The EXPOSE command can determine what ports should be exposed to the outside world and what ports should be kept local [44].

This file also allows for automating the use of the latest versions, mitigating vulnerable older versions from being used by using the Docker command

FROM OS: latest. The ensures the latest version is always used every time the image is built.

To build the Dockerfile into an image, a simple Docker command is used which can also take optional parameters to achieve varying scenarios.

```
$ docker build -t <image_name> .
```

This command builds a Docker image called image_name in the current directory. The current directory is denoted by the dot (.) operator. There must be a Dockerfile present to build the image with the dot notation at the end of the command indicating the current directory.

The Docker build command is used to build the Docker image when the Dockerfile is prepared. This command uses the Dockerfile to setup the image using whatever parameters were defined inside the file. At this stage, the image imports any base image, updates and imports any additional files required to run the application. When the build successfully completes, the image is ready to run.

Docker run is the command that transforms the Docker image into a Docker container. This command also has additional parameters that can be passed while starting an image. Parameters such as Docker volumes (shared storage with host OS for persistence), the opening of required ports and networking options can be specified at this stage [45].

```
$ docker run -p 80:80 <image_name>
```

This sample Docker command starts the Docker image and opens port 80 on the container.

Docker containers provides a wide range of benefits, especially when it comes to using containers to deploy honeypots. The proposition of leveraging Dockers flexibility, portability and scalability is highly advantageous, especially when it comes to deploying honeypots, since the deploying and maintaining honeypots has been a huge barrier to the widespread use of them.

3.9 Python Socketserver

Python3 is a high-level, interpreted programming language for general purpose programming. Pyhton.org contains comprehensive documentation on the language, complete with introductions and tutorials. Source code and installers are freely available to download for all platform [46].

Figure 3.3 Python image [47]

Python3 socketserver module simplifies the writing of network servers. With this module, Transmission Control Protocol (TCP) servers and User Datagram Protocol (UDP) servers can be easily created using server objects contained within the socketserver class [46]. These objects add functionality that make the class more usable. Other methods in the class are designed to control the server, define how the server reacts when a connection is made and how to respond to that connection.

Additional handler classes are required to complete the server and add functionality that is required to handle data transfer. The socket-server class contains a method for this exact function. The socketserver.BaseRequestHandler class provides the ability to send and receive data easily across the network and increases the functionality of the server object [48]. This class will allow a response to every connection request.

3.9.1 TCP Server

TCP (Transport Control Protocol) is a connection orientated transport protocol. The connection process is known as the “Three-way handshake”. After the connection has been established, data is broken into packets for transportation across a network. Each packet contains a sequence number which is used to reassemble the packets upon delivery. Additionally, each packet is acknowledged as it is received, if a packet is lost or corrupted, the packet is requested again to construct a reliable service [49].

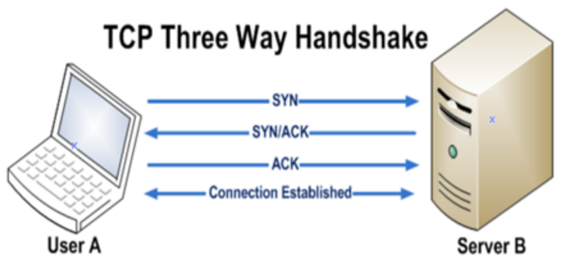


Figure 3.4 TCP Three Way Handshake [50]

3.9.2 UDP server

UDP (User Datagram Protocol) is a connectionless transport layer protocol. This means no connection is established before sending data and no effort is made to ensure all packets are received. This type of connection results in less overhead to TCP, which does provide packet integrity checks [51]. The lack of integrity in UDP makes the protocol more efficient when speed is more important than integrity, such as video streaming of VOIP (Voice Over IP) calls.

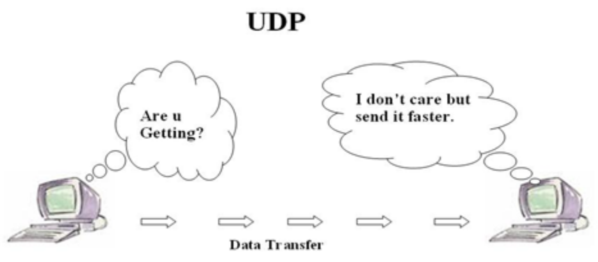


Figure 3.5 User Datagram Protocol [52]

3.9.3 Threaded programming

Python3 can easily incorporate threaded functionality into a program which can increase the overall stability and efficiency of a Python program. Python threads allow a program to run several processes concurrently. Each thread is a separate flow of execution [53]. Python currently have two threading modules, “thread” and “threading”. The “thread” module is considered deprecated but was renamed to “_thread” for backwards compatibility [54]. With the “thread” module, every thread is considered a function, where with the “threading” module, each thread is considered a separate object with its own functions.

3.10 Bash

Bash is a command language interpreter for the GNU (GNU's Not Unix) operating system and comes default on almost all Unix and Linux operating systems [55]. This language is very powerful and interacts with the kernel, making system calls in order to complete user commands. To complement bash, many other utilities are freely available to further enhance the functionality of Bash and extend the usability. Utilities such as SSH (Secure Shell), SCP (Secure Copy), CAT, among many others, enhance the usefulness and add functionality which makes bash even more powerful.

Bash scripting can be used to automate mundane tasks, such as tasks that are run quite often and require a similar group of commands each time. These should be automated for efficiency and simplicity.

Scripts are files consisting of various commands that are executed sequentially from top to bottom. Commands can be stringed together to ensure completion before running the next command. The file is saved normally with a .sh extension, as a convention. While using the shebang line as the first line in the script, a .sh extension is unnecessary. The extension is only necessary with the absence of the shebang line.

“Shebang” line. E.g. #!/bin/bash. This line is used by the shell to interpret what program is to be used, i.e. bash [56]. Any lines after that can be whatever is needed. Any lines that begin with # are ignored by the interpreter and are treated as comments. Comments are used to explain the flow of the code and what variables are used for within the code.

E.g. # This is a valid bash comment and will be ignored by the interpreter.

The only other types of text used are bash commands, and these can be in many different structures depending on what logic is being implemented. The bash script structure is as follows.

```
#!/bin/bash
# This is a comment
Command [parameter] [parameter]
```

Parameters can also be easily passed to Bash scripts using the $1 $2 $3 positional parameters. These parameters allow arguments to be easily passed to a script, used directly, or passed to other variables. E.g. var=$1.

$sudo apt-get update && sudo apt-get upgrade -y.

Where a logical "&&" (AND) is used, this ensures the first command completes without error before running the second command. The second command will not run if the first command produces an error. Another method to test a command is to run a command using the "||" (OR) operator. This will try to run the first command, if an error is produced, the second command is attempted. If the first command runs successfully, the second command will not be run at all.

$ apt-get upgrade -y || sudo apt-get upgrade -y.

SUDO or "su 'do'" or "Super User do", is a method in which administrators on a Linux or Unix system can allow ordinary users run commands that require escalated privileges [57]. Some command line applications require escalated privileges to correctly run as intended, such as Wireshark, a network protocol analyser application which requires sudo privileges to access the network interface on which to capture data.

3.11 NMAP

NMAP (Network Mapper) is a free and open source network mapping tool for network discovery and network security audits [58]. Currently over twenty years in production, NMAP is by far the most widely used network mapping utility that has won many awards such as ""Information Security Product of the Year" by Linux Journal, Info World and Codetalker Digest" [59].

Nmap includes a suite of additional tools that further enhance the usability and functionality for all users. Its primary CLI (Command Line Interface) design can command all the power NMAP has to offer, allowing intricate changes to be made easily with a simple command line layout. The layout consists of the keyword Nmap, then optional switches, followed by the target IP (Internet Protocol) .

```
$ nmap -sS -p- ip_address
```

The syntax can be viewed from the Nmap man page. The man page, short for manual page, displays all possible Nmap arguments, available options and output formats [60]. Sample commands are also viewable to demonstrate the expected syntax.

Other tools available from Nmap include a GUI (Graphical User Interface) version called ZENMAP. This tool provides an easy to use interface which simplifies the scan choices with drop down menus and other options. One really good feature of this tool is the ability to view the command that is being executed.

Nmap also provides another tool called Ncat, a similar tool to the better known Netcat tool. Ncat is a networking tool that can read and write data across networks using either UDP or TCP.

3.12 Web Framework Design

Flask consists of three components based on the Pocoo project. Werkzeug which contains the WSGI (Web Server Gateway Interface) web application library with services like routing and debugging [61], Jinja2 a templating language modelled after Django's templates [62] and Click, a command-line interface executed from the terminal that allows access to built-in extensions and application-defined commands [63].

Flask has no database abstraction layer, form validation, user authentication or any high-level functionality that is required by web applications today. However, Flask allows for all of these extensions to be integrated with its core packages. Meaning a user is not bound to a specific extension and has the ability to use extensions that work best with their project in contrast to larger frameworks where most of the extensions are built in and are pre-configured with the core packages making them hard to change [64].

3.12.1 Flask vs Django Comparison

When it comes to building web pages in Python there are two main frameworks, Django and Flask. Flask and Django are two of the most popular web frameworks for Python, while Django is the more popular choice for web development in Python due to the fact that it's been around since 2005, Flask is also quite a popular framework which was first released in 2010 [65].

The choice however falls to the developer and a decision can be made based off the overall expectations for the web framework. Django focuses on the final product and provides a fully-featured framework. Flask on the other hand focuses on the experience and learning opportunity as well as providing a simple framework with a wide range of extensions [65]. Flask has a concept of doing one thing at a time and doing it well and with Flask it rarely ends up with extensions that are not necessary. Flask also has a much lighter footprint, hence providing short and simple code that's easy to interpret and understand. The web framework for this project is expected to handle cloning scripts of IOT instances as well as have support for AWS (Amazon Web Services) and therefore knowing that Flask has a more extensible framework it was the better framework to choose for this project.

3.12.2 Web Server

Flask is an open source BSD licensed microframework written in Python by Armin Ronacher [66]. Flask is small enough to be called a microframework meaning once its developer understands how it works then they should be able to understand its source code. It was decided to use Flask as the web server for this project. Being a small framework Flask is designed for great scalability and growth. Flask comes pre-installed with basic core services and allows the developer to control its extensibility [64].

Flask was chosen due to its scalability and extensibility, making it compatible with the scripts associated with this project. Django, a popular Python web server was also considered as an alternative to Flask. However, Django being a larger pre-configured framework gives the developer less control and freedom over the frameworks extensions and therefore could have been potentially incompatible with this project.

3.12.3 Routing System

Flask uses the Werkzeug routing system which has an implicit design of ordering a web applications routes based by their convolution. This means that routes can be placed in random order and will still work fine in an application. This is a prerequisite if you want to appropriately implement a decorator into an application as decorators are supposed to work in an undefined order when an application is split into multiple py files. The Werkzeug routing system also plays a vital part in applications design by ensuring that routes in an application have a unique URL. Werkzeug is considered as a rather smart routing system as it will even indistinctively redirect a user to a "canonical" URL route if a user tries to redirect themselves to an ambiguous route [67].

3.12.4 Key Objectives

The objective of this framework is to combine the elements of this projects such as scanning a target device, creating a clone, deploying a honeypot, displaying the honeypot and displaying all these elements on a single web framework for user convenience and ease of access.

3.12.5 Framework Structure

A Wireframe was used to design the web framework structure. The figures portrayed below are observations of the project's framework.

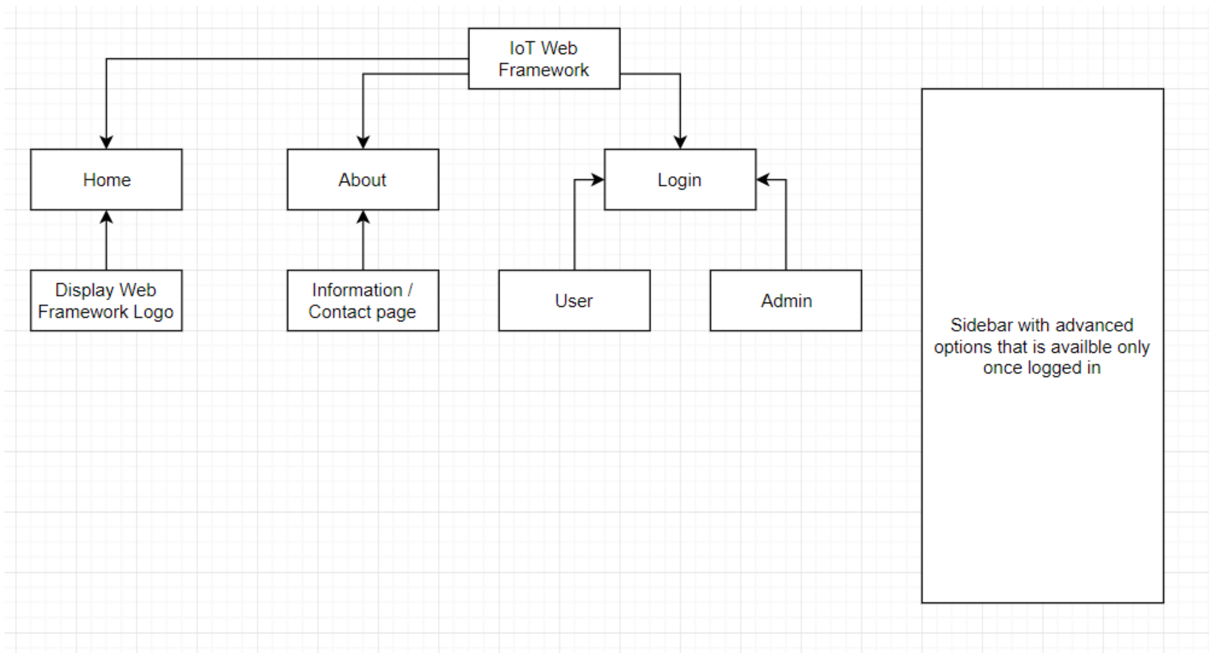

IoT Web Framework
Home
About
Login
Display Web Framework Logo
Information / Contact page
User
Admin
Sidebar with advanced options that is availble only once logged in

Figure 3.6 WireFrame of the Main Web Framework

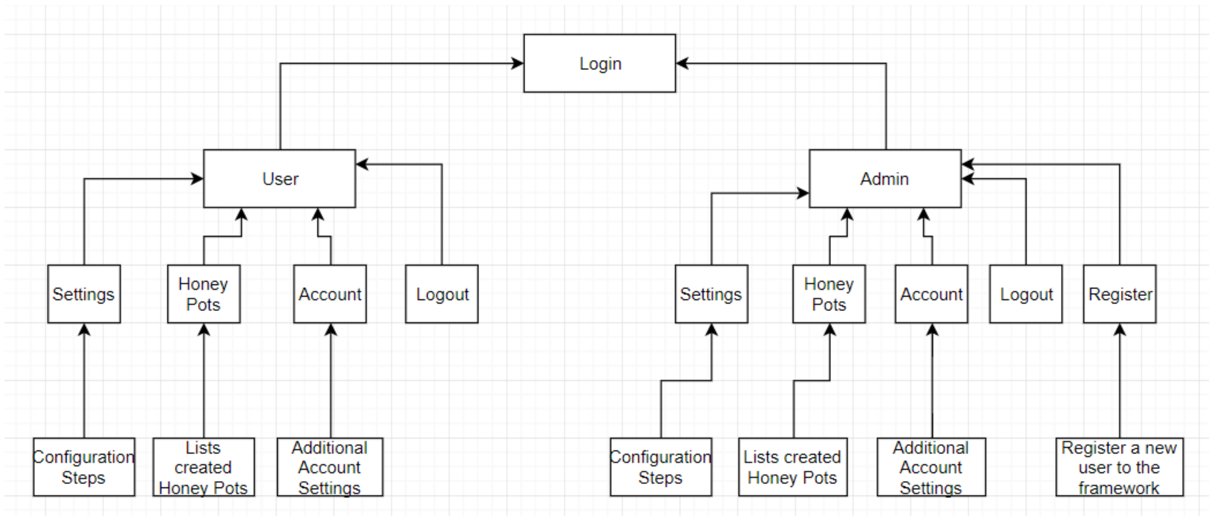


Figure 3.7 WireFrame of the Web Framework's Login Function

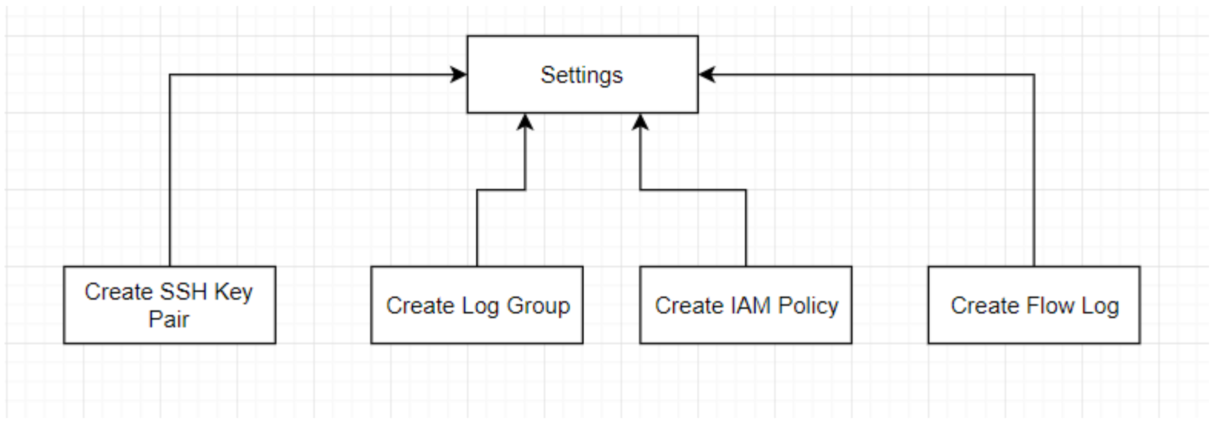


Figure 3.8 WireFrame of the Web Framework's Settings Page

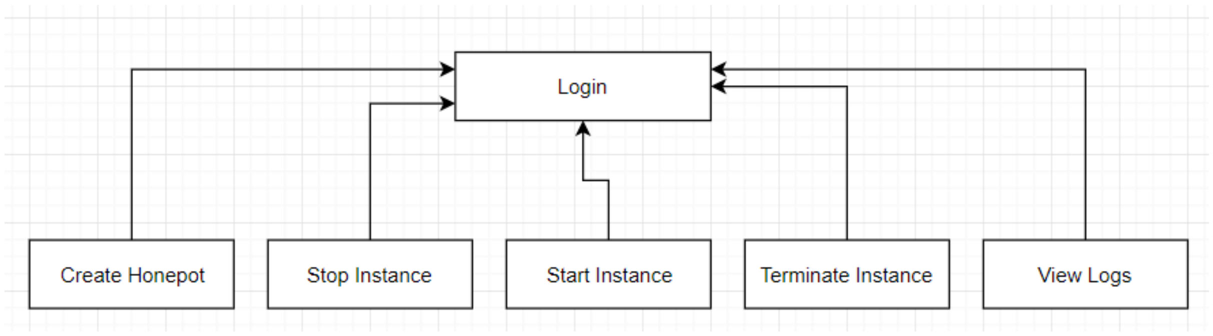


Figure 3.9 WireFrame of the Web Framework's Sidebar for all Web pages except the Home (no sidebar) and Settings (custom sidebar) page

3.12.6 Technical Specifications

The frameworks design will allow a user to log in and have the ability to create honeypots, manage honeypots as well as read logging information. The user will have the ability to create honeypots through the use of WTF-Flask forms.

3.12.7 Non-Functional Requirements

This web framework is designed to be used on a computer and hasn't been tested on a tablet or mobile device due to the fact that it is a localhost web framework but taking into consideration that the web framework is using Bootstrap, screen sizing is likely to be re-adjusted automatically. This web framework is aimed at all types of users who want to create a honeypot of IoT devices.

Chapter 4: Implementation

This chapter focuses on the implementation of The IOT Honeynet Framework and covers the implementation of the design features discussed in chapter three. The technologies used to overcome the issues presented in chapter three are explored in depth and changes made to the systems design are outlined.

The IOT Honeynet Framework takes a holistic approach by taking the desired features which include scanning a device specified by the user, then making a clone of that device and finally deploying it as a honeypot. There are a lot of steps and elements at work to make these features operate correctly as intended. This chapter describes the steps and processes taken to achieve the desired features outlined above and follows the methodology and thought process used.

4.1 Scanning the IOT device

The first step of the process in cloning an IOT device is to scan a target IOT device to establish what return values are produced and should be emulated in some manner to reproduce the same values. Many open source scanning tools are readily available and produce favourable results that aid the project but the tool that is used must be readily available and simple to install on the Linux operating system, produce scan results in a format that is easily greppable and be command line based so the task can be automated. Many such tools are available for most Linux distributions including Netcat, Angry IP Scanner, Unicornscan and Nmap (Network Mapper).

It was decided to use Nmap based on its reputation and wide spread use in the industry as a scanning tool. Nmap provides versatility required to extract the information from the target device. The parameters and switches can be specified to ensure that the necessary banner messages and ports are captured into a type of persistent storage for later use with an emulation script. Other criteria, such as speed, ease-of-use, reliability and cross platform usability were factors that Nmap met.

The Nmap file output format has many options, such as XML (Extensible Markup Language), greppable output and normal output. As this project requires the output be greppable, this option proved favourable to standardise the output in a reliable format each time the scan is run. This format must be reliable as other scripts later will depend on being able to search the scan results to complete other tasks.

Dedicated scripts inside Nmap that can be used to refine the scan results and provide focused results, proved promising in being able to accomplish the tasks required. One such script was particularly notable, a banner grab script, specifically designed to grab all the banners from a target device. This script was easy to implement and test.

$ nmap -sV --script=banner <target> .

The script connects to an open port and returns anything that is broadcasted by the service within the first five seconds. The first five seconds is the default time and can be changed if needed. The script prints the first line by default but can print additional lines if increased verbosity is chosen [68].

The end result of using Nmap was a simple command that returned all open ports with the associated banner messages where available, with fast, efficient speeds and reliable output formats.

$Nmap -sV -T5 -p- ip > file.txt

-sV – Service/Version Check.

-T5 – Timing 1-5.

-p- - Scan all ports.

Ip – IP of target device.

This command scans a provided target IP address using the version check switch, along with the all ports switch (-p-) and outputs the information to a text file for later use. This file can later be grepped for valuable information needed to reproduce the services running on these ports.

The banner script for Nmap was considered due to its focused role on grabbing banners, however, the service version switch proved more efficient as it was able to provide both the port numbers and the banners for the associated ports. This meant not having an added switch in the Nmap scan, making the script faster since the banner script takes around five seconds to run per port. The complexity of the script is also reduced as the banners would have to be associated with the ports detected, some of which may not have banners at all. Assigning the correct banner to each port would cause unnecessary complexity and could lead to errors with ports being assigned the wrong banner. It was for this reason that the Nmap banner script was avoided in favour of the service version switch.

4.2 Creating the Clone Script

The purpose of the clone script is to harvest specific information that will be used later to emulate the scanned device. This involved incorporating Linux standard bash utilities in the cloning script, such as Cat, Awk, Pipe and Grep. The commands are described briefly below to identify their function and better highlight the context in which they add to the overall script.

Cat (Concatenate) is a command line utility that reads files sequentially and writes it to standard output, i.e. console [69].

Grep (Global regular expression print) is a command line utility for searching text to match a particular character pattern. The pattern is also known as a regular expression [70]. This can be used to search for patterns in the scan output that are known to exist, which can then be extracted.

Awk (Aho, Weinberger, and Kernighan) is a scripting language used to manipulate data. This tool can split data into columns to aid in data extraction and is used to extract the data that is searched using the grep command [71].

Linux simplifies the redirection of output from a command to a text file using the greater than character (>). This character used after a command redirects the output into a file name that follows the character and is used to control the direction and location in which the data extracted will be outputted to.

Pipe is a Linux/Unix form of data redirection. With the pipe character (|) placed after a command, the output is passed to the next command that follows [72]. This is used to control the flow of the overall execution of commands, which is essential as the information from the scan file must first be outputted before it can be searched for the desired patterns using the grep command, then extracted using the Awk scripting language, before finally being redirected to the designated location for later use. This can be demonstrated as seen below.

Using Cat to read the file to standard output, the data can be piped to another command that performs another operation on the output. To achieve the desired overall output, all three commands were used to manipulate the output to get the required information.

```
$cat file | grep ‘open’ |  awk -F '[/]' '{print $1}' > ports_file
```

This command pipes the output from each command to the next command until the final result is outputted to another file for later use. This particular command pulls all the open port numbers from the input file and stores them in the output file. In a similar fashion, the banner messages are extracted from the same input file and redistributed to an output file after additional reformatting. The original file is never changed.

```
$cat file | grep ‘open’ |  awk '{$1=$2=$3="";print $0}'  | awk '{$1=$1};1'  > banner_file
```

With this task completed, another command was formulated to extract the protocols of each port into a separate file.

```
$cat file | grep "open" | awk -F '/' '{print $2}' | awk '{print $1}' > protocol_file
```

Along with these commands, the sequence in which the Nmap command executes is important and should be placed at the beginning of the script, thus the order in which the script executes will begin with the Nmap command and then followed by the commands discussed in this section in the order seen above.

After these commands were finalized, additional functionality is required to improve the overall functionality of the script. The ability to pass parameters to the script adds the ability to change the naming of files and directories.

The script is assigned two positional parameters. One for IP addressing and another for file naming. This will incorporate the ability to pass IP addresses and file names directly into the script. The user can now name the cloned devices themselves and remove the need for generic naming conventions. Inside the script, variables $1 and $2 receive the parameters being passed and allow the script to input these values wherever they are required. The Nmap command will use the first parameter and the second is passed to other commands for creating directories.

To make the script more usable on different systems, the script needs to know where the files being created are destined for in respect to the directory to where the script is being called from. A script stored in directory 'B' but called from directory 'A' will execute from directory 'A', the calling directory. Therefore, the absolute path or full path must be used to specify the directory the script should create its files in. Solving this issue involved using a Linux/Unix command called PWD (Print Working Directory). This command prints the full path to the current directory to which the command was executed in. As with many commands, the return from this command can be stored in a variable and used wherever the path must be specified. To capture the path in a variable, the following command is used: path=$(pwd). This stores the path in a variable named path and can now be passed to other commands.

With the path for each directory solved, additional checks must also be implemented for the overall functionality of the script to handler any errors that may arise. Through testing it was discovered that if a directory with the same name is already created, an error is thrown when an attempt is made to recreate the directory. A workaround for this, is to check if a directory name is present in the current directory before trying to create a new directory with the same name. If this directory does exist, remove it and recreate the new directory. After the scan has complete, an additional check for the presence of the main scan file is also important. This ensures the file is available before proceeding to create any further directories or files.

When the checks have passed, and the text files are created from the main scan file, the next process is to execute the Python scripts to create the remaining files required to build the Docker image. Two Python scripts need execution, one to create the Dockerfile and the other to create the Bash script that installs Docker and runs the image on the remote host.

Lastly, the files need to be moved into the newly created directory before copying the folder to the remote host. After all necessary files are copied into the directory, the main scan file is removed from the system to avoid any issues later.

These commands were placed into a file and renamed clone.sh. This will allow this sequence of commands to be run regularly without retyping the entire commands.

$bash clone.sh <ip>

```
#!/usr/bin/env bash

 ip=$1
 clone_name=$2

# absolute file path
path=$(pwd)
# directory name
dir="scripts"
# file that is created every time a scan is run
file="scan.txt"

#output file names
ports="ports.txt"
banners="banners.txt"
protocols="protocols.txt"

#scan ip address using nmap
#nmap -sV -p- $ip > $path/$dir/$file
nmap -sV -T5 -p- $ip > $path/$dir/$file

#check if file was populated
if [ -s $path/$dir/$file ]; then
    #grep files for ports and banners
    cat $path/$dir/$file | grep "open" | awk -F '[/]' '{print $1}' > $path/$dir/$ports
    cat $path/$dir/$file | grep "open" | awk '{$1=$2=$3="";print $0}' | awk '{$1=$1};1' > $path/$dir/$banners
    cat $path/$dir/$file | grep "open" | awk -F '/' '{print $2}' | awk '{print $1}' > $path/$dir/$protocols
    #check if directory name already exists
    if [ -d $path/$dir/$clone_name ]; then
        rm -rf $path/$dir/$clone_name;
    fi;
    mkdir $path/$dir/$clone_name;

    # create the docker file
    python3 $path/$dir/create_dockerfile_script.py $path/$dir/
    #create docker install script
    python3 $path/$dir/create_docker_build_script.py $clone_name $path/$dir/

    # copy all the necesasary files into folder
    cp $path/$dir/$ports $path/$dir/server_script.py $path/$dir/$clone_name
    mv $path/$dir/$banners $path/$dir/Dockerfile $path/$dir/install_run_docker_container.sh $path/$dir/$clone_name
    # rm $path/$dir/$file
else
    echo "   # # # #        Problem scanning device             # # # #"
fi;
```

Figure 4.1 Clone.sh Script

4.3 Emulating services

The emulation of services is an integral element of any honeypot which can be accomplished in a variety of ways. The methods explored in emulating the services for the IOT Honeynet Frameworks honeypots are elaborated below and outlines the methodology being used for each approach.

4.3.1 High Interaction method

The first method considered in emulating an IOT device was to establish a highly interactive IOT honeypot by using a Linux distribution as a base image. This image would then have the exact versions of software installed that were recovered from the scan. The software would be installed and opened on the same port as the device to replicate the services provided. In theory, this would be an excellent method of capturing activity that the machine may have with hackers or worms attempting to access restricted content. However, in practice many issues arose while researching this method such as how to close ports that the OS is using for internal services, how to open the port numbers required including the restricted port numbers below 1024. Other security concerns regarding the level of controlled interaction from a possible hacker and hardening the system against possible attackers proved difficult if using older versions of software that may be running on IOT devices.

Establishing the services running on scanned devices proved to be another issue with this method. It was found that many versions on scanned devices were outdated with multiple exploits available. For example, an IOT device was scanned using Nmap with version detection to retrieve the versions running on each port. This IOT device was running an outdated version of Apache web server. The version running is 2.2.6. When this version of Apache is typed into Google, the results returned security vulnerabilities for this version and previous versions. Another issue with this approach was when examining the scan results, the version of the service could not be determined in many occasions. This made determining the appropriate version impossible and therefore impossible to reproduce. Installing the most current version would remove possible security risks that would otherwise attract attention.

The biggest issue and concern regarding this approach is the security of the OS when attracting attention from possible attackers. If access was gained to a restricted part of the OS, unknown damage may occur.

4.3.2 Bash Method

With a working script that scans a given IP address and returns a file which can provide the required data. The next step is a method of emulating an open port and returning the banner message relevant to the port number.

After researching ways to open a port on a Linux based machine, Netcat was a tool that was being regularly mentioned. This tool can read and write to network connections using TCP or UDP protocols [73]. Included with this tool is the ability to perform port scanning and Netcat also provides a tunnelling mode.

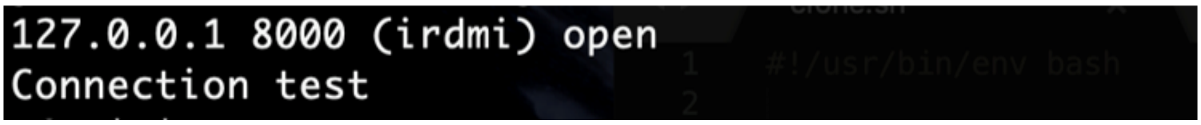
Using the Netcat man page to search for instructions on how to open a specific port revealed that Netcat is unable to send a particular piece of text itself. A work around is to "echo" the text and pipe the output into Netcat. To test the "echo" command piping the text to a Netcat command, a test is conducted locally. On a local machine, this command is displayed below.

$echo "Connection test" | nc -l -p port_number

Another terminal is open to connect to the specified port, again using Netcat.

$nc 127.0.0.1 port_number

The results show that this does return the message as intended.



```
127.0.0.1 8000 (irdmi) open
Connection test
```

Figure 4.2 Netcat fails to return correct message.

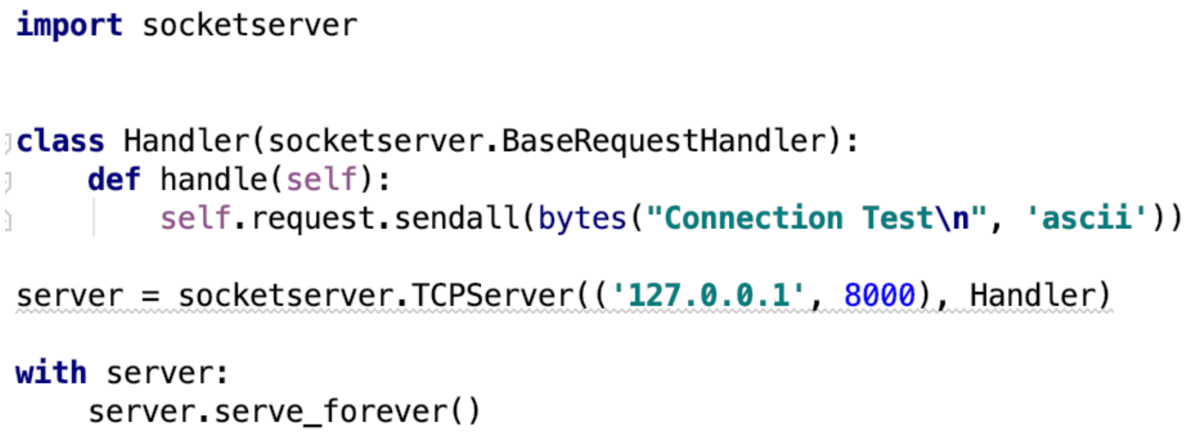
To achieve this on a larger scale will require further scripting to ensure the message is returned to each connection request. The only method that meets this requirement is using an infinite while loop to continuously return the message to each connection. Bash made this easy with a one-line solution.

$ while True; do echo "Connection test" | nc -l -p 8000; done;

This proved effective for a single port and banner but with an undetermined number of ports and messages to return, the script would need to be more dynamic. Each device being cloned can have varying ports open and some of these ports might not provide a banner to return at all. Bash does provide a type of solution which allows each command to be run in a separate process in the background. This can be achieved by using the "&" character between each command, much like the pipe character was used. Each command would then have its own process to run until the process is terminated. This solution would have a high overhead on the system with an undefined number of processes possibly being started which could have detrimental effects on the host system.

4.3.3 Python Method

As an alternative approach, Python was highlighted as a possible overall solution. Python has many importable modules to further advance the usability of the language. One such module is the socketserver module. This module allows the simple creation of TCP or UDP network servers which can open ports and allow the reading and writing of data across the network. With enough research done on how the module works, the construction of a TCP network server began. Below a simple TCP server written in python 3 can be seen.



```python
import socketserver


class Handler(socketserver.BaseRequestHandler):
    def handle(self):
        self.request.sendall(bytes("Connection Test\n", 'ascii'))

server = socketserver.TCPServer(('127.0.0.1', 8000), Handler)

with server:
    server.serve_forever()
```

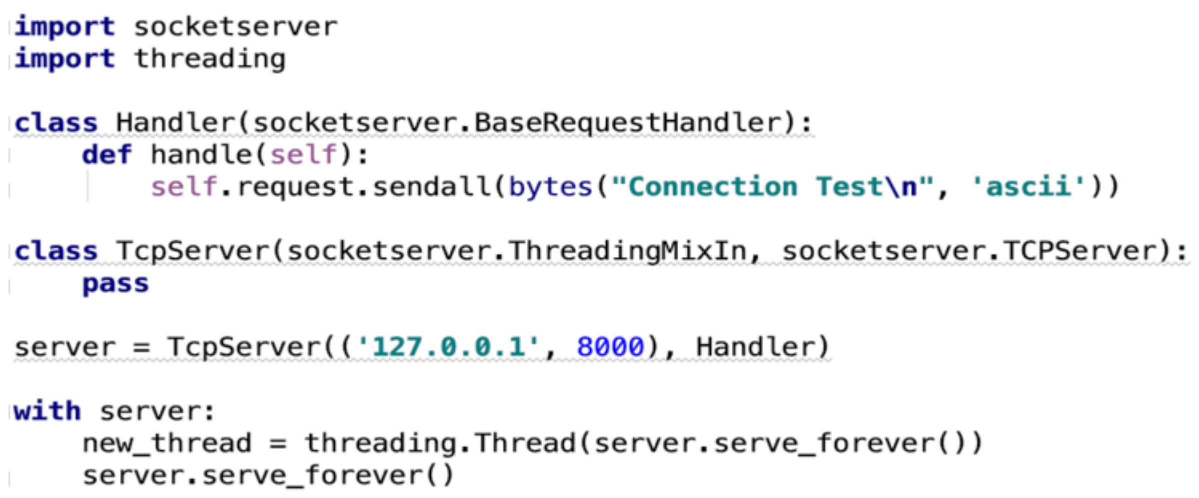
Figure 4.3 simple Python3 TCP server.

The first step was importing the Python module "socketserver". The next step is to create a class which instantiates the socketserver.BaseRequestHandler class. This creates an instance of the request class which is necessary to handle data being sent and received through the port being opened. This class contains a simple function which sends a "Connection Test" to any request on the port opened.

server = socketserver.TCPServer((‘127.0.0.1), 800), Handler)

This line instantiates the socketserver.TCPServer class and passes a tuple containing the IP address and the port number. The IP address passed is the loopback address or local address of the host machine. This is telling the server to use the host machine address. The port number is the port number to be opened. The last parameter is the handler class which will handle the data for each request on that port. The Python "with" statement is similar to a try catch block as in the "with" statement closes the server gracefully [74]. The last line server.serve_forever() tells the server to continue handling requests until an implicit shutdown command is given.

With this simple example working as expected, the code now needs to be scaled up to incorporate multiple connections. To achieve multiple simultaneous connections, Python threading must be used.



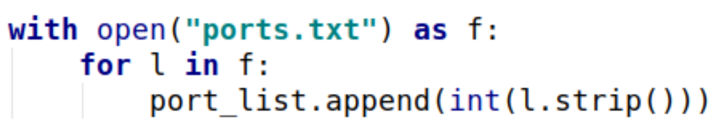
```python
import socketserver
import threading

class Handler(socketserver.BaseRequestHandler):
    def handle(self):
        self.request.sendall(bytes("Connection Test\n", 'ascii'))

class TcpServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass

server = TcpServer(('127.0.0.1', 8000), Handler)

with server:
    new_thread = threading.Thread(server.serve_forever())
    server.serve_forever()
```

Figure 4.4 Multithreaded Python TCP socket server.

This will create a separate thread of execution for each request made to the server. This required that the socketserver.TCPServer be implemented as a class in addition to socketserver.ThreadingMixIn. The socketserver.ThreadingMixIn class adds threading to the program and can be used to start a new thread for each additional request.

With threads implemented for one port, the program now needs to implement a separate thread for each port opened and additionally, a separate thread for each request made to each port. Each port opened should display a unique banner message where applicable and return no data when necessary. As this server will be emulating an IOT device on a remote server, text files will be used containing all the data needed by the server. This will ensure persistence when the server shuts down or restarts. The serve script must then read the data from a text file when starting up. This functionality must also be included in the script.

To accomplish reading in files, another function was added to the script to automate this every time the script was run. This function reads in both files, ports.txt and banners.txt, places the data into a dictionary using the port number as an index value and the banner message as the data. This will ease the retrieval of the banner message relevant to a port number later on. This can be seen in the below figure.



```python
with open("ports.txt") as f:
    for l in f:
        port_list.append(int(l.strip()))
```

Figure 4.5 Reading in a text file.

To achieve additional interaction with port 22 (Secure Shell) and port 21 (Telnet), a second handler function was constructed to specifically deal with these ports. SSH and Telnet are ports that have a higher level of interaction. Each port should have a username and password login prompt and allow attempted logins. The login attempts could be recorded in persistent storages foe later analysis.

```
self.request.sendall(bytes("Please login: \nUsername  ", 'ascii'))
user = str(self.request.recv(1024), 'ascii')
self.request.sendall(bytes("Password:  ", 'ascii'))
passwd = str(self.request.recv(1024), 'ascii')
response = bytes("Username {}\n or Password {}\n are invalid".format(user, passwd), 'ascii')
self.request.sendall(response)
```

Figure 4.6 Basic login prompt.

4.4 Clone Deployment

This section covers the process of deploying a clone device as a honeypot.

4.4.1 Building and running the Docker Image

With the Python server script constructed and threading implemented into the script for each port and connection, a deployment platform must be decided upon. The dependencies of the script are Python and two text files consisting of port numbers and banner messages.

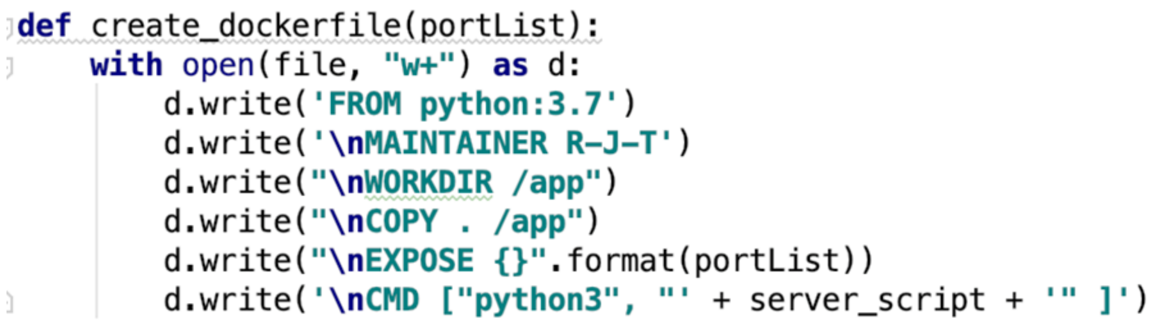
Docker was chosen for this project due to its cross-platform robustness and fast deployment design. This ensures the containers built, will run no matter the OS on the destined machine and remove any OS dependent issues. This would make the deployment uniform for each device and streamline the process when completed.

The process of building and deploying a Docker container consists of the creation of a Dockerfile, building the image using the Dockerfile and using the Docker run command to run the image to become a container.

The Dockerfile contains a list of dependencies, a list of ports that should be exposed during the build and any additional commands to execute when the image is run. In the case of this project, the dependencies are Python 3.7 and the files to be included are the server script and both text files. The ports to be exposed are the ports listed inside the ports.txt file and the only command to be executed is to run the Python script.

An issue did arise when trying to open the ports inside the text file. Docker has no commands that can automate the exposure of ports from a file. Attempts were made to accomplish this with Bash, but this proved cumbersome. Due to Pythons simple file writing manner, it was decided to use Python to create a script to write the entire Dockerfile each time, reading it the port numbers and formatting them as Docker expects. This script proved highly dynamic and allowed easy editing and control over the Dockerfile.

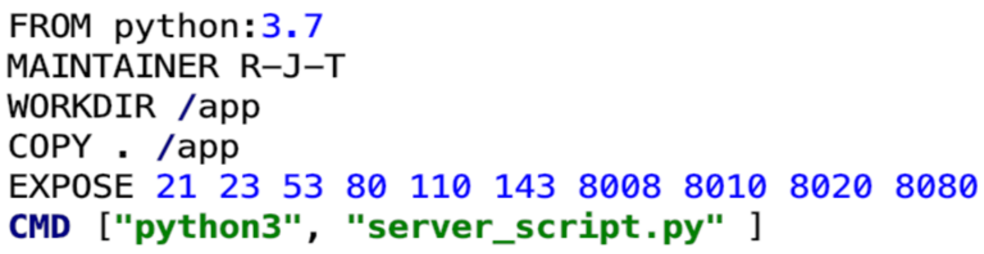
The code required to create the Dockerfile was developed and tested. This code snippet is the function that writes the Dockerfile. The parameter “portlist” is passed as a return variable from another file read function, that returns a preformatted string consisting of all the port numbers.



```
def create_dockerfile(portList):
    with open(file, "w+") as d:
        d.write('FROM python:3.7')
        d.write('\nMAINTAINER R-J-T')
        d.write("\nWORKDIR /app")
        d.write("\nCOPY . /app")
        d.write("\nEXPOSE {}".format(portList))
        d.write('\nCMD ["python3", "' + server_script + '" ]')
```

Figure 4.7 Dockerfile creation function

The Dockerfile created as a result from running this script is as shown.



```
FROM python:3.7
MAINTAINER R-J-T
WORKDIR /app
COPY . /app
EXPOSE 21 23 53 80 110 143 8008 8010 8020 8080
CMD ["python3", "server_script.py" ]
```

Figure 4.8 Dockerfile completed

With the Dockerfile creation now complete, the image was ready to be built. Using the Docker command “docker build -t image_name .”, the image is constructed using the Dockerfile and all the files in the current directory. The current directory is symbolised by the dot notation at the end of the command.

With the Docker image now built, it’s time to run the image. At this stage, the port number must again be used to map each port of the container to a port on the host operating system (OS). The syntax is as follows -p 23:23. This maps the containers port 23 to port 23 on the host OS. This must be done for each additional port that is to be exposed publicly to the outside network. As with the previous difficulty with numerous port numbers being dynamically exposed, this will again be solved with another Python script to auto generate the command for each container created.

As this script is intended to run on a remote host, checks must also be put in place to ensure that Docker is installed on the intended host machine. To achieve this, this script must also have a check to discover which Linux distribution is currently in use to determine which package manager is to be used to install the Docker software. Many Linux distributions exist and each use varying package managers to install software from a reliable source. A package manager is a collection of software tools that automate the installation process of software. The package manager name is required to construct the command for which to install the software. Following the checks and installing the Docker software, the Docker service must then be started before any Docker command can be used. The Docker build command must then be issued to build the image before the final Docker run command can be executed which deploys the container. The Docker run command must include the mapping of all ports in the expected format -p port:port -p port:port etc. Similarly, to the previous script, this one included a function to read the ports.txt file and construct a preformatted string using the port numbers from the file. This script simplifies the process of building and running the Docker image on the remote server.

The end result from executing this Python script results in a uniquely formatted bash script which firstly, checks which OS the script is being executed on, then using the appropriate command syntax, updates the system to the current versions. Secondly, it installs the Docker software using a suitable package manager. When the software has successfully installed, the Docker daemon is started. Finally, the image in the current directory is built using the previously generated Dockerfile and associated text files. When this command completes, the image executed into a running container, emulating the services passed from within the text files.

4.4.2 Transferring files

With testing completed locally on the Docker image, the next step is to establish a method of securely transferring the files to a given host and executing a series of commands. This will involve two steps. Step 1 transfer the files to the remote host. Step 2 execute a series of commands to complete the objective.

When secure file transfer is required, one of the best and most secure methods is SCP (Secure Copy). SCP uses SSH (Secure Shell) to securely transfer files between hosts without the need for an FTP (File Transfer Protocol) sessions or having to log in manually. SCP uses either an SSH key or a pass phrase linked to the account or the user must be logged into the account. With logging in and pass phrases not practical, an SSH key will be used.

SSH keys are an asymmetric key pair used to SSH into a remote server and are primarily used for authentication. SSH provides secure terminal access to a remote host. The key consists of two keys, a public key, which will be stored on the remote host, and a private key, which is stored on the local machine and never shared. The private key is used while connecting to the remote host to authenticate the user when logging in without the need for a username and password.

This is the basic syntax to secure copy files from a local machine to a remote host:

```
$ scp path/to/files user_name@ip_address:/path/on/remote/host
```

The syntax is straight forward. The command, followed by the files to be transferred, followed by the user name and IP of the remote host and the path to where the files should be transferred to. Additional syntax is required to use the SSH key rather than being prompted to login, which will not be possible later when the scripts are implemented with some type of GUI (Graphical User Interface) to simplify the process further.

To use a SSH key with the SCP command, the -I switch must be added, followed by the path to the SSH key itself. After numerous tests, another issue arose from using this command. When the command is executed on a new remote host, a prompt is displayed, asking if the host should be added to the SSH known_hosts file. This file holds a list of all the known hosts the local machine has connected with. This will only be an issue when automating the process as confirmation will not be possible.

Viewing the SCP man (Manual) page, an option to specify SSH commands inside the SCP command, allows the passing of a parameter directly to SSH. A parameter called ‘StrictHostKeyChecking’ which asks the user if connecting to an unauthenticated remote host, ‘Are you sure you want to continue connecting (yes/no)?’. This feature can be turned off in the /etc/ssh/ssh_config file by changing the ‘StrictHostKeyChecking ask’ to ‘StrictHostKeyChecking no’. This will turn off this feature completely for all new SSH connections. A more simplistic method is to turn it off for specific commands, or per command. This can be achieved by using the -o switch followed by ‘StrictHostKeyChecking=no’. The completed command to now copy files to the host is:

```
$scp -o StrictHostKeyChecking=no -i path/to/ssh-key -r path/to/files user@remote_IP:~/remote/path
```

The addition of the -r switch is to copy all files recursively inside that directory.

4.4.3 executing the scripts

Now that the files have been successfully transferred to the remote host, the scripts need to be executed in succession. This was quickly achieved using an SSH login command followed by the commands that needed to be executed.

With a precompiled bash script included in the file transfer, with all the necessary update and build commands, it is now only a matter of executing the script. Similar to the file transfer command, the SSH command is:

$ssh -o StrictHostKeyChecking=no -I /path/to/ssh-key username@remote_IP ‘sudo bash directory/script_name’

This successfully executes the script which checks the package manager to use, updates all the packages on the system, installs the Docker package and all the dependencies. When the installation has completed, the Docker daemon is started. The next commands executed are the Docker build command followed by the Docker run command. The clone is now running on a remote host.

4.5 Deploying A Honeypot to the Honeynet

The process of creating a clone of a device and setting up the honeypot have been discussed. Now the honeypot needs to be deployed as part of a honeynet exposed to the internet. As discussed earlier in the Design and Methodology chapter, it was decided to use Amazon Web Services (AWS) as the hosting provider.

Attaching a honeypot through the IOT Honeynet Framework to the honeynet requires a programmatic approach. This requires the framework to be able to securely connect to a user’s AWS account and deploy an EC2 instance using the user’s account with all the specified requirements requested by the user. In the following sections, focus will be placed on the code used which incorporates the boto3 library to handle the requested actions from the user through the framework. The code is designed to work in tandem with the web framework which uses forms to pass information. As such the use of forms will be referenced but the appearance, functionality and use of the forms will be discussed later in this chapter.

4.5.1 Requirements and Setup

This section focuses on the initial environment setup and configuration required for using Amazon Web Services. Section 4.5.1.1 describes the initial setup a user must perform before the framework can operate correctly. Section 4.5.1.2 covers the creation of a keypair that is used when creating the instance. Section 4.5.1.3 covers the creation of the log group where the logs collected from honeypots are stored. Section 4.5.1.4 elaborates on the creation of Roles and Policies. The section 4.5.1.5 details how to create a flow log which is created on a VPC to direct where the logs go.

4.5.1.1 Setup IAM User and Environment

To allow the IOT Honeynet Framework to create and manage EC2 instances, the Boto3 library in Python and the AWS Command Line Interface (CLI) tool need to be installed. After this, it's necessary to setup an account and create an IAM user. This is important as the credentials for the user (access key and access id) are needed to allow the IOT Honeynet Framework to interact with a user's AWS account. These requirements should be more clearly elaborated to better understand their role in enabling the framework to achieve the desired features outlined earlier [75].

Adding a user involves a few simple steps. Once logged into an AWS account, launch the Identity and Access Management console. Click *"Users"* on the navigation menu on the left of the screen as seen in the below figure.

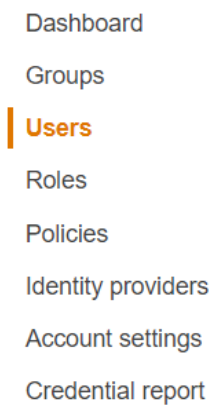


Figure 4.9 IAM Navigation Menu.

Next, in the popup window choose *"Add User"*.

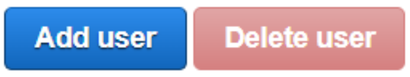
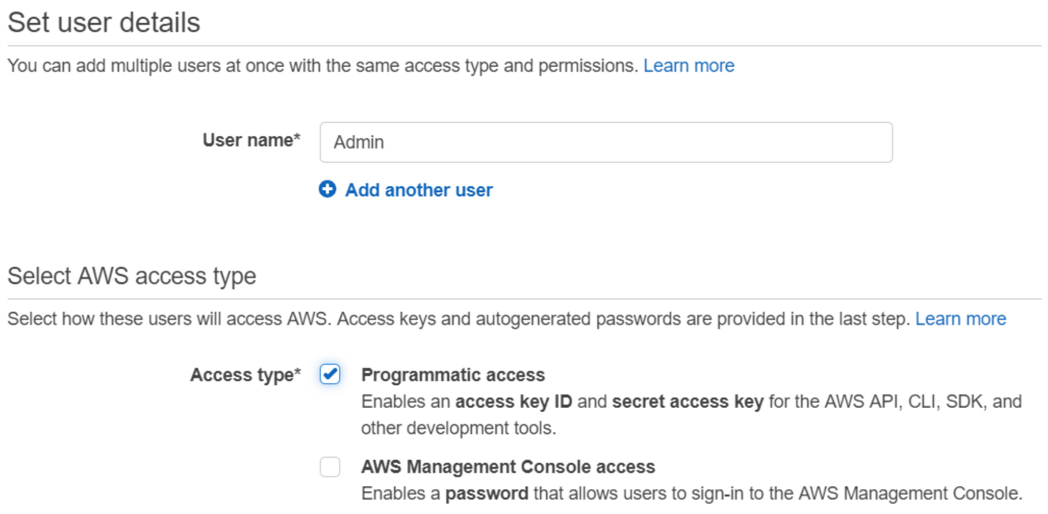


Figure 4.10 IAM Add User.

This will present a new window that requests a user name and the type of access this user should have. The type of access this user is allowed is important for when a user is working through the IOT Honeynet Framework. Programmatic access is essential to using the AWS CLI, a core feature of the IOT Honeynet Framework. For the purposes of this project it is necessary for the user with administrator privileges to have programmatic access, which is the option selected as can be seen in the figure below.



Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* Admin

Add another user

Select AWS access type

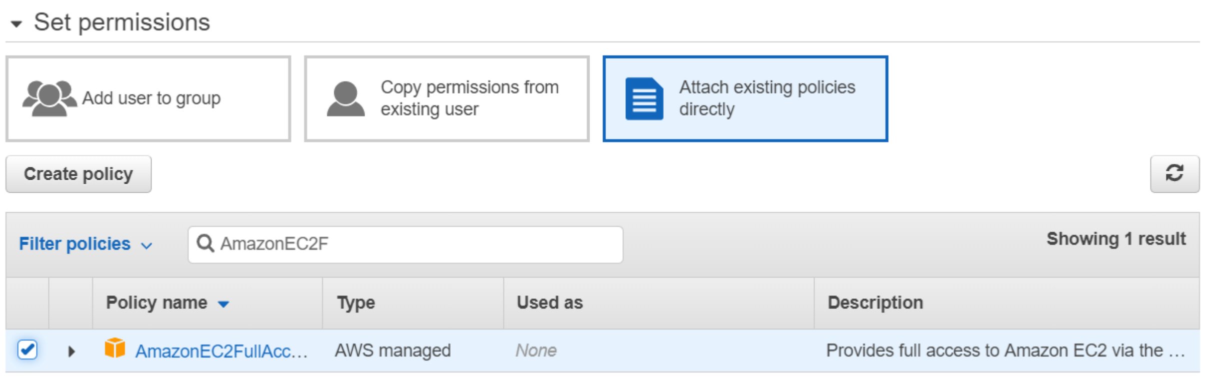
Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* [✓] **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

[ ] **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

Figure 4.11 IAM Username and Access Type.

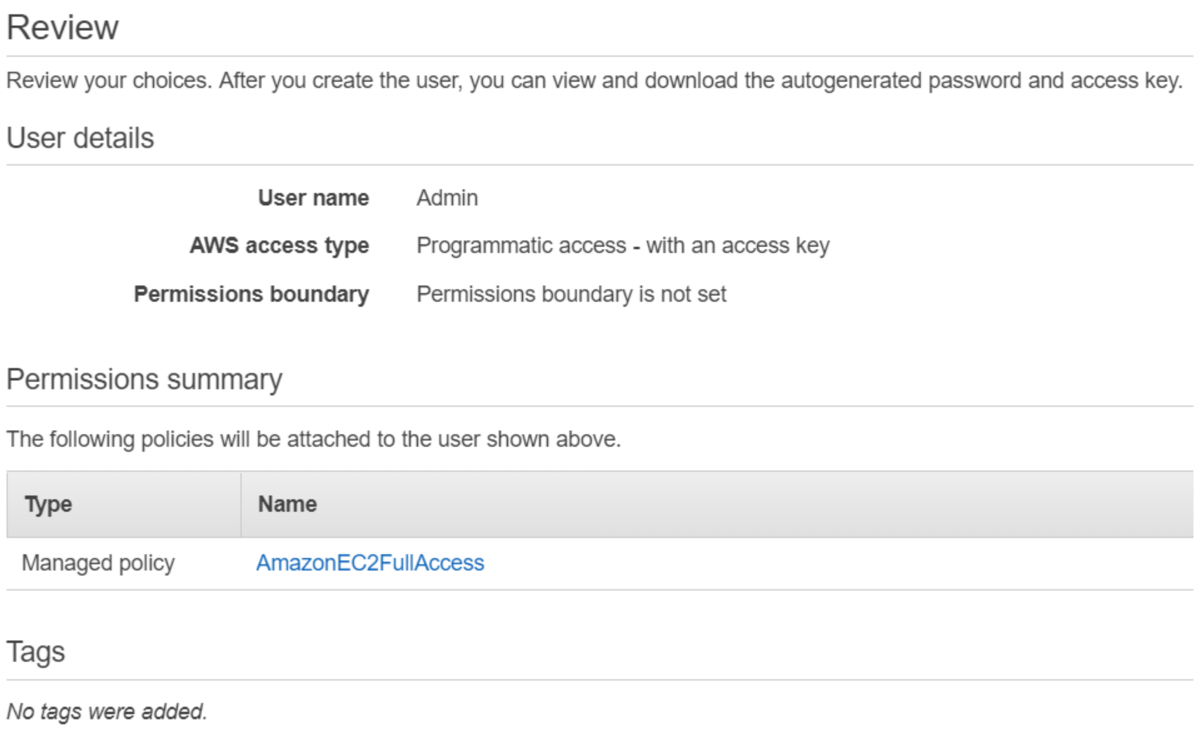
The next step is to determine what permissions the user should have. It is possible to create different groups to divide users based on what access privileges they are permitted to have. For a user to be able to fully interact with their instances and manage them, the IAM user they are setting up must have administrator level privileges. The policy titled *"AmazonEC2FullAccess"* was selected and attached to the user which can be seen in the figure below.



		Policy name	Type	Used as	Description
✓	▸	AmazonEC2FullAcc...	AWS managed	None	Provides full access to Amazon EC2 via the ...

Figure 4.12 IAM user permissions.

Finally, the optional choice to add tags to the user is presented which can be used to assign metadata to help track, organize and control access for the user [23]. After this, the user details and permission levels can be reviewed before finally creating the user.



Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	Admin
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

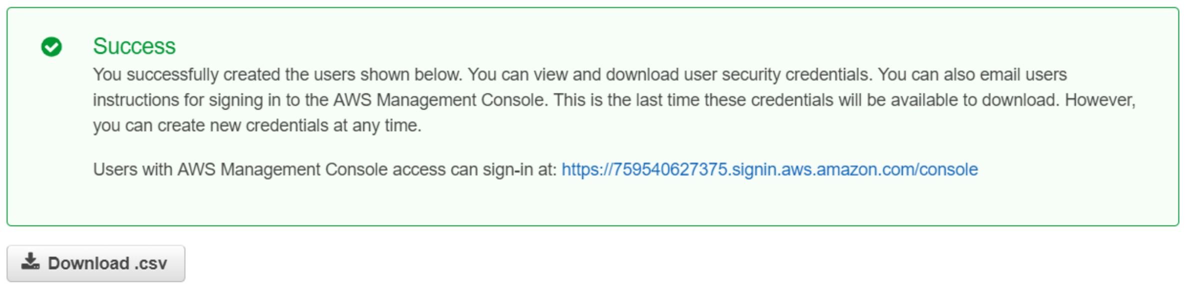
Type	Name
Managed policy	AmazonEC2FullAccess

Tags

No tags were added.

Figure 4.13 Review IAM User Details.

When the user is created, a window is presented which shows the user's access key id and access secret key. These are only available once, so it's important to download and save them safely in a secure location.



Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: https://759540627375.signin.aws.amazon.com/console

Figure 4.14 IAM User Created Successfully.

Now that an IAM user is created, it is possible to programmatically access Amazon EC2 service, which is required to permit the IOT Honeynet Framework to remotely access and manage EC2 services.

The Boto3 library in Python is, according to the boto3 documentation, “the Amazon Web Services (AWS) SDK for Python” [76]. It allows developers to create, configure and manage amazon services such as EC2 by providing an easy to use, object-oriented API, as well as low-level access to AWS services [76]. This library is used extensively throughout the IOT Honeynet Framework to provide many of the essential functions such as displaying logging information and creating new EC2 instances. The entirety of boto3’s contribution will be outlined in the coming sections, but it is important to note it’s integral part in the project.

Amazon outlines the AWS CLI as “an open source tool that enables you to interact with AWS services using commands in your command-line shell” [77]. Once a user with the necessary credentials (Access Key, Access ID) has been created, it is possible to configure the Python scripting environment with these credentials in order for the IOT Honeynet Framework to manage EC2 instances remotely. The “*aws configure*” command is used to set the credentials and other configuration details which include the region and output format. Once those are provided, credentials are saved in a local file at path “*~/.aws/credentials*” and other configurations for region and output format are stored in “*~/.aws/config*” [75].

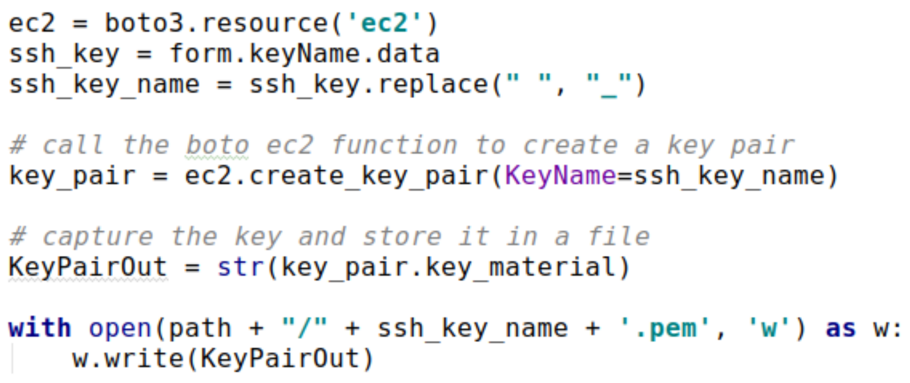
In order for the IOT Honeynet Framework to operate correctly, the user must have an AWS account with the associated credentials setup. The python boto3 library and the AWS CLI must be installed for the IOT Honeynet Framework to be able to manage Amazon EC2 instances. Once the python environment is setup with the users AWS credentials, the IOT Honeynet Framework can operate as designed, with all intended features.

4.5.1.2 Creating a key pair

According to the boto3 documentation the *create_key_pair()* is used to create a 2048-bit RSA key pair with the specified name [78]. Amazon EC2 stores the public key and displays the private key for a user to save to a file. The script used to achieve this takes in the name of the key pair provided by the user. The key is then stripped of spaces and replacing them with underscores as well as all letters being changed to lower case if not already so. A call is made using the boto3 session object to make a key pair which is then captured and written to a file with the name provided by the user. This new ssh key is then stored in the *ssh-key* directory.

An important final step to take when the key pair is created is to change the mode of the key pair file to read-only otherwise it will be denied access. For Linux this accomplished by simply using the below command:

```
chmod 400 <name of file with private key>
```

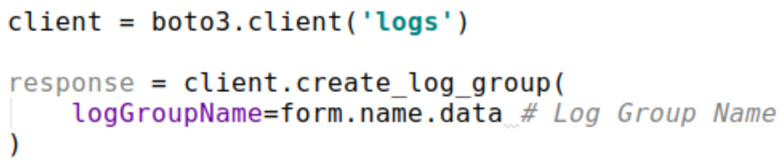


```
ec2 = boto3.resource('ec2')
ssh_key = form.keyName.data
ssh_key_name = ssh_key.replace(" ", "_")

# call the boto ec2 function to create a key pair
key_pair = ec2.create_key_pair(KeyName=ssh_key_name)

# capture the key and store it in a file
KeyPairOut = str(key_pair.key_material)

with open(path + "/" + ssh_key_name + '.pem', 'w') as w:
    w.write(KeyPairOut)
```

Figure 4.15 Script creates keypair.

4.5.1.3 Create a Log Group

The implementation of logging requires some initial setup. When collecting logs, it is necessary to specify a destination to send those logs. CloudWatch uses log groups which stores a separate log stream for each instance, containing the logs captured from that instance. A user creates a log group by providing a name for the log group. This name is passed to a script which then creates the log group using the *create_log_group()* function.

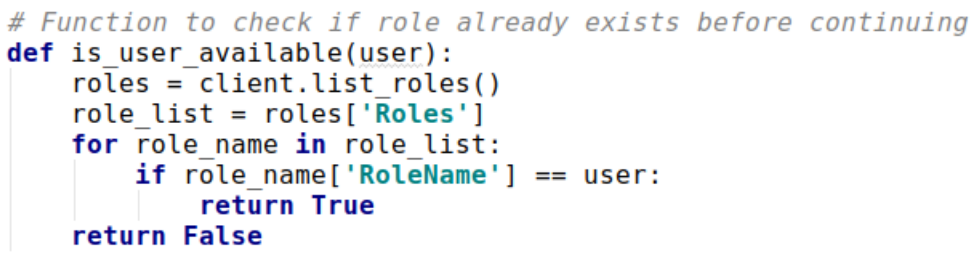


```
client = boto3.client('logs')

response = client.create_log_group(
    logGroupName=form.name.data # Log Group Name
)
```

Figure 4.16 Create Log Group.

4.5.1.4 Create a Role and IAM Policy

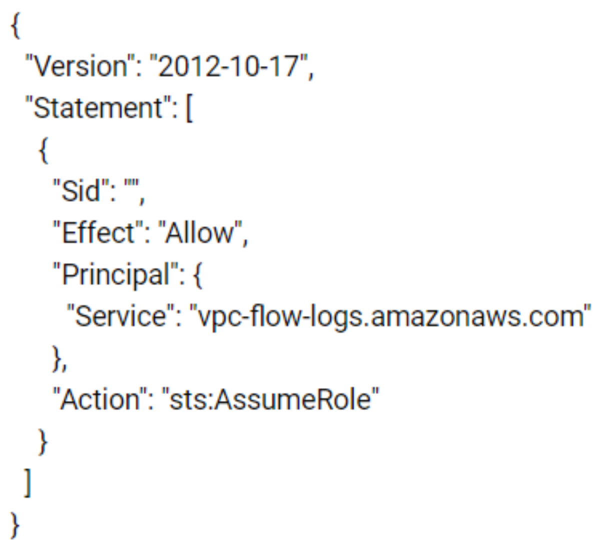
It is necessary to create an IAM policy to permit flow logs which will be created later to store the logs in CloudWatch. This policy will be assigned to a role which will be then attached to the specified IAM user. To create a new policy, the user starts by entering in a name for the new policy they wish to create. A function called *is_user_available()* is defined to check if the name of the new role entered by the user is already used.



```
# Function to check if role already exists before continuing
def is_user_available(user):
    roles = client.list_roles()
    role_list = roles['Roles']
    for role_name in role_list:
        if role_name['RoleName'] == user:
            return True
    return False
```

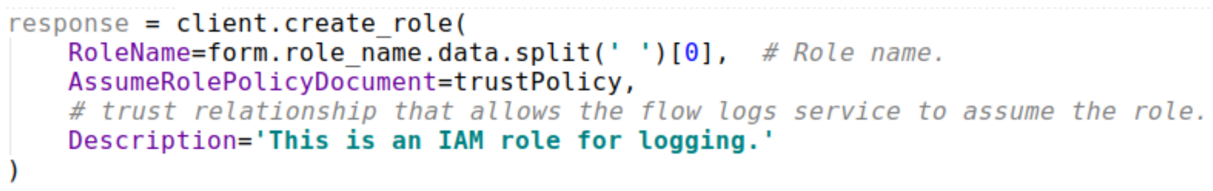
Figure 4.17 Checks if role name is occupied.

If not, a new role is created using the *create_role()* function and a preconfigured JSON trust relationship is attached to the role which allows the flow logs service to assume the role.



```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
    "Service": "vpc-flow-logs.amazonaws.com"
   },
   "Action": "sts:AssumeRole"
  }
 ]
}
```

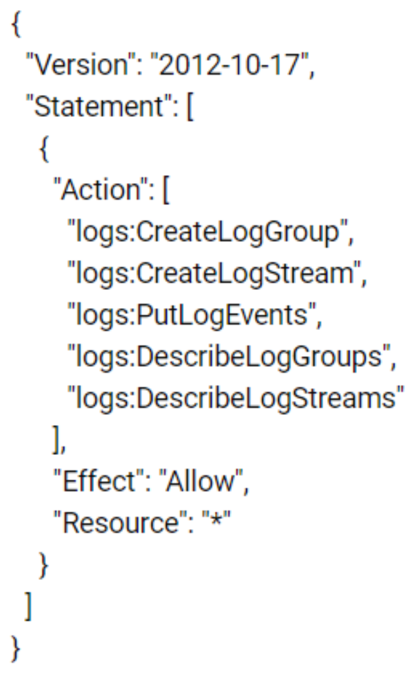
Figure 4.18 Trust Relationship.



```
response = client.create_role(
    RoleName=form.role_name.data.split(' ')[0],  # Role name.
    AssumeRolePolicyDocument=trustPolicy,
    # trust relationship that allows the flow logs service to assume the role.
    Description='This is an IAM role for logging.'
)
```

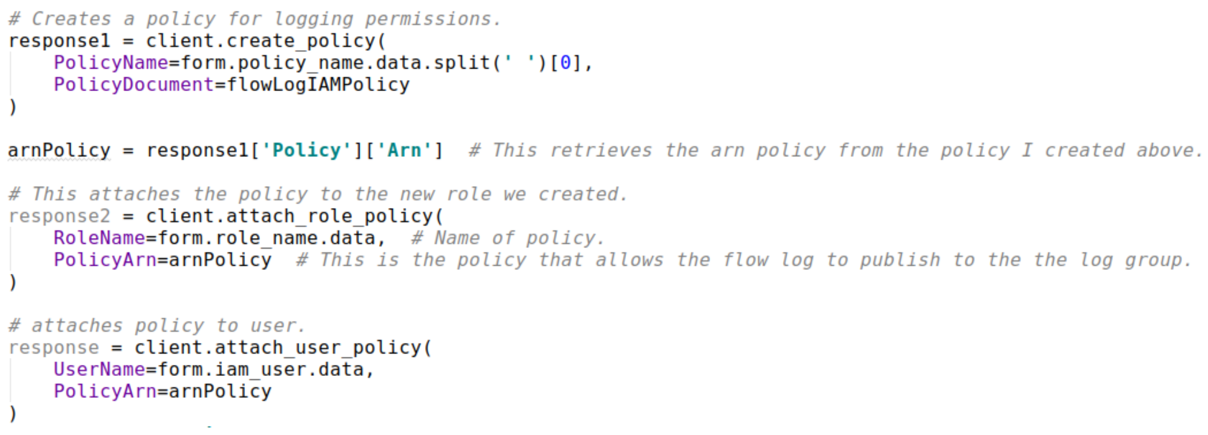
Figure 4.19 Create a Role

Next a policy is created using the *create_policy()* function which is assigned the name provided by the user and also uses a preconfigured JSON flow log IAM policy which is added to the policy and establishes the permission this policy possesses. The ARN of the policy is retrieved which is comparable to the policy's ID and is used to attach the policy to the new role that was created earlier using the *attach_role_policy()*. Finally, this role is attached to the IAM username provided the user using the *attach_user_policy()*.



```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents",
    "logs:DescribeLogGroups",
    "logs:DescribeLogStreams"
   ],
   "Effect": "Allow",
   "Resource": "*"
  }
 ]
}
```

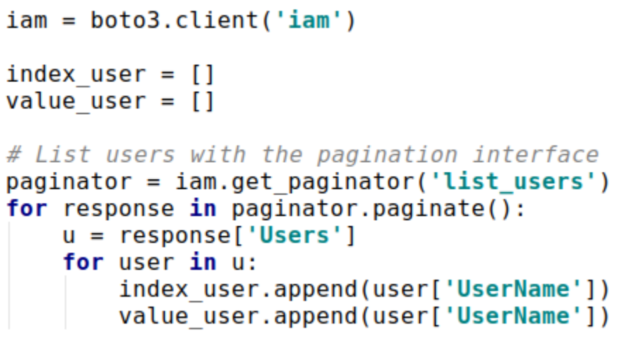
Figure 4.20 Flow Log IAM Policy.



```
# Creates a policy for logging permissions.
response1 = client.create_policy(
    PolicyName=form.policy_name.data.split(' ')[0],
    PolicyDocument=flowLogIAMPolicy
)

arnPolicy = response1['Policy']['Arn']  # This retrieves the arn policy from the policy I created above.

# This attaches the policy to the new role we created.
response2 = client.attach_role_policy(
    RoleName=form.role_name.data,  # Name of policy.
    PolicyArn=arnPolicy  # This is the policy that allows the flow log to publish to the the log group.
)

# attaches policy to user.
response = client.attach_user_policy(
    UserName=form.iam_user.data,
    PolicyArn=arnPolicy
)
```

Figure 4.21 Creates a policy, attaches policy to Role, then attaches role to IAM User.

The list of possible IAM usernames available for the user to select from is provided by using *get_paginator()* function which returns information about current IAM users available including the username.



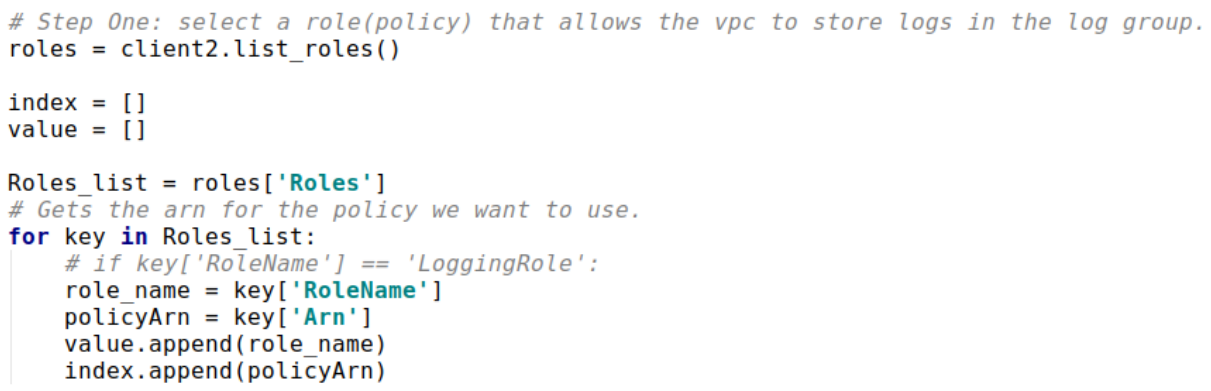
```
iam = boto3.client('iam')

index_user = []
value_user = []

# List users with the pagination interface
paginator = iam.get_paginator('list_users')
for response in paginator.paginate():
    u = response['Users']
    for user in u:
        index_user.append(user['UserName'])
        value_user.append(user['UserName'])
```

Figure 4.22 List IAM users.

4.5.1.5 Create a Flow Log

To create a flow log, an IAM Policy that has permissions to allow logging and a log group in CloudWatch must be created, hence why the previous two sections were covered first. The user supplies the IAM policy they wish to use, the name of the log group they wish their logs to be stored in and the VPC they wish to use for the flow log they are creating.

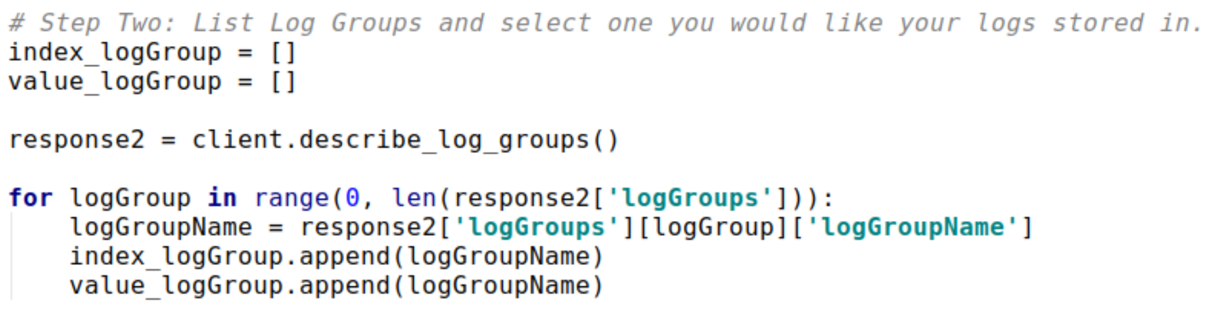
The *list_roles()* function was used to list all the current roles a user has created and allows them to choose which one that they desire, which in this case would be the role that has the policy that provides logging permissions. The role name and the policy ARN are retrieved for later use.



```
# Step One: select a role(policy) that allows the vpc to store logs in the log group.
roles = client2.list_roles()

index = []
value = []

Roles_list = roles['Roles']
# Gets the arn for the policy we want to use.
for key in Roles_list:
    # if key['RoleName'] == 'LoggingRole':
    role_name = key['RoleName']
    policyArn = key['Arn']
    value.append(role_name)
    index.append(policyArn)
```

Figure 4.23 List Roles.

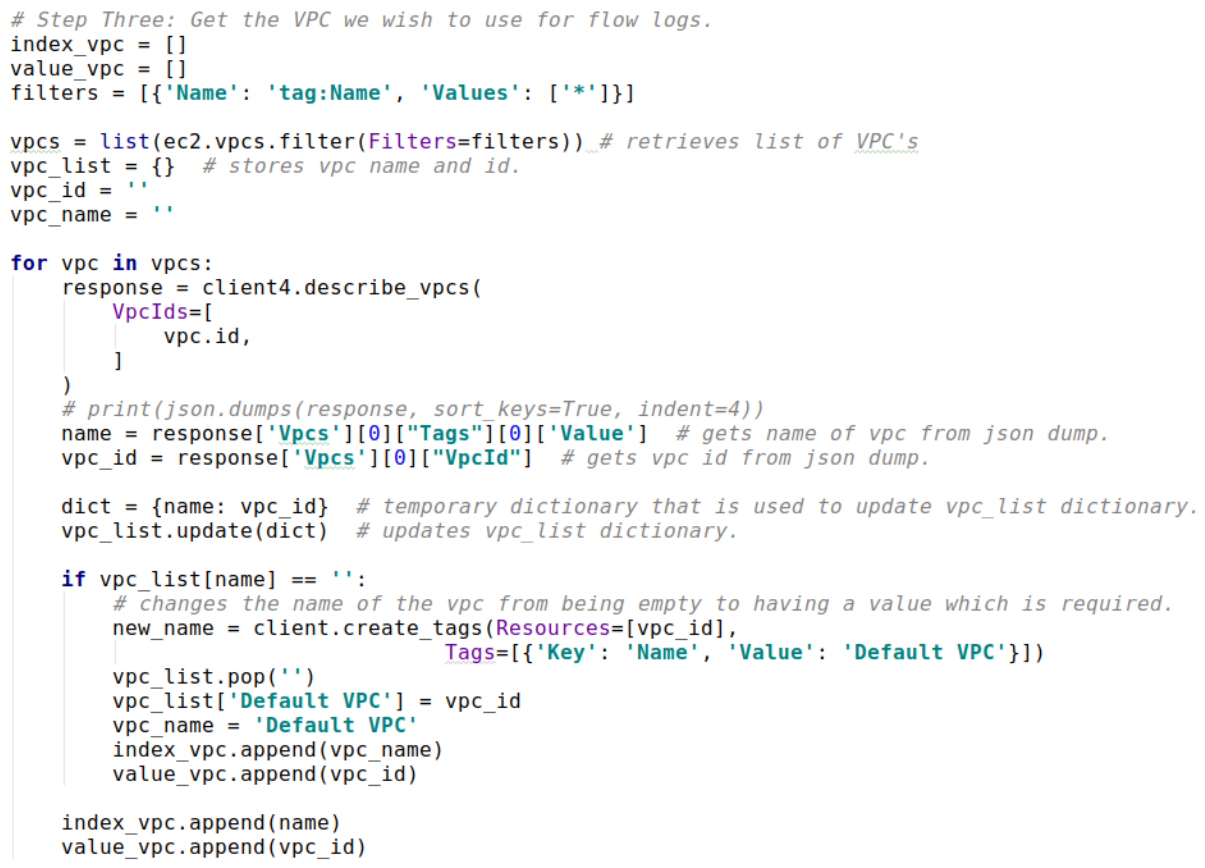
The *describe_log_groups()* function is used to list all the available log groups a user has where they can choose to store their logs.



```
# Step Two: List Log Groups and select one you would like your logs stored in.
index_logGroup = []
value_logGroup = []

response2 = client.describe_log_groups()

for logGroup in range(0, len(response2['logGroups'])):
    logGroupName = response2['logGroups'][logGroup]['logGroupName']
    index_logGroup.append(logGroupName)
    value_logGroup.append(logGroupName)
```

Figure 4.24 List Log Groups.

Finally, the VPC's that are available to use are presented to the user. To scope the results for the VPC ID, it is necessary that a VPC have a name. This is checked and if no name is found then a default name is provided since there should only be one VPC available to the user unless the user has created additional VPC's. Finally, the default VPC is listed to the user.



```
# Step Three: Get the VPC we wish to use for flow logs.
index_vpc = []
value_vpc = []
filters = [{'Name': 'tag:Name', 'Values': ['*']}]

vpcs = list(ec2.vpcs.filter(Filters=filters)) # retrieves list of VPC's
vpc_list = {}  # stores vpc name and id.
vpc_id = ''
vpc_name = ''

for vpc in vpcs:
    response = client4.describe_vpcs(
        VpcIds=[
            vpc.id,
        ]
    )
    # print(json.dumps(response, sort_keys=True, indent=4))
    name = response['Vpcs'][0]["Tags"][0]['Value']  # gets name of vpc from json dump.
    vpc_id = response['Vpcs'][0]["VpcId"]  # gets vpc id from json dump.

    dict = {name: vpc_id}  # temporary dictionary that is used to update vpc_list dictionary.
    vpc_list.update(dict)  # updates vpc_list dictionary.

    if vpc_list[name] == '':
        # changes the name of the vpc from being empty to having a value which is required.
        new_name = client.create_tags(Resources=[vpc_id],
                                      Tags=[{'Key': 'Name', 'Value': 'Default VPC'}])
        vpc_list.pop('')
        vpc_list['Default VPC'] = vpc_id
        vpc_name = 'Default VPC'
        index_vpc.append(vpc_name)
        value_vpc.append(vpc_id)

    index_vpc.append(name)
    value_vpc.append(vpc_id)
```

Figure 4.25 List VPC's.

4.5.2 Creating a Honeypot

Once a device is scanned and cloned, with all the necessary files created, the next step in the process of building a honeypot is to provide a public node that is exposed to the internet which allows the honeypot to be attacked. The clone will be deployed on this public node and will provide the low level-interactions for an intruder. The use of AWS instances provides this and can be implemented through the framework programmatically using the python boto3 module, which provides the necessary functions.

For any script that seeks to perform an action with an AWS account, it must first create a resource or client object. After the boto3 library is imported and a resource or client object is created which according to the boto3 documentation creates an object-oriented interface to Amazon Web Services (AWS). To use resources or provide low-level access to AWS service, you invoke the *resource()* or *client()* functions which opens a session and takes in a service name parameter which depends on what resource or service is being used [79]. Two samples can be seen below.

```
ec2 = boto3.resource('ec2')
vpc = boto3.client('ec2')
```

Figure 4.26 Example resource and client session objects.

The script used to create a honeypot is broken down from top to bottom as follows. With the boto3 library imported and the form successfully submitted, a variable is declared that finds the absolute path to a directory called '*scripts'* where the files generated from the *clone.sh* script is stored.

```
# get absolute path to scripts directory
path = os.path.abspath('scripts')
```

Figure 4.27 Stores absolute path of Scripts directory.

Another variable is declared to store the name of the clone without any spaces which is used later with the bash scripts.

```
clone_name = form.cloneName.data.split(' ')[0].lower()
```

Figure 4.28 Store clone name.

A function is declared next titled '*check_for_port_22()'* to check if the entry's for port 22 is present in the *ports.txt* and *protocol.txt* files provided by the bash script. It is necessary to have port 22 opened on each instance that is deployed to allow the transfer of files to the instance and provide users access to their running instances. If there are no specifications for port 22, then the specifications are added to the files *ports.txt* and *protocols.txt* which will be elaborated on later.

```
# Check if port 22 will be in file... must be included for ssh
def check_for_port_22():
    val = '22'
    # open file and read into list
    with open(path + '/ports.txt', 'r') as p:
        list_values = [port.rstrip() for port in p]
    # Check if port 22 is in ports.txt file
    if val not in list_values:
        # Add port 22 to text file if not already there
        with open(path + '/ports.txt', 'a') as w:
            w.write('22')\
        # Add extra tcp protocol to protocols.txt
        with open(path + '/protocols.txt', 'a') as w:
            w.write('tcp')
```

Figure 4.29 Function checks ports.txt and protocols.txt for port 22 specifications.

With the absolute path to the '*scripts'* directory, then name of the clone with no spaces and a function that checks for port 22 established, the process of creating a honeypot can begin by taking in a target address through the form page that creates honeypots on the framework. This target IP is submitted and past to a python sub process module which uses the path and clone name variables declared earlier to run the *clone.sh* bash script that scans the IP address provided to it, producing information to help with deploying the AWS instance.

```
# execute bash script to scan ip and make required files from the results
subprocess.run(['bash', path + '/clone.sh' , form.target.data, clone_name])
```

Figure 4.30 Sub Process module passing form data to clone bash script.

Once the *clone.sh* bash script is completed, it will have generated a variety of files with different purposes. The two important files that are used to create a security group for an instance with AWS are the *ports.txt* and *protocols.txt* files. These files are used in conjunction with a for loop to dynamically create rules for the security group according to how many ports were picked up by the *clone.sh* script. The rules defined consist of the protocol type (TCP, UDP) associated with a port, the port range that the rule applies to which is specified by the FromPort and ToPort parameters and the CIDR range that the CIDR range applies to. In this script the CIDR range is set to allow any device to connect to these ports since a honeypot is designed to pick up as much traffic as possible. These new rules are then stored in the permissions list as python dictionaries. This portion of the script reads from both the *ports.txt* and *protocols.txt* file which are used to pass in the data sequentially to build the ingress rules for the security group one at a time using a for loop. These rules are then stored in a python list titled permissions.

```
permissions = []

f = open(path + "/ports.txt", "r")
f1 = f.readlines()

t = open(path + "/protocols.txt", "r")
t1 = t.readlines()

for x in range(len(f1)):
    permissions.append({'IpProtocol': t1[x].rstrip(),
                        'FromPort': int(f1[x]),
                        'ToPort': int(f1[x]),
                        'IpRanges': [{'CidrIp': '0.0.0.0/0'}]})
```

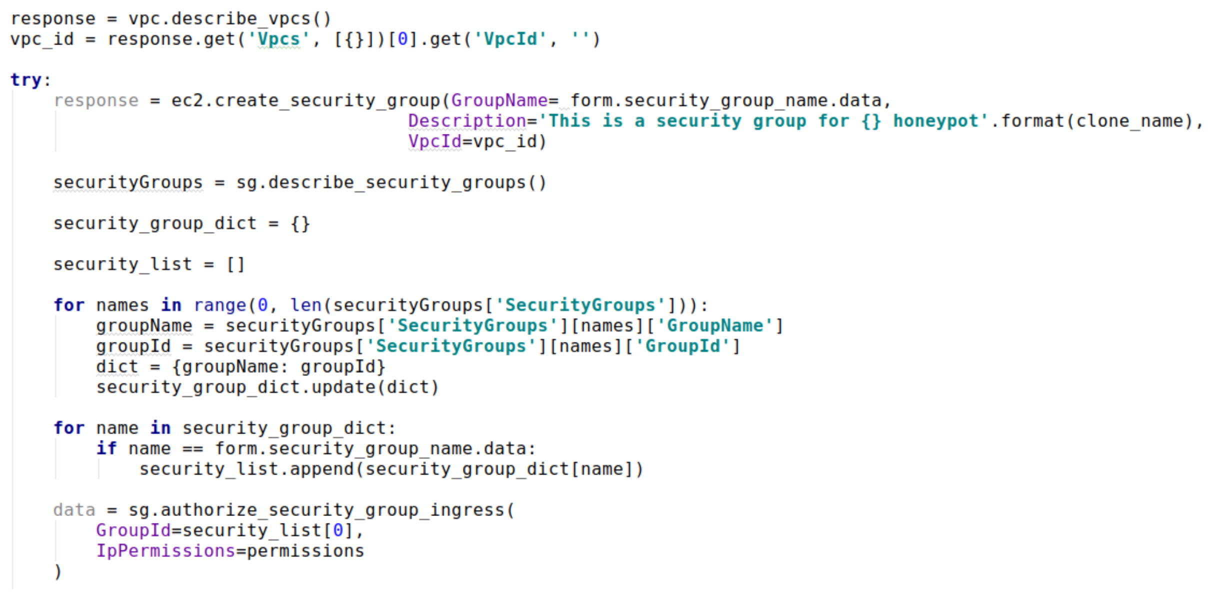
Figure 4.31 Create ingress rules for security group.

With the ingress rules created for the security group, the next step is to create the security group itself. The *create_security_group()* function takes in three parameters which are required according to the boto3 documentation [79]. The first parameter GroupName defines the name of the security group, the second parameter GroupDescription specifies a description for the security group and the VPCid parameter specifies the VPC the security group is being made for [79]. Once the security group is created, the security group id associated with the newly created security group is retrieved and will be used in the *authorize_security_group_ingress().*

As just previously explained, the next step is to create the security group using the *create_security_group()*. The *create_security_group()* function takes in three parameters which are required according to the boto3 documentation [79]. The first parameter GroupName defines the name of the security group, the second parameter GroupDescription specifies a description for the security group and the VPCid parameter specifies the VPC the security group is being made for [79]. Once the security group is created, the security group id is retrieved and will be used in the *authorize_security_group_ingress().*

Since the security group is being created for the user's default VPC, it is necessary to specify the VPC ID when creating the security group. The *describe_vpcs()* function lists all the VPCs a user has. The first VPC is selected which is the default VPC unless the user has created a different VPC. The name of the security group provided through the form page and a brief default description are also incorporated when creating the security group. With these parameters satisfied, the security group is created.

Once the security group is created, it is now time to attach the ingress rules created earlier to the security group using the *authorize_security_group_ingress()* function. In this script below the *authorize_security_group_ingress()* function takes in two parameters which includes GroupId and IpPermissions parameters. The GroupId contains the security group ID retrieved earlier after the security group is created and the IpPermissions contains the list of permissions created earlier in the script. The IpPermissions parameter takes in a list of dictionaries hence why they are stored as such when created [79]. The security group name and ID is found using a for loop that goes through each security group that exists and then adds these values to a dictionary. This dictionary is then used in a for loop to find the name of the security group that was created earlier and its associated ID. Finally, the security group ID and the list of ingress rules created earlier are added as parameters to the *authorize_security_group_ingress()* creating the rules for the security group.



```
response = vpc.describe_vpcs()
vpc_id = response.get('Vpcs', [{}])[0].get('VpcId', '')

try:
    response = ec2.create_security_group(GroupName= form.security_group_name.data,
                                         Description='This is a security group for {} honeypot'.format(clone_name),
                                         VpcId=vpc_id)

    securityGroups = sg.describe_security_groups()

    security_group_dict = {}

    security_list = []

    for names in range(0, len(securityGroups['SecurityGroups'])):
        groupName = securityGroups['SecurityGroups'][names]['GroupName']
        groupId = securityGroups['SecurityGroups'][names]['GroupId']
        dict = {groupName: groupId}
        security_group_dict.update(dict)

    for name in security_group_dict:
        if name == form.security_group_name.data:
            security_list.append(security_group_dict[name])

    data = sg.authorize_security_group_ingress(
        GroupId=security_list[0],
        IpPermissions=permissions
    )
```

Figure 4.32 Creating security group and adding ingress rules.

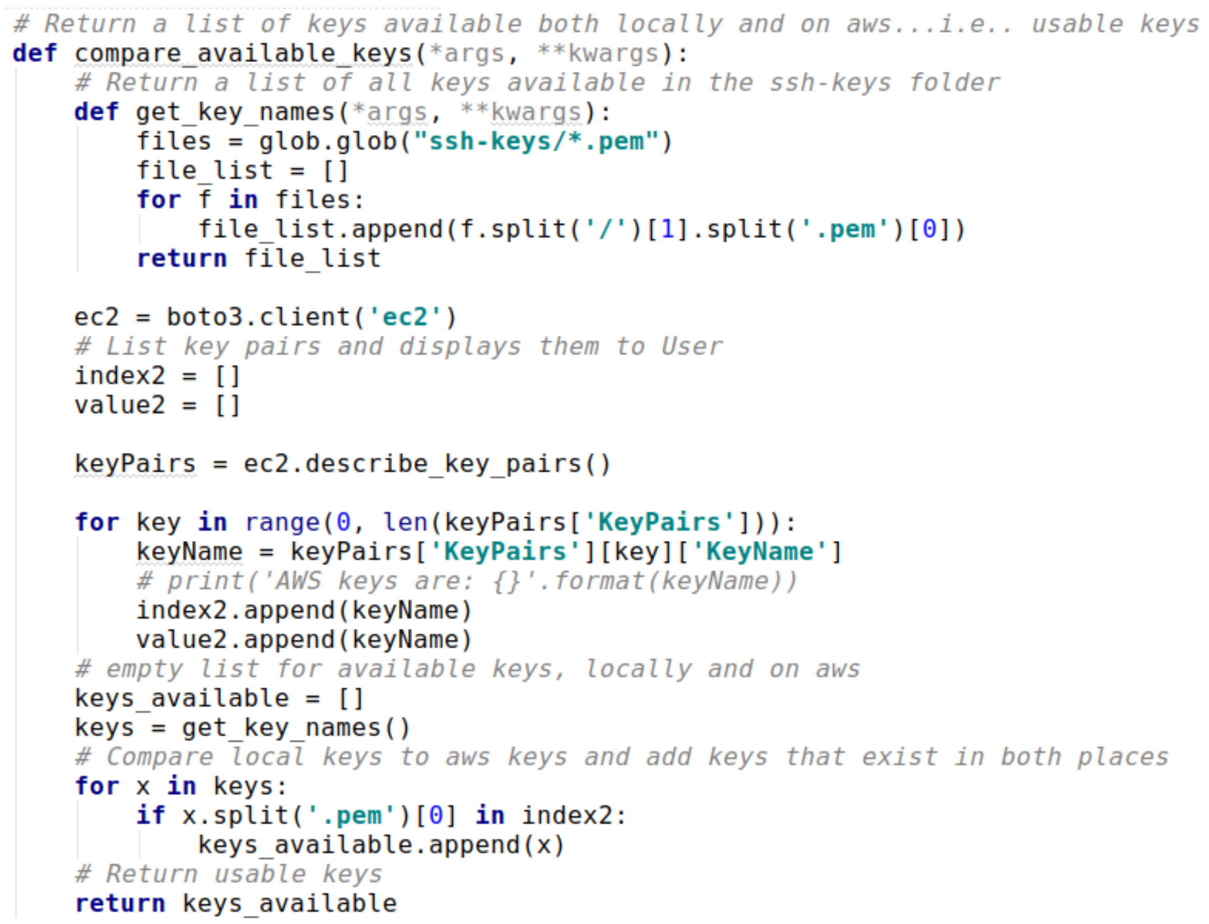
The next step is to create the instance using the *create_instances()* function, which can be used to deploy an instance. The first parameter in the function specifies the ID of the Amazon Machine Image (AMI) to be used when the instance is launched. This AMI, for the purposes of this project, indicates the operating system of the honeypot being deployed [78]. In this script, the AMI image ID specified is the Amazon Linux 2 free tier AMI (64-bit x86), which is the only option available for users to select in the form.

The MinCount and MaxCount parameters specify the number of instances to deploy using an integer. In the script the MinCount and MaxCount are set to one so only a single instance is deployed each time a script is used. Both parameters are required by the function in order to deploy an instance [20].

The instance type parameter defines the hardware of the EC2 instance being deployed. In this script the t2.micro instance type is the only option a user can select. The decision to use this instance type was explained in the Design Chapter under the AMI and Instances heading (3.1.1.3) [78].

The parameter entitled KeyName refers to the name of the KeyPair that allows a user to access an instance once it is launched. It is important to note that if a KeyPair is not specified when an instance is launched then a user will not be able to remotely login to the instance unless the AMI used enables the user to login through other means [78]. This is an essential parameter that uses a resource that was created in another script earlier. The key selected by the user through a form when creating the honeypot is passed to the function creating the instance.

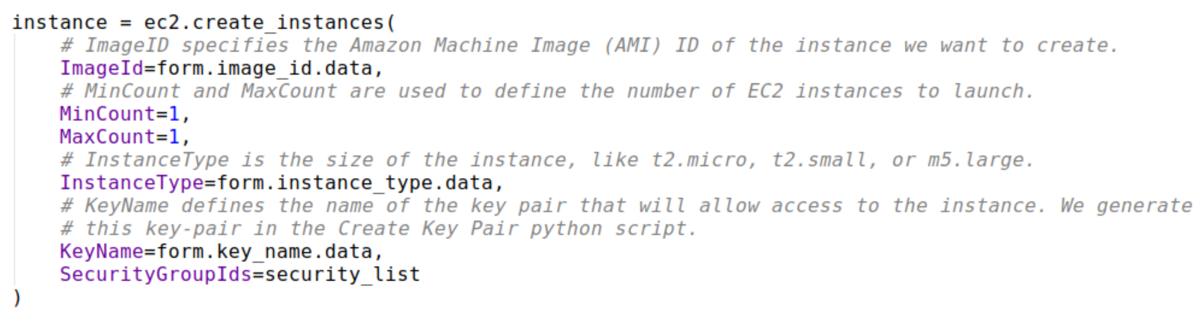
The keys available to a user are displayed on the form. A function called *compare_available_keys()* is used to find the available keys. Inside this function is another function called get_key_names() that retrieves the name of the keys located in the *ssh-keys* folder. The script then looks for all the available keys created on the users AWS account. Finally, a list of the keys available to the user is returned and used in a select field on the flask form.



```python
# Return a list of keys available both locally and on aws...i.e.. usable keys
def compare_available_keys(*args, **kwargs):
    # Return a list of all keys available in the ssh-keys folder
    def get_key_names(*args, **kwargs):
        files = glob.glob("ssh-keys/*.pem")
        file_list = []
        for f in files:
            file_list.append(f.split('/')[1].split('.pem')[0])
        return file_list

    ec2 = boto3.client('ec2')
    # List key pairs and displays them to User
    index2 = []
    value2 = []

    keyPairs = ec2.describe_key_pairs()

    for key in range(0, len(keyPairs['KeyPairs'])):
        keyName = keyPairs['KeyPairs'][key]['KeyName']
        # print('AWS keys are: {}'.format(keyName))
        index2.append(keyName)
        value2.append(keyName)
    # empty list for available keys, locally and on aws
    keys_available = []
    keys = get_key_names()
    # Compare local keys to aws keys and add keys that exist in both places
    for x in keys:
        if x.split('.pem')[0] in index2:
            keys_available.append(x)
    # Return usable keys
    return keys_available
```

Figure 4.33 Displaying Keypairs to user.

The final parameter labelled SecurityGroupIds indicates the security group to use when deploying the EC2 instance. It is worth noting that when a security group id is not specified, then a default security group is assigned to the instance being launched [78]. The security group that is created earlier when the script is executed is assigned to the instance when it is being created.



```python
instance = ec2.create_instances(
    # ImageID specifies the Amazon Machine Image (AMI) ID of the instance we want to create.
    ImageId=form.image_id.data,
    # MinCount and MaxCount are used to define the number of EC2 instances to launch.
    MinCount=1,
    MaxCount=1,
    # InstanceType is the size of the instance, like t2.micro, t2.small, or m5.large.
    InstanceType=form.instance_type.data,
    # KeyName defines the name of the key pair that will allow access to the instance. We generate
    # this key-pair in the Create Key Pair python script.
    KeyName=form.key_name.data,
    SecurityGroupIds=security_list
)
```

Figure 4.34 Create Instance.

With all the above parameters satisfied, an EC2 instance will be deployed. It is worth mentioning that there are a few rules that apply when deploying an EC2 instance. If an instance is launched without a subnet specified, then a default subnet is taken from the user's default VPC. Each instance when launched has a primary private IP address that, if not specified, is chosen from the IPv4 range of the subnet assigned to the instance. Finally, if an AMI is selected for which a user has not subscribed to then the launch will fail [78].

4.5.2.1 Managing EC2 Instances

Once a honeypot is created by launching EC2 instance, it is possible to allow the user to remotely manage their EC2 instances (honeypots) from the IOT Honeynet Framework, which is working in line with the holistic approach the IOT Honeynet Framework is designed to achieve. Achieving this in a programmatic way is quite simple when using the python boto3 library. The IOT Honeynet Framework uses this library to programmatically manage EC2 instances, allowing the user to stop, start, list and terminate instances.

To change the state of an instance, the instance ID needs to be passed to the *instance()* function and then use the *stop(), start()* or *terminate()* method depending on the desired action the user wishes to take with an instance. By allowing a user to provide an instance ID, it is possible manage the state of an instance as demonstrated in the below script.

```
ec2 = boto3.resource('ec2')

ec2.Instance(form.instanceid.data).start()

ec2.Instance(form.instanceid.data).stop()

ec2.Instance(form.instanceid.data).terminate()
```

Figure 4.35 Python Scripts that Manage Instance States.

To list all honeypots currently created, an ec2 resource will be created. This EC2 resource is used to iterate through all EC2 instances, then access the individual properties of each virtual machine and print the list of all EC2 instances and their respective properties on the console.

```
{% for instance in ec2.instances.all()%}

    <article class="media content-section">
        <div class="media-body">
            <div class="article-metadata">
                <h1 class="article-title"> Honey Pot </h1>
                <p class="article-content">{{instance.id}}</p>
                <p class="article-content">{{instance.platform}}</p>
                <p class="article-content">{{instance.instance_type}}</p>
                <p class="article-content">{{instance.public_ip_address}}</p>
                <p class="article-content">{{instance.image.id}}</p>
                <p class="article-content">{{instance.state}}</p>
```

Figure 4.36 Display Honeypots on Framework.

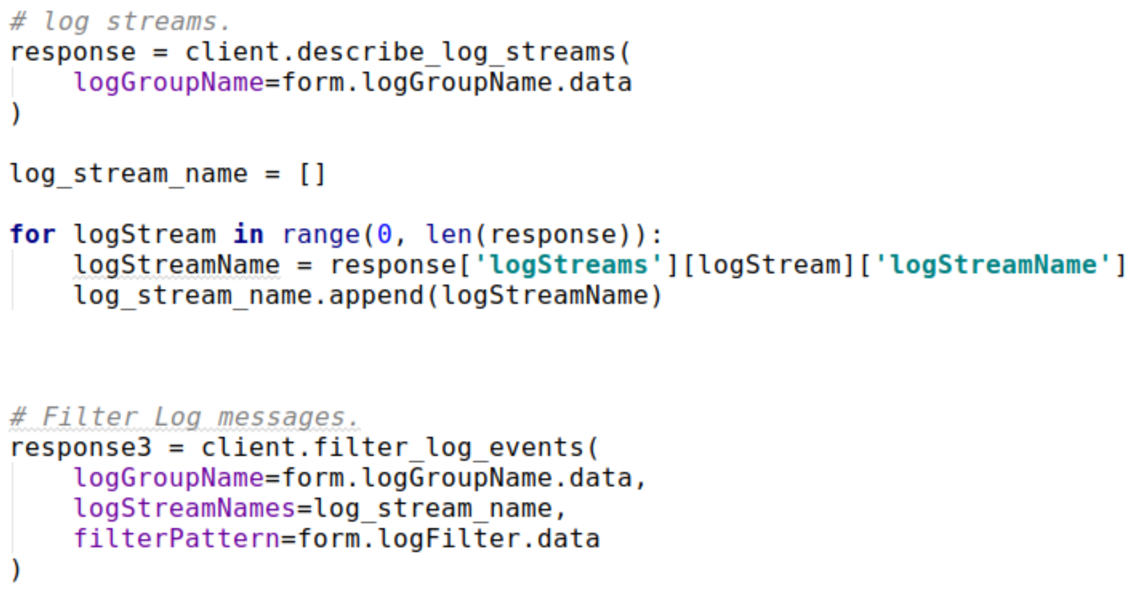
4.5.2.2 View Logs

To allow a user to view the logs collected from a honeypot, a form was created which requested the user to provide the name of the log group they stored their logs in and the private IP address of one of their currently running honeypots. These options were listed to the user by using the below script.

```
# List Log Groups.
index_logGroup = []
value_logGroup = []

response2 = client.describe_log_groups()

for logGroup in range(0, len(response2['logGroups'])):
    logGroupName = response2['logGroups'][0]['logGroupName']
    index_logGroup.append(logGroupName)
    value_logGroup.append(logGroupName)

# List Private IP addresses of EC2 instances
index_PrivateIP = []
value_PrivateIP = []
for instances in ec2.instances.all():
    index_PrivateIP.append(instances.private_ip_address)
    value_PrivateIP.append(instances.private_ip_address)
```

Figure 4.37 List Log Groups and Honeypot Private IP addresses.

The log group name is used to return a list of the log streams currently in the log group using the function *describe_log_stream()* and the private IP address of the instance was used as the filter in the *filter_log_events()* function. This returned all logs associated with the private IP address of the selected honeypot.



```
# log streams.
response = client.describe_log_streams(
    logGroupName=form.logGroupName.data
)

log_stream_name = []

for logStream in range(0, len(response)):
    logStreamName = response['logStreams'][logStream]['logStreamName']
    log_stream_name.append(logStreamName)


# Filter Log messages.
response3 = client.filter_log_events(
    logGroupName=form.logGroupName.data,
    logStreamNames=log_stream_name,
    filterPattern=form.logFilter.data
)
```

Figure 4.38 Return Filtered Logs.

4.6 Setting up Flask

Installing Flask has a requirement of having Python installed on one's computer. For this project Python version 3.7 was used. PyCharm, a Python IDE was also used for the development of this project and was used to create a virtual environment which is a "private copy of the Python interpreter onto which you can install packages privately, without affecting the global Python interpreter installed in your system" [64].

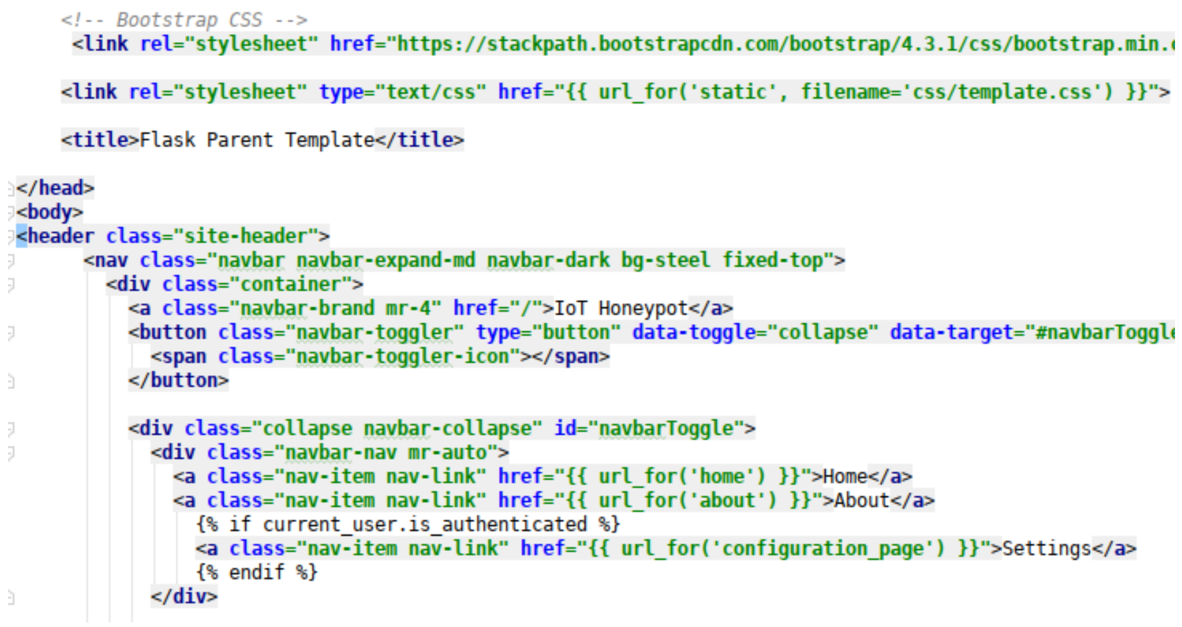
Once the above requirements are initialized, the next step is to install flask itself. Pip is a Python utility tool that can be used to install flask. It is included by default with the Python binary installers and starts from Python version 3.4 [80]. Flask can be installed directly from PyCharm terminal using the following pip command ***pip install flask***. Alternatively, the Flask package can be installed from PyCharm "Available Packages" menu located in *File>>Settings>>Project Interpreter>>Install (Alt + Insert).*

Additionally, if a web framework is using bootstrap the command ***pip install flask-bootstrap*** can be used from PyPI to install this extension or can be directly used by importing the *bootstrap.min.css* and *bootstrap.min.js* files into a base template. Bootstrap also offers an extension for Flask called *Flask-Bootstrap* which provides a base template that has the Bootstrap framework installed. As one can observe, there are multiple ways to implement Bootstrap into a web framework. Flask supports all versions of Bootstrap, for this project version 4 was used [81].

4.6.1 Templating and Bootstrap

A web application can be written in Flask without the need of any third-party templating engines. Flask provides support for Jinja2 templating language by default, but any other templating engine can also be used due to Flasks extensible nature. “Templates are files that contain static data as well as placeholders for dynamic data” [82]. A template can then be rendered using the *render_template ()* method to produce a final document. Templating is useful for code structuring and allows a user to create formatted text like HTML. Jinja behaves similarly to Python [82]. It uses special delimiters to distinguish Jinja syntax from the static data. Jinja uses curly braces *{{…}}* as expressions that will be printed to the template output. Jinja also uses *{%...%}* as control flow statements, which are equivalent to *if* and *for* statement in Python. *{#...#}* can be used for comments that will not be included in the template output [83].

“Bootstrap is an open source framework from Twitter that provides user interface components to create a clean and attractive web pages that are compatible with all modern web browsers” [64]. Bootstrap doesn’t just offer a skeleton for web frameworks that look professional and feels clean, but it also handles desktop, tablet and phone screen sizing as well as offering very customizable layouts. Bootstrap is a client-side framework, meaning its server is only used to provide HTML responses to style sheets and JavaScript referenced files. Jinja2 and Bootstrap work hand to hand in displaying a template output. Jinja2 implements the template inheritance by referencing the *bootstrap.min.css* and *bootstrap.min.js* files, while Bootstrap outputs a base template that includes all the Bootstrap CSS and JavaScript files. Using Bootstrap in Flask requires the web application to have a very specific layout of templates. By default, Flask expects the templates to be located inside a folder called *templates*. If the layout is correct, then Flask can automatically read the contents and make them available for use with the *render_template ()* method [64].



```
    <!-- Bootstrap CSS -->
     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.

    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/template.css') }}">

    <title>Flask Parent Template</title>

</head>
<body>
<header class="site-header">
      <nav class="navbar navbar-expand-md navbar-dark bg-steel fixed-top">
        <div class="container">
          <a class="navbar-brand mr-4" href="/">IoT Honeypot</a>
          <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarToggl
            <span class="navbar-toggler-icon"></span>
          </button>

          <div class="collapse navbar-collapse" id="navbarToggle">
            <div class="navbar-nav mr-auto">
              <a class="nav-item nav-link" href="{{ url_for('home') }}">Home</a>
              <a class="nav-item nav-link" href="{{ url_for('about') }}">About</a>
                {% if current_user.is_authenticated %}
                <a class="nav-item nav-link" href="{{ url_for('configuration_page') }}">Settings</a>
                {% endif %}
            </div>
```

Figure 4.39 template.html using Bootstrap, expressions and control flow statements.

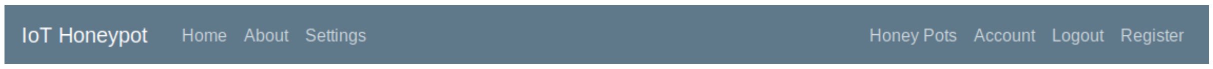
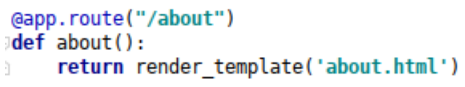


Figure 4.40 Navbar displayed on the Web Framework.

4.6.2 Route Functions

Web browsers are seen as clients that communicate with a web server by sending requests. The application instance needs to know what code needs to run for each URL requested, and so these URLs need to be mapped to Python functions in one's code. The functions that handles URL requests are known are routes in Python and can be used to access a desired page directly without navigating to it through the home page.

A route can be implemented into a Flask application by using the app.route decorator, which registers the decorated function as a route.

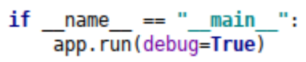


```
@app.route("/about")
def about():
    return render_template('about.html')
```

Figure 4.41 routes.py app.route decorator.

The above example registers the function *about ()* as the handler for the applications about page. The URL "/about" is now bound to the *about ()* function. Navigating to the following URL http://www.example.com/about on a browser would redirect the user to the about page by triggering the *about ()* to run on a server. The return value of this function known as a response determines what information will be displayed on the about page [64].

A minimal application needs to also have a run method that can launch a Flask web server. While an application is under development, debugging mode can be enabled by setting the debug property of the application to True.



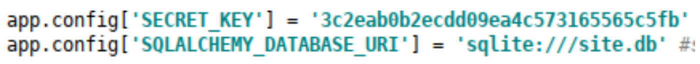
```
if __name__ == "__main__":
    app.run(debug=True)
```

Figure 4.42 run.py run method.

4.6.3 WTF Forms and Form Handling

Flask's built in features are sufficient for the handling of web forms, but there are numerous tasks that can be tedious to perform. Such examples include the generation of HTML code for forms and the validation of submitted form data. Flask-WTF extension makes working with forms a lot easier and convenient.

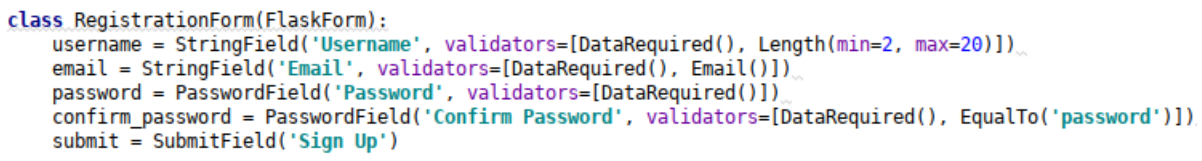
"By default, Flask-WTF protects all forms against Cross-Site Request Forgery (CSRF) attacks" [64] and to implements this feature a developer needs to configure an encryption key and add it to the *app.config* object with a *SECRET_KEY* configuration variable. A random 16 digit SECRET_KEY can be generated by importing the secrets module and using the *secrets.token_hex(16)* command through the python console. Flask-WTF then uses this encryption key to generate encrypted tokens that are used for authenticity verification of requests with form data. "The *app.config* dictionary is a general-purpose place to store configuration variables used by the framework, the extensions, or the application itself" [64]. "The SECRET_KEY configuration variable is used as a general-purpose encryption key by Flask and several third-party extensions" [64].



```
app.config['SECRET_KEY'] = '3c2eab0b2ecdd09ea4c573165565c5fb'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db' #
```

Figure 4.43 __init__.py using a SECRET_KEY and SQLALCHEMY_DATABASE_URI configuration variables.

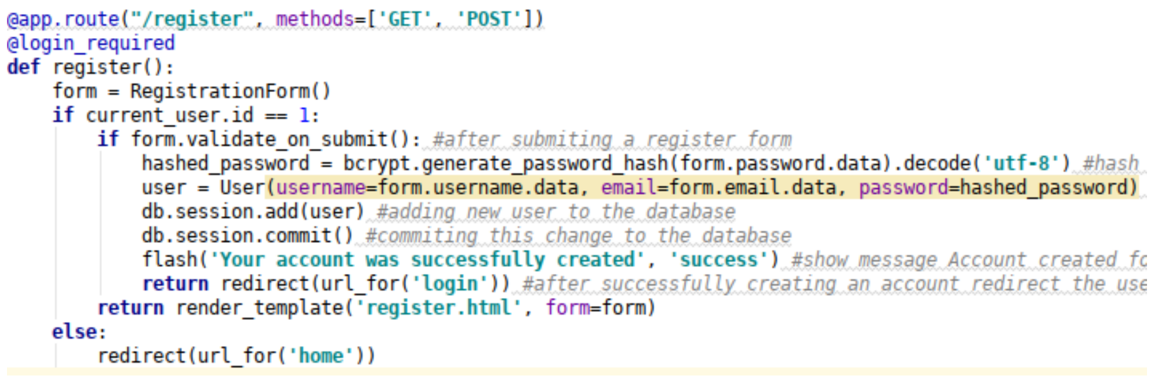
Form classes are web form classes that inherit from a *Form* class. A web form class defines the number of fields and the field type in the form. A form field is represented by an object and this object can have one or more *validators* (that are optional), which are constraints or requirements put in place by the developer to ensure the user inputs valid data, or if the users' data is essential for the completion of a form which can be marked by the *Required ()* validator. "Fields in a form are defined as class variables, and each class is assigned an object associated with the field type" [64].



```
class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=2, max=20)])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Sign Up')
```

Figure 4.44 forms.py class with validators.

Flask-WTF forms can use *GET* and *POST* requests to handle form field data. Alternatively using *POST* requests to handle form data is a preferred method because it is done in a way that the URL query string is invisible in the browser address bar to a user. Forms use a *validate_on_submit ()* method to return either a *True* if the data is valid or *False* otherwise. The return value of the *validate_on_submit ()* method decides from the user input whether to render or process the form. The method can also decide to perform other functions like *redirect ()* to redirect the user to a different web page after a successful form completion *or flash ()* display a "success" or danger" message if the user successfully created an account or entered the data in one of the form fields incorrectly [64]. "success" and "danger" are Bootstrap alert classes that are used to display different coloured flash messages based on the alert class.



```
@app.route("/register", methods=['GET', 'POST'])
@login_required
def register():
    form = RegistrationForm()
    if current_user.id == 1:
        if form.validate_on_submit(): #after submiting a register form
            hashed_password = bcrypt.generate_password_hash(form.password.data).decode('utf-8') #hash
            user = User(username=form.username.data, email=form.email.data, password=hashed_password)
            db.session.add(user) #adding new user to the database
            db.session.commit() #commiting this change to the database
            flash('Your account was successfully created', 'success') #show message Account created fo
            return redirect(url_for('login')) #after successfully creating an account redirect the use
        return render_template('register.html', form=form)
    else:
        redirect(url_for('home'))
```

Figure 4.45 routes.py function using GET and POST methods and using the validate_on_submit () method.

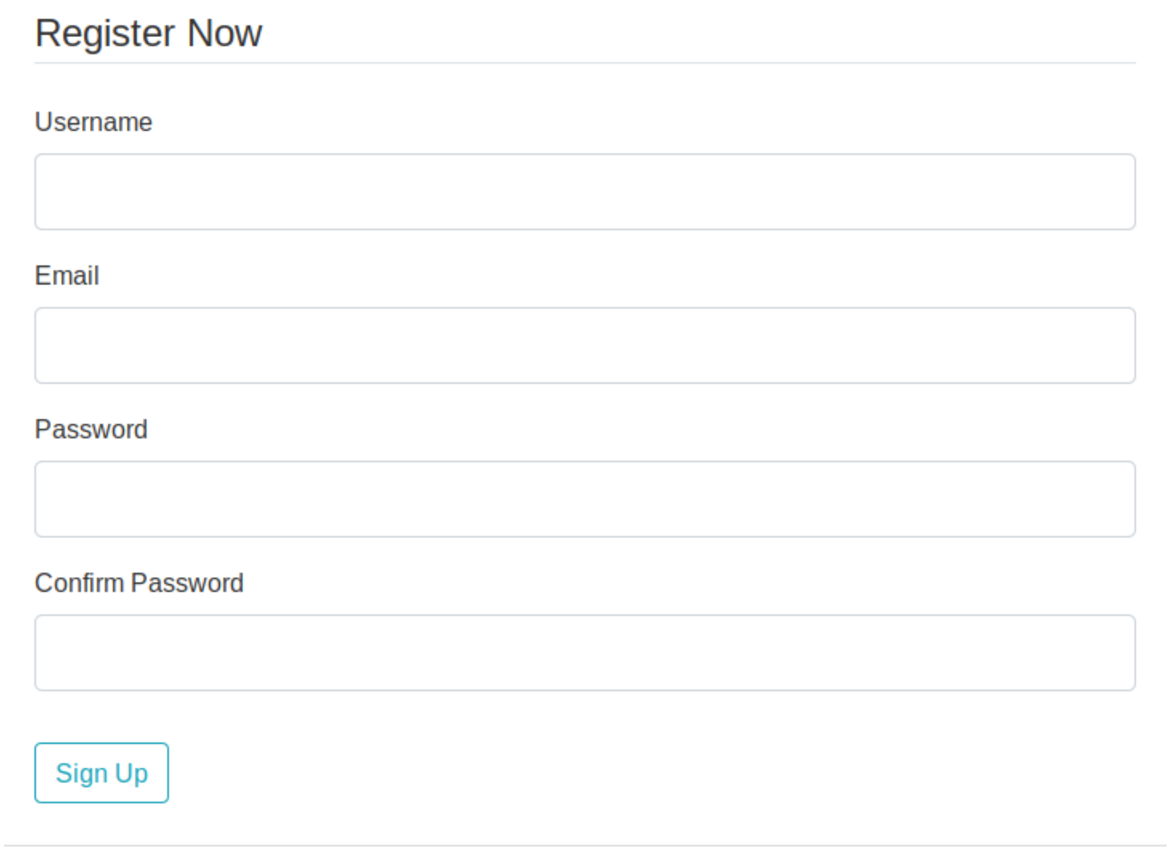


Figure 4.46 Registration Form displayed on the Web Framework.

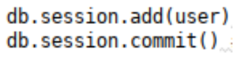
4.6.4 Data Modelling with SQL Alchemy

A database stores structured and easily accessible sets of data. Applications can issue queries to retrieve quantities of specific data that is being required. “A query is a request for data or information from a database table” [84]. Web applications commonly use relational model databases which are also known as Structured Query Language (SQL) databases. A relational model database stores data in a tabular form, as in contrast to a database system that stores data as files. NoSQL databases have also become popular that store semi-structured data or data associated with arrays [64].

Flask uses a Flask-SQLAlchemy extension, which is a simplified version of SQLAlchemy. SQLAlchemy uses a relational database model designed and implemented by the Pocoo Team who are also the founders of Flask. SQLAlchemy is the default database for the Python SQL library that has a database abstraction layer (DBAL) which makes it compatible with other models of SQL databases [85].

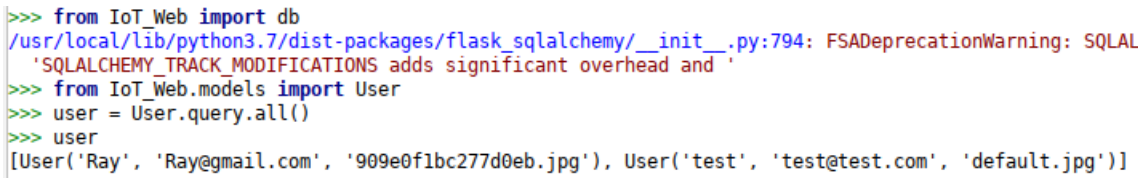
Flask-SQLAlchemy as mentions previously is an extension that is designed specifically for Flask applications. “It allows you to configure the SQLAlchemy engine through your configuration file and binds a session to each request, giving you a transparent way to handle transactions” [85]. Flasks extension of SQLAlchemy also offers high-level object-relational mapping (ORM) which is required for populating objects with data in a database [64].

A simple SQLite database can be initialized by configuring the *SQLALCHEMY_DATABASE_URI* key in the Flask configuration file. It's also important to keep in mind that SQLite databases don't have a server, and so the hostname, username and password are intentionally excluded. It's also necessary to instantiate the *db* object in the Flask configuration file from the SQLAlchemy class as it provides access to all functionality of the Flask-SQLALchemy extension. Committing changes to the database can be achieved by using the *db.session.commit ()* method [64].



```
db.session.add(user)
db.session.commit()
```

Figure 4.47 routes.py using db.session.commit () method.



```
>>> from IoT_Web import db
/usr/local/lib/python3.7/dist-packages/flask_sqlalchemy/__init__.py:794: FSADeprecationWarning: SQLAL
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
>>> from IoT_Web.models import User
>>> user = User.query.all()
>>> user
[User('Ray', 'Ray@gmail.com', '909e0f1bc277d0eb.jpg'), User('test', 'test@test.com', 'default.jpg')]
```

Figure 4.48 Users committed to the database.

Chapter 5: Analysis

This chapter focuses on analysing the final developed solution and evaluating the features implemented for the framework. The objective of this thesis was to create a framework that could provide a seamless approach to deploying and monitoring honeypots, while also providing an innovative feature of being able to clone devices directly into honeypots through the framework. The final solution is an IOT Honeynet Framework that makes the deployment and management of dynamically created honeypots, a holistic process through the framework. These honeypots can then be used for research or production purposes.

5.1 Framework Features

This section evaluates the main features of the project that were implemented and present in the final version of the IOT Honeynet Framework.

5.1.1 Cloning Device

The unique selling point of the framework was to provide a means for a user to clone a device and emulate its services. Cloning a device was successfully implemented into the framework. The cloning feature allows a user to create a low-level interactive honeypot that emulates the services on the device that was cloned. This provides a few added benefits which are missing from current projects.

The ability to clone a device and the services it is running into a honeypot is quite powerful. Rather than searching for a honeypot that can imitate a certain device, the IOT Honeynet Framework provides a novel solution to imitating devices that a user wants to test for security threats. These new cloned honeypots can then be shared and used by other users, making the distribution and updates of honeypots a regular occurrence.

Being able to clone a device also helps reduce the complexity in creating a new honeypot. Discovering all the services and ports that are on any given device is a labour-intensive task which is avoided by automating the process itself. Tweaking existing honeypots can also present technical challenges that would be a barrier to using honeypots for users with weak technical backgrounds.

Another advantage that the cloning feature provides is the speed at which a new type of honeypot can be created and deployed. Designing a new honeypot even with low-level interaction takes time and can be quite complex, requiring users to have a good technical knowledge of how to develop honeypots. By enabling a user to simply pick a device to clone and produce a honeypot that is immediately deployed into a honeynet is highly advantageous and reduces the time it takes to deploy a honeypot for a user.

5.1.2 Deploying and Managing Honeypots

The ability to deploy and manage honeypots is an essential feature the project aimed to provide. This was achieved successfully and implemented into the framework. The process of learning how to use a hosting provider and making a honeypot public can be a daunting challenge that absorbs time and resources. The IOT Honeynet Framework is able to perform and automate many tasks with minimal user input when deploying and managing honeypots on the AWS hosting service.

The IOT Honeynet Framework supports the setup and configuration that needs to be performed before making a honeypot. As previously explained in the earlier chapters for design and implementation, the files created after cloning the device are used to deploy a python server in a docker container which is stored on an EC2 instance. There are several requirements that are necessary before making a honeypot which would need to be configured by a user. These requirements which are covered in detail in *Chapter 3,* are necessary to allowing the framework to operate correctly. By enabling the user to perform these initial configurations through the framework, it allows them to avoid spending time on researching and configuring these settings through their AWS account, mitigating errors and time wasted on learning AWS services.

The ability to create honeypots (EC2 Instances), manage their states and view the logs captured from the honeypot are all aggregated into the framework. This is useful for the user who does not have to go between their AWS account and the cloning script which would involve the transfer of files and the manual input of data through their AWS account to allow the creation, deployment and management of EC2 instances. This avoids unnecessary complications and errors, while allowing the user to focus on other tasks without wasting time in their AWS account.

5.1.3 Visualisation Data

The visualisation of data is a key feature that was achieved and successfully implemented in the framework. It is possible to view the logs captured from the honeypots deployed and filter the logs to search for certain criteria that is of importance to the user. The visualization of data is limited to the logs that are returned to the framework and does not provide any means to visualize the information collected into graphs.

Chapter 6: Conclusions and Results

The primary goal set out for this research was to develop an IoT Honeynet Framework that would streamline the process of utilising honeypots as a security solution to mitigate threats. The final developed product provided valuable insight on how the considerations that are made when designing a system that provides a holistic approach to the creation and management of honeypots.

6.1 Conclusions

The research has provided a substantial amount of knowledge gained from overcoming the challenges that were presented during the design and development of the IoT Honeynet Framework. The major achievements and findings drawn from this research are described in the following sections.

6.1.1 IOT Honeynet Framework

The major contribution of this research has been the development of a novel IoT Honeynet Framework that provides a holistic approach to creation and management of honeypots. The advantages of such a framework is evident where it can provide a seamless approach to creating low-level interactive honeypots that are deployed as part of a honeynet and then managed through the framework.

There is tangible evidence that the IoT Honeynet Framework developed during this research provides greater efficiency in creating and managing honeypots, then what would've been achieved from using currently available solutions. This is made evident from the list of major elements which make the framework beneficial seen below.

- Cloning an active device on a network and deploying it as a low-level honeypot by emulating its services, which is then attached to a honeynet makes the process of creating honeypots a dynamic process and does not involve the setup or active configuration of current honeypots to emulate a desired device.
- The setup and configuration of AWS services hosting is mostly automated or streamlined to require minimal input from a user when performing AWS (Amazon Web Services) service requests. This means a user requires less technical background knowledge with AWS services when performing actions such as creating instances where the honeypot is stored or performing the initial setup and configuration for allowing logs to be collected and stored.
- The aggregation of information from currently running honeypots and the logs collected from these honeypots provides a combined approach to managing and monitoring the state of the

deployed honeypots, while also observing the log data being collected while the honeypots are operating.

- An added benefit from developing a script that create honeypots by cloning devices and emulating their services, is that a repository of clone's is created. This provides new honeypots that had not previously existed. The constant creation of honeypot clones by users provides a repository of honeypot clones that can be redistributed and used by other framework users.

The development of the framework has effectively addressed many of the barriers that prevent the use of honeypots in a production or research role. This has added a valuable contribution to the field of security.

6.1.2 The Potential of Containers and Cloud Computing

The use of containers and cloud computing services such as Amazon Web Services (AWS) was used as the foundation for the IoT Honeynet Framework developed during this research. It serves as a proof-of-concept with regards to the usability of such technologies in enabling the ability to automatically deploy and configure different honeypots in a fast and efficient manner.

As this research demonstrates, the combination of using containerisation and hosting services is a potent mix, that offers the potential for massive gains from honeypots both in a production and research role. The flexibility and scalability of these technologies allows the potential for future frameworks like the one developed during this research to be further expanded on and become less restricted by resource costs or constrictions.

6.1.3 The Future of IOT Security

It is evident from this research that the future of IoT Security is uncertain. Society is becoming more reliant on the use of interconnected devices which can control critical service infrastructures and applications. This creates a fertile environment for cyber threats to grow and develop which will motivate attackers to continue searching for vulnerabilities to exploit and take advantage off.

There will need to be huge innovation in the area of IoT Security itself before the major problems facing interconnected devices are addressed. The massive demand in the market for interconnected devices is only growing and without any consideration given to the development of good security in these devices considered, the security threats will continue to be exacerbated. It is likely that until there is a truly market-driven incentive for manufacturers to implement decent security mechanisms in their products, this is unlikely to change.

6.2 Future Work

Whilst a functional IoT Honeynet framework was successfully developed, there are many additional features and work that can be added to the framework. This section covers further opportunities for additional developments and features which can be added to the existing IoT Honeynet Framework.

6.2.1 Complete Platform Independence

The IoT Honeynet Framework was implemented using the boto3 amazon SDK. This restricted the range of hosting platforms that could be used for hosting the honeypots created by the framework. It is clear from the challenges of creating an IoT Honeynet Framework that could provide a holistic approach to creating, deploying and managing honeypots, it would be difficult to provide a solution to achieve complete platform independence.

Docker could provide a means of creating complete platform independence by having all features such as logging and creating honeypots achieved through a framework that was a separate docker instance. This environment could be deployed and used on any hosting service or even on personal equipment without having to leverage the boto3 SDK for AWS.

6.2.2 Improve Honeypot Design

The level of interaction provided by a honeypot created from the cloning script used by the framework is not particularly high. There is opportunity for the improvement in the level of interaction that a honeypot can provide. There were no previous examples of honeynet frameworks that provided the ability to clone a device and deploy it as a honeypot. An integral part of creating this honeypot was emulating the services provided by the cloned device. Emulating these services proved to be quite challenging as attempting to install the exact version of a services running on a port for any given device proved difficult, especially when there is no information in the banner to indicate what service version running or if any service is even running at all.

Some improvements for the honeypot could include the following elements.

- Providing further isolation between the honeypot and the host operating system. This removes the potential for an attacker to gain access that was not intended to be possible. This could lead to an attacker damaging the configuration of the honeypot, damaging important files or changing the data that is returned, resulting in useless data.

- Providing a wider range of data that is captured from the honeypot such as the metrics of the honeypot or the activity of an attacker once they gain access to a honeypot, however this would only be possible if a higher level of interaction could be implemented for a honeypot.
- Use a solution that avoids the transfer of files between the host device and the honeypot. It is necessary to transfer files that create the honeypot to the instance that hosts the honeypots. This means ports like the SSH or FTP port need to be open and available when creating a honeypot. This can reduce the security and deception of the honeypot. Solutions such as port knocking can be used to fix this.

Implementing these improvements would greatly increase the overall level of the honeypots being deployed through the framework. This would also make the scanning script more versatile and provide a continually growing repository of high-level interactive honeypots that could be used by future researchers.

6.2.3 Visualisation of Data

The visualisation of log data is an essential component for any honeypot. The main objective of the framework from the outset was to provide a feasible and practical solution to deploying and managing honeypots in a holistic approach. The visualisation of data was considered and implemented but the representation of data and the filtering of data was not implemented in a way that would be of most relevance to a researcher.

The visualisation of data is represented in the framework by displaying the logs in a table format. This does not provide a means to quantify or analyse specific results in a clear or concise way. It would be considered beneficial to implement a data visualisation tool such as Splunk or the ELK stack, which provides a means to filter and visualize the data in a more meaningful manner.

This would provide more setup and configuration for the user but would allow a more meaningful representation of data. In addition, ongoing maintenance overheads must be considered by virtue of the addition of another separate system for logging.

6.3 Closing Remark

This research has seen the development of an IoT Honeynet Framework which was developed to provide a holistic approach to deploying a honeypot security solution for the purposes of a production role in an organization or as a research role for learning more about the cyber threat landscape. Though there is significant work remaining to bring this system to a production-ready state, there is many elements in the current system that addresses the significant challenges that are being faced in these types of infrastructures by providing a feasible and practical solution for the creation and deployment of honeypots.

The use of honeypots will only continue to grow as the gap between currently implemented security mechanisms and newly evolving threats continues to grow. The need for an easy-to-use security solution that is quick to implement and adaptive to modern threats is necessary for the security of critical infrastructure services and IoT devices.

The crucial takeaway from this research is the need to remove barriers from the deployment of security solutions such as the implementation of honeypots, making the adoption of such security solutions more cost effective and easy to implement. Until security solutions are more widely included as part of software design and implementation, it is likely that more risks will be mitigated and that fewer vulnerabilities will arise in modern applications and devices. Though it is only theoretically possible to eliminate all vulnerabilities present in a system, the implementation of security solutions is essential to mitigating the risk of modern cyber threats.

Bibliography

[1] Europa.eu. (2010). Critical Infrastructures and Services. [online] Available at: https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services [Accessed 15 May 2019].

[2] Amazon.com. (2019). Creating IAM Policies - AWS Identity and Access Management. [online] Available at: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_create.html [Accessed 17 May 2019].

[3] Github.io. (2019). mhn. [online] Available at: http://threatstream.github.io/mhn/ [Accessed 17 May 2019].

[4] DiGiacomo, J. (2017). *Active vs Passive Cyber Attacks Explained | Revision Legal*. [online] Revision Legal. Available at: https://revisionlegal.com/cyber-security/active-passive-cyber-attacks-explained/ [Accessed 2 May 2019].

[5] Uma, M. and Padmavathi, G. (2013). A Survey on Various Cyber Attacks and Their Classification. *International Journal of Network Security*, [online] 15(5), pp.390–396. Available at: https://pdfs.semanticscholar.org/ba7b/234738e80b027240e9bfd837bfba61c13e17.pdf.

[6] Cowan, C., Pu, C., Maier, D., Walpole, J., Beattie, S., Grier, A., Wagle, P. and Zhang, Q. (1998). *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*. [online] Available at: https://www.usenix.org/legacy/publications/library/proceedings/sec98/full_papers/cowan/cowan.pdf [Accessed 2 May 2019]

[7] Ieee.org. (2019). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=601338 [Accessed 2 May 2019].

[8] Defined Term - A dictionary of defined terms for the legal profession. (2013). *Passive Wiretapping*. [online] Available at: https://definedterm.com/passive_wiretapping/140598 [Accessed 2 May 2019].

[9] Al-Hammadi, Y. and Aickelin, U. (2008). Detecting Bots Based on Keylogging Activities. *SSRN Electronic Journal*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/1002/1002.1200.pdf [Accessed 7 May 2019].

[10] Bhardwaj, M. and Singh, G.P. (2011). Types of Hacking Attack and their Counter Measure. *International Journal of Educational Planning & Administration*, [online] 1(1), pp.43–53. Available at: https://www.ripublication.com/ijepa/ijepav1n1_7.pdf [Accessed 7 May 2019].

[11] Stallings, W., Brown, L., Bauer, M., Howard, M., Columbus, B., New, I., San, Y., Upper, F., River, S., Cape, A., Dubai, T., Madrid, L., Munich, M., Montreal, P., Delhi, T., Sao, M., Sydney, P., Kong, H., Singapore, S. and Tokyo, T. (n.d.). *COMPUTER SECURITY PRINCIPLES AND PRACTICE Second Edition*. [online] Available at: https://412dnet.files.wordpress.com/2016/01/computer-security-principles-and-practice-2nd-edition.pdf [Accessed 7 May 2019].

[12] FireEye. (2013). *How Cyber Attacks Compromise Your Network | FireEye*. [online] Available at: https://www.fireeye.com/current-threats/how-cyber-attackers-get-in.html [Accessed 7 May 2019].

[13] Spitzner, L. and Wesley, A. (2003). Examples Honeypots: Tracking Hackers. [online] Available at: http://www.it-docs.net/ddata/792.pdf.

[14] Kreibich, C. and Crowcroft, J. (n.d.). Honeycomb -Creating Intrusion Detection Signatures Using Honeypots. [online] Available at: http://www.icir.org/christian/publications/honeycomb-hotnetsII.pdf.

[15] Mokube, I. and Adams, M. (n.d.). Honeypots: Concepts, Approaches, and Challenges. [online] Available at: http://www.cs.potsdam.edu/faculty/laddbc/Teaching/Ethics/StudentPapers/2007Mokube-Honeypots.pdf [Accessed 9 May 2019].

[16] Honeynet.org. (2019). Know Your Enemy: Honeynets. [online] Available at: http://old.honeynet.org/papers/honeynet/ [Accessed 10 May 2019].

[17] Europa.eu. (2010). Critical Infrastructures and Services. [online] Available at: https://www.enisa.europa.eu/topics/critical-information-infrastructures-and-services [Accessed 15 May 2019].

[18] Amazon Web Services, Inc. (2019). What is Docker? | AWS. [online] Available at: https://aws.amazon.com/docker/ [Accessed 24 Apr. 2019].

[19] Sarkar, C., Nambi S. N., A.U., Prasad, R.V., Rahim, A., Neisse, R. and Baldini, G. (2015). DIAT: A Scalable Distributed Architecture for IoT. *IEEE Internet of Things Journal*, [online] 2(3), pp.230–239. Available at: https://www.researchgate.net/profile/Chayan_Sarkar/publication/270567077_DIAT_A_Scal

able_Distributed_Architecture_for_IoT/links/54ad493f0cf2213c5fe39bfe.pdf [Accessed 19 May 2019].

[20] Ieee.org. (2014). *IoT Security: Ongoing Challenges and Research Opportunities - IEEE Conference Publication*. [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6978614 [Accessed 18 May 2019].

[21] Khan, R., Khan, S.U., Zaheer, R. and Khan, S. (2012). Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. *2012 10th International Conference on Frontiers of Information Technology*. [online] Available at: https://pure.qub.ac.uk/portal/files/81384964/PID2566391.pdf [Accessed 19 May 2019].

[22] Github.io. (2017). T-Pot 17.10 - Multi-Honeypot Platform rEvolution. [online] Available at: http://dtag-dev-sec.github.io/mediator/feature/2017/11/07/t-pot-17.10.html [Accessed 17 May 2019].

[23] Amazon Web Services, Inc. (2019). What is AWS? - Amazon Web Services. [online] Available at: https://aws.amazon.com/what-is-aws/ [Accessed 22 Apr. 2019].

[24] Amazon.com. (2019). What Is Amazon EC2? - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html [Accessed 22 Apr. 2019].

[25] Amazon.com. (2019). Setting Up with Amazon EC2 - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html [Accessed 24 Apr. 2019].

[26] Amazon.com. (2013). Virtual Private Clouds - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-vpc.html [Accessed 24 Apr. 2019].

[27] Amazon.com. (2013). What Is Amazon VPC? - Amazon Virtual Private Cloud. [online] Available at: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html#what-is-connectivity [Accessed 24 Apr. 2019].

[28] Amazon.com. (2019). Instances and AMIs - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html [Accessed 24 Apr. 2019].

[29] Amazon.com. (2017). Instance Types - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html [Accessed 24 Apr. 2019].

[30] Amazon.com. (2019). Regions and Availability Zones - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html [Accessed 24 Apr. 2019].

[31] Amazon.com. (2019). Amazon EC2 Key Pairs - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html [Accessed 24 Apr. 2019].

[32] Amazon.com. (2019). Amazon EC2 Security Groups for Linux Instances - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html [Accessed 24 Apr. 2019].

[33] Amazon.com. (2019). VPC Flow Logs - Amazon Virtual Private Cloud. [online] Available at: https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html [Accessed 24 Apr. 2019].

[34] domainbigdata.com (2015). DomainBigData.com - Online investigation tools. [online] Domainbigdata.com. Available at: https://domainbigdata.com/ [Accessed 26 Apr. 2019].

[35] Iana.org. (2017). Protocol Numbers. [online] Available at: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml [Accessed 26 Apr. 2019].

[36] Amazon.com. (2019). What Is Amazon S3? - Amazon Simple Storage Service. [online] Available at: https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html [Accessed 26 Apr. 2019].

[37] Amazon Web Services, Inc. (2018). Amazon CloudWatch - Application and Infrastructure Monitoring. [online] Available at: https://aws.amazon.com/cloudwatch/ [Accessed 26 Apr. 2019].

[38] Amazon Web Services, Inc. (2012). AWS IAM FAQs. [online] Available at: https://aws.amazon.com/iam/faqs/ [Accessed 18 May 2019].

[39] Docker Documentation. (2019). [online] Available at: https://docs.docker.com/engine/reference/builder/ [Accessed 24 Apr. 2019].

[40] Docker Documentation, 25 Apr. 2019, docs.docker.com/engine/reference/builder/. Accessed 1 May 2019.

[41] “About Images, Containers, and Storage Drivers.” Docker Documentation, 28 Feb. 2019, docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/. Accessed 1 May 2019.

[42] “What Is a Container? | Docker.” Docker, 2013, www.docker.com/resources/what-container. Accessed 1 May 2019.

[43] Docker Documentation. (2019). [online] Available at: https://docs.docker.com/engine/reference/run/ [Accessed 25 Apr. 2019].

[44] “Best Practices for Writing Dockerfiles.” Docker Documentation, 25 Apr. 2019, docs.docker.com/develop/develop-images/dockerfile_best-practices/. Accessed 28 Apr. 2019.

[45] Python.org. (2019). Welcome to Python.org. [online] Available at: https://www.Python.org/ [Accessed 25 Apr. 2019].

[46] Docs.Python.org. (2019). 20.17. SocketServer — A framework for network servers — Python 2.7.16 documentation. [online] Available at: https://docs.Python.org/2/library/socketserver.html [Accessed 25 Apr. 2019].

[47] “Download Python.” Python.Org, Python.org, 2019, www.Python.org/downloads/. Accessed 1 May 2019.

[48] “SocketServer – Creating network servers. - Python Module of the Week,” Pymotw.com, 2019. [Online]. Available: https://pymotw.com/2/SocketServer/. [Accessed: 01-May-2019].

[49] “TCP/IP Overview.” Cisco, 2019, www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html#tcp. Accessed 28 Apr. 2019.

[50] Don, Cher. “An Introduction to HTTP: Exploring Telecommunication in Computer Systems.” FreeCodeCamp.Org, freeCodeCamp.org, 10 Sept. 2018, medium.freecodecamp.org/an-introduction-to-http-understanding-the-open-systems-interconnection-model-9dd06233d30e. Accessed 29 Apr. 2019.

[51] https://community.cisco.com/t5/user/viewprofilepage/user-id/167065. “UDP.” Cisco.Com, Mar. 2019, community.cisco.com/t5/networking-documents/udp/ta-p/3114870. Accessed 29 Apr. 2019.

[52] “Communication Networks/TCP and UDP Protocols - Wikibooks, Open Books for an Open World.” Wikibooks.Org, 2017, en.wikibooks.org/wiki/Communication_Networks/TCP_and_UDP_Protocols. Accessed 29 Apr. 2019.

[53] tutorialspoint.com, “Python 3 Multithreaded Programming,” www.tutorialspoint.com, 2016. [Online]. Available: https://www.tutorialspoint.com/Python3/Python_multithreading.htm. [Accessed: 01-May-2019].

[54] “Python Advanced: Threads and Threading,” Python-course.eu, 2011. [Online]. Available: https://www.Python-course.eu/threads.php. [Accessed: 09-May-2019].

[55] Gnu.org. (2019). Overview of the GNU System- GNU Project - Free Software Foundation. [online] Available at: https://www.gnu.org/gnu/gnu-history.en.html [Accessed 25 Apr. 2019].

[56] Shotts, William. “Writing Shell Scripts - Lesson 1: Writing Your First Script and Getting It to Work.” Linuxcommand.Org, 2019, linuxcommand.org/lc3_wss0010.php. Accessed 3 May 2019.

[57] “Sudo in a Nutshell,” Sudo.ws, 2019. [Online]. Available: https://www.sudo.ws/intro.html. [Accessed: 03-May-2019].

[58] “Nmap.” Nmap.Org, 2017, nmap.org/. Accessed 26 Apr. 2019.

[59] Nmap.org. (2019). Nmap: the Network Mapper - Free Security Scanner. [online] Available at: https://nmap.org/ [Accessed 26 Apr. 2019].

[60] “Chapter 15. Nmap Reference Guide | Nmap Network Scanning.” Nmap.Org, 2019, nmap.org/book/man.html. Accessed 26 Apr. 2019.

[61] Pallets. (2019). *Werkzeug*. [online] Available at: https://palletsprojects.com/p/werkzeug/ [Accessed 22 Apr. 2019].

[62] Jinja.pocoo.org. (2019). *Welcome to Jinja2 — Jinja2 Documentation (2.10)*. [online] Available at: http://jinja.pocoo.org/docs/2.10/ [Accessed 22 Apr. 2019].

[63] Flask.pocoo.org. (2019). *Command Line Interface — Flask 1.0.2 documentation*. [online] Available at: http://flask.pocoo.org/docs/1.0/cli/ [Accessed 22 Apr. 2019].

[64] Grinberg, M. (2014). *Flask Web Development, 2nd Edition*.

[65] Dwyer, G. (2017). *Flask vs. Django: Why Flask Might Be Better*. [online] Codementor.io. Available at: https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v [Accessed 24 Apr. 2019].

[66] Fullstackpython.com. (2019). *Flask*. [online] Available at: https://www.fullstackpython.com/flask.html [Accessed 22 Apr. 2019].

[67] Pocoo.org. (2010). *Design Decisions in Flask — Flask 1.0.2 documentation*. [online] Available at: http://flask.pocoo.org/docs/1.0/design/ [Accessed 16 May 2019].

[68] “Banner NSE Script.” Nmap.Org, 2010, nmap.org/nsedoc/scripts/banner.html. Accessed 26 Apr. 2019.

[69] “The cat Command,” Linfo.org, 2019. [Online]. Available: http://www.linfo.org/cat.html. [Accessed: 09-May-2019].

[70] “grep(1) - Linux manual page,” Man7.org, 2018. [Online]. Available: http://man7.org/linux/man-pages/man1/grep.1.html. [Accessed: 09-May-2019].

[71] “The GNU Awk User’s Guide,” Gnu.org, 2009. [Online]. Available: https://www.gnu.org/software/gawk/manual/gawk.html. [Accessed: 09-May-2019].

[72] “Introduction to Named Pipes | Linux Journal,” Linuxjournal.com, 2019. [Online]. Available: https://www.linuxjournal.com/article/2156. [Accessed: 09-May-2019].

[73] “The GNU Netcat -- Official homepage,” Sourceforge.net, 2019. [Online]. Available: http://netcat.sourceforge.net/. [Accessed: 09-May-2019].

[74] “Understanding Python’s ‘with’ statement,” Effbot.org, 2019. [Online]. Available: https://effbot.org/zone/Python-with-statement.htm. [Accessed: 10-May-2019].

[75] Singh, P. (2018). How to Create an AWS EC2 Instance with Python. [online] Ipswitch.com. Available at: https://blog.ipswitch.com/how-to-create-an-ec2-instance-with-python [Accessed 26 Apr. 2019].

[76] Amazonaws.com. (2019). Boto 3 Documentation — Boto 3 Docs 1.9.136 documentation. [online] Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html [Accessed 26 Apr. 2019].

[77] Amazon.com. (2019). What Is the AWS Command Line Interface? - AWS Command Line Interface. [online] Available at: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html [Accessed 26 Apr. 2019].

[78] Amazon.com. (2019). Amazon Machine Images (AMI) - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html [Accessed 28 Apr. 2019].

[79] Amazonaws.com. (2019). Resources — Boto 3 Docs 1.9.137 documentation. [online] Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/resources.html [Accessed 27 Apr. 2019].

[80] Python.org. (2019). *Installing Python Modules — Python 3.7.3 documentation*. [online] Available at: https://docs.python.org/3/installing/index.html [Accessed 1 May 2019].

[81] Grinberg, M. (2018). *The Flask Mega-Tutorial Part XI: Facelift - miguelgrinberg.com*. [online] Miguelgrinberg.com. Available at: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-xi-facelift [Accessed 15 May 2019].

[82] Aggarwal, S. (2014). *Flask Framework Cookbook*.

[83] Pocoo.org. (2010). *Templates — Flask 1.0.2 documentation*. [online] Available at: http://flask.pocoo.org/docs/1.0/tutorial/templates/ [Accessed 15 May 2019].

[84] Techopedia.com. (2019). *What is Query? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/5736/query [Accessed 15 May 2019].

[85] Maia, I. (2015). *Building web applications with Flask*.