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

School of Computer Science and Statistics

Cloud-based network telescope for Internet background radiation collection

Joseph O'Hara

Supervisor: Dr. Stefan Weber

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Master in Computer Science (MCS)

Submitted to the University of Dublin, Trinity College, April, 2019

Declaration

I, Joseph O'Hara, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed: _____ Date: _____

Summary

Botnets are collections of individual computers that have been taken over by an adversary in order to assemble a number of devices that can be controlled to cause disruption to services in the Internet. Distribution mechanisms for botnets scan IP address ranges of the Internet in order to find vulnerable computers that can be infected and added to existing networks. Researchers monitor blocks of IP addresses to detect scanning activities and other abnormal activities in the Internet; collectively referred to as Internet Background Radiation. A tool such as a network telescope, is used to monitor unused IP address ranges that are not hosting services and are not expected to receive legitimate network traffic.

This research proposes a novel network telescope design that is based on a diverse pool of IP addresses controlled by cloud computing providers. In contrast, traditional network telescope deployments that make use of a homogeneous, compact range of IP addresses, a diverse set of IP addresses offers the advantage that the assumed 'geographical' location of the IP addresses can be spread around the world. Also, in contrast to a block of IP addresses that may be identified as a trap and avoided by an attacker, a diverse set of IP addresses is more difficult to distinguish and avoid by possible attackers.

The data collected from this novel system was compared against data collected from a larger traditional network telescope monitoring a contiguous region of IP addresses for the same period. This research focused on the ability of the systems to detect the presence of a particular botnet. While it has been known that both larger and more diverse IP address ranges improve the performance of network telescopes, this research finds that the diversity of IP addresses is significantly more important.

Acknowledgements

Thank you to Eoin Kenny from HEAnet, without your generous help this research would not have been possible.

I would like to thank Stefan Weber for supervising this project. Thank you to Stephen Farrell for your input at different stages of the project.

Measure what can be measured,
and make measurable what
cannot be measured.

- GALILEO

Abstract

Botnets are collections of individual computers that have been taken over by an adversary in order to assemble a number of devices that can be controlled to cause disruption to services in the Internet. Distribution mechanisms for botnets scan IP address ranges of the Internet in order to find vulnerable computers that can be infected and added to existing networks. Researchers monitor blocks of IP addresses to detect scanning activities and other abnormal activities in the Internet; collectively referred to as Internet Background Radiation. A tool such as a network telescope, is used to monitor unused IP address ranges that are not hosting services and are not expected to receive legitimate network traffic.

This research proposes a novel network telescope design that is based on a diverse pool of IP addresses controlled by cloud computing providers. In contrast, traditional network telescope deployments that make use of a homogeneous, compact range of IP addresses, a diverse set of IP addresses offers the advantage that the assumed 'geographical' location of the IP addresses can be spread around the world. Also, in contrast to a block of IP addresses that may be identified as a trap and avoided by an attacker, a diverse set of IP addresses is more difficult to distinguish and avoid by possible attackers.

The data collected from this novel system was compared against data collected from a larger traditional network telescope monitoring a contiguous region of IP addresses for the same period. This research focused on the ability of the systems to detect the presence of a particular botnet. While it has been known that both larger and more diverse IP address ranges improve the performance of network telescopes, this research finds that the diversity of IP addresses is significantly more important.

Contents

List of Figures

List of Tables

1 Introduction

Imagine being handed the keys to your new home and before you can even boil the kettle and make a cup of tea someone is standing outside trying to kick the door down while another person shoves as much junk mail as they can fit into the letter box. You might second-guess your decision to live in this area. This is analogous to the behaviour experienced by Internet users every day. Other hosts are constantly scanning the entire Internet looking for new victims for their latest exploit or perhaps even someone who has left their door wide open. Except on the Internet there is no nice neighbourhood in which to hide from this activity. This unsolicited and often malicious traffic is characterised by the term Internet Background Radiation (IBR). It is a fact of being connected the Internet that is unavoidable.

IBR is typically measured using a tool called a Network Telescope. A tool so named for its suitable comparison to a traditional telescope for the way in which packets land on it such as how photons land on the aperture of a telescope. Analysing this traffic can provide many valuable insights for security professionals and researchers on a wide variety of network security events including infection of hosts by Internet worms, network scanning and denial-of-service (DoS) attacks.

This research will compare the ability of a traditional large network telescope and a smaller distributed cloud-based network telescope to detect network events caused by a popular Internet worm.

1.1 Motivation

IP addresses in the IPv4 address space are scarce. Many organisations today do not have spare IP addresses available for research. There are however some organisations who, as a legacy of how IP addresses were previously distributed still maintain large swathes of IP address space. Most large network telescopes are operated by a small number of such organisations. While there are projects enabling access to these data-sets, such as the Center for Applied Internet Data Analysis (CAIDA)[1] access is limited to certain countries and the process for obtaining access can be lengthy. Distributed network telescopes have been

identified as a potential solution to some of the pitfalls of classic network telescopes[2].

The technical infrastructure of small to large companies has also changed. Companies no longer undertake the large capital expenditure projects of purchasing and maintaining their own data centres. They instead opt to use the services of a cloud provider, who can offer them far greater scalability and economies of scale. This research aims to enable more researchers to collect data through the use of smaller scale systems which can be quickly deployed, as well as to suit this new deployment paradigm.

The novelty of the approach used in this research is the leveraging of a cloud provider's diverse IP address range to enjoy the benefits of a distributed cloud-based network telescope without the need for the construction of a distributed system.

1.2 Objectives

This research aims to determine if a small cloud-based network telescope with a very diverse IP address range can perform better at the task of botnet analysis than a traditional and larger network telescope. In order to narrow the scope of analysis this research will focus on the analysis of Internet background radiation caused by one popular botnet.

The primary objective of this research is to design and deploy a novel cloud-based approach to a network telescope. The successful deployment of this system will enable the collection of data in order to:

- Compare number of botnet hosts seen by a large telescope and the proposed telescope
- Number of scans from botnet hosts
- Overlap in botnet hosts seen by each system
- Compare traffic patterns observed by the two systems

The secondary objective of this research is to operate and collect data from a larger network telescope provided by HEAnet [1], Ireland's National Education and Research Network. This second data set will be used to evaluate the performance of the smaller proposed system.

1.3 Overview

Chapter 2 - Background of this dissertation will give an overview of the various causes of Internet background radiation and the different tools and methods used for its analysis. This

[1] http://www.heanet.ie

chapter will also look at similar and related projects in the area.

Chapter 3 - Design will detail the design of the proposed novel system. It will also detail the operation of a traditional network telescope. Alternative designs will also be explored in this chapter.

Chapter 4 - Implementation will discuss the implementation of the system along with tools and scripts developed for its deployment.

Chapter 5 - Analysis will discuss the tools and techniques used to analyse the collected data.

Chapter 6 - Evaluation will present the results obtained in this research along with an evaluation of the results in relation to the background and state of the art.

Chapter 7 - Conclusion will conclude this paper with a summary of the findings.

2 Background

This section will give an overview of the causes of Internet Background Radiation and the methods used to measure it. It will also describe other systems in use today for monitoring background radiation.

2.1 Internet Background Radiation

Internet Background Radiation is unsolicited and unproductive Internet traffic. Unsolicited and unwanted network traffic has long existed on the Internet[3]. In 2004 Pang et al.[4] first introduced the term 'background radiation' along with a characterisation of the sources of the traffic. Although many years have passed and the general nature of traffic on the internet has changed[5] since their seminal paper, the sources of background radiation can still be characterised as 'fundamentally nonproductive traffic, either malicious or benign'[4].

2.2 Sources of Background Radiation

There are two main sources of malicious background radiation worms and DDoS attacks. Misconfiguration is the main benign source of background radiation.

2.2.1 Worms

A worm is a program that can run by itself and can propagate a fully working version of itself to other machines[6]. Worms use scanning techniques (Section 2.4) to discover other vulnerable victims to propagate to. The traffic caused by this scanning is classified as background radiation. The threat landscape has continued to evolve since Pang et al. identified worms as a source of Internet background radiation. Worms have become more complex and now typically take the form of botnets (See Section 2.5 & 2.7).

2.2.2 DDoS attacks & Internet Backscatter

A common technique used in Distributed Denial of Service (DDoS) attacks is for an attacker to send many connection requests to a host. The host has no way of determining the legitimacy of such a request and so allocates some space in memory for the connection and responds. Eventually the victim will run out of memory and will no longer be able to accept legitimate connection requests. The attacker can forge this connection request to have a different or random source address than their actual address as they are not interested in actually establishing a connection with the victim. This allows them to conceal their identity and prevent the victim from blocking their address.

This behaviour results in hosts receiving unsolicited messages from hosts who are under attack. This type of traffic is referred to as backscatter. The data can be classified by the receipt of TCP SYN+ACK, RST, RST+ACK, and ACK packets[5].

2.2.3 Misconfiguration

Misconfiguration is a benign form of background radiation caused by either hardware or software errors. It accounts for the majority of the background radiation seen and is continuing to grow[5]. Wustrow et al. propose that the growth of misconfiguration could be linked to the general growth of the Internet along with the emergence of new services which can be misconfigured.

2.3 CIDR Notation

Classless Inter-Domain Routing (CIDR) notation will be used throughout this research document. The CIDR notation was introduced in RFC4632[7] to address issues caused by the growth of the Internet and the previous classful system.

A full explanation of the CIDR notation is outside the scope of this research document. The CIDR notation consists of an IP address and its associated routing prefix. The notation is made up of an IP address followed by a slash (/) and a number eg: *192.168.0.0/24* . This number indicates the number of leading 1 bits in the subnet mask. The larger the number the bigger the mask and thus more specific the route.

Of importance to this work is the size of the network and number of available addresses indicated by the CIDR notation. Table 2.1 contains some examples of CIDR blocks along with the number of available hosts in the block.

Suffix	**IP Addresses**
/8	16,777,216
/16	66,560
/24	256
/32	1

Table 2.1: Example CIDR suffixes and number of IP addresses

2.4 Internet scanning

Scanning has been a fact of life on the Internet for many years[8]. Scanning allows an observer to determine whether or not a host responds on a given port and the data they return. This can indicate the services running on the system as most services have a well defined port number[9]. Both attackers[10] and researchers[11] engage in scanning activities. The nature of scanning has changed over time[10]. With the release of the ZMap tool a single machine is capable of scanning for a given open port across the entire public IPv4 address space in under 45 minutes[11]. Previously scanning the entire IPv4 address space required considerable resources and could take up to 100 days. For this reason attackers used to rely more heavily on the use of botnets for Internet-wide scanning, now attackers are more likely to make use of so-called 'bullet proof' hosting services[10].

These tools allow researchers to regularly analyse the use of software and the adoption of new protocols on the Internet. They also allow attackers to scan for vulnerable targets within

minutes of the release of a vulnerability.

Web based search engines such as Censys[1] and Shodan[2] allow users to search indexed scans of the Internet for hosts running specific services. Best practices[10] dictate that the operator of a scanning service should maintain a website at the address they are scanning from providing more information about the scanning and a means of opting out. However, most scanning activity originates from malicious sources[10].

2.5 Botnets

A botnet refers to a collection of 'bots' or victim machines that have been compromised and are now under the command and control of a central controller or botmaster. The botmaster can issue commands to the victim machines. Botnets are used by cyber-criminals to carry out nefarious tasks, such as sending spam e-mail, denial of service attacks, or stealing personal data such as mail accounts or bank credentials[12]. Denial of service attacks 'as a service' also referred to as 'booting' services have become a very lucrative business in recent years with many operators making large sums of money annually[13]. This market has evolved into a subscription based service where for as little as $5 per month a user with very little technical knowledge can launch as many attacks as they want[14]. This market mostly consists of gamers 'booting' or kicking their opponents or gaming related websites offline, mostly in the realm of the popular game Minecraft[15]. These attacks are most successful against small Internet operators with weak or little DDoS mitigation[14].

It has become easier than ever to maintain and grow a botnet, with the emergence of so-called bullet-proof hosting services (BPHS). According to Goncharov, 'Without BPHS, many, if not all major cybercriminal groups would cease to operate'[16]. BPHS provide many of the benefits of cloud computing to cyber-criminals such as reliability and managed hardware. They have a key benefit of being located in regions where law-enforcement can't or won't react to abuse complaints. A BPHS allows a botnet operator to ensure they will be able to maintain communication with their victims over an extended period of time. Some botnet operators will opt to use a Domain Name Service (DNS) to refer to their command and control infrastructure instead, this allows them to more easily move their infrastructure by updating the IP address in their DNS records. However, this introduces a new point of failure for the botnet operator as the DNS infrastructure can be manipulated by good actors to hijack a botnet in order to disrupt or analyse its behaviour[12].

[1] https://censys.io
[2] https://www.shodan.io/

2.6 The Internet of Things

Internet of Things (IoT) devices are becoming ubiquitous in the everyday environment. The term Internet of Things was coined by Kevin Ashton in 1999 in reference to supply chain management[17]. The term is now synonymous with everyday objects, relating to applications including computers, sensors, people, actuators, refrigerators, TVs, vehicles, mobile phones, clothes, food, medicines, books, passports, luggage, etc[18]. Most forecasters agree that the impact of IoT on markets and economic impact will increase year on year[19]. With the spread of IoT into all facades of life, it has become clear to standards authorities as far back as 2010 that the future of IoT depends on good security[20].

2.7 The Mirai Botnet - an Internet of Things Botnet

IoT devices have been at the centre of many large scale cyber security incidents in recent years including a series of high profile Distributed Denial of Service (DDoS) attacks. These attacks were performed by a botnet called "Mirai" and its variants. This strain of malware was first identified by researchers in August of 2016[21]. The original Mirai malware propagated through the use of brute force attacks against random hosts on the Internet. Following the publication of the source code for the Mirai botnet by its author on a popular web forum over a dozen variants began to appear with a variety of different propagation techniques[22]. The source code can also be found on GitHub.com[3].

During September 2016 the blog of security expert and journalist Brian Krebs was hit with a series of DDoS attacks, the peak of which was 620Gbps of traffic[23]. At about the same time an attack against the French web host OVH broke the then record for the largest recorded DDoS attack in the range of 1.1 Tbps to 1.5 Tbps[24]. Researchers at the time prophesied that this scale of attack will only become more common in the future[25]. This record has since been surpassed by a different method of attack - a reflection attack of 1.35 Tbps[26].

Many of the attacks in use today against IoT devices are of a trivial nature. Security researchers have long warned against the use of default credentials and weak encryption techniques. Despite this, some IoT manufacturers appear to be in a race to the bottom to release IoT devices at the lowest cost and highest profit. While not all manufacturers are cutting such corners, with such large estimates of the number of IoT devices it only requires a small proportion of vulnerable devices to create a sizeable botnet. For example the Mirai botnet amassed 65,000 IoT devices in its first 20 hours and grew to a peak total size of

[3] `https://github.com/jgamblin/Mirai-Source-Code`

600,000 devices[22].

2.7.1 Behaviour

As mentioned, the source code for the Mirai botnet has been released publicly. This allows researchers and other interested parties to view and analyse the source code.

Propagation Behaviour

The propagation behaviour of the Mirai Bot is as follows:

- Generate a random IP address not in the blacklist
- Attempt connection on TCP ports 23 and 2323 (Telnet) and TCP port 22 (SSH)
- Attempt authentication using hardcoded credentials list
- Report successful authentication attempts to reporting server
- Loader on remote server infects new bot

The source code included 46 hardcoded username and password combinations which were mostly default credentials for systems. The manufacturers of many devices could be deduced from these combinations[22]

The selection of IP addresses to scan is based on a pseudo-random number generator (PRNG). A detailed statistical analysis of the randomness of Mirai's PRNG was undertaken by Riegel in[27] and found the random number generation technique to be 'rather good, statistically speaking' but very poor cryptographically. The generated IP address is only accepted if it does not come from a blacklist of networks. This blacklist contains networks such as the US Department of Defence, the US Postal Service as well as IP addresses defined as reserved or special use.

Default Behaviour

While scanning the bot simultaneously awaits commands to perform DDoS attacks from the command and control server. The Mirai botnet can perform different types of DDoS attacks including TCP state exhaustion, TCP flooding, UDP flooding, HTTP flooding and some bespoke flooding techniques for use against specific gaming services.

2.7.2 Fingerprinting

Fingerprinting is the term used to refer classifying network traffic based on attributes specific to an attack method which distinguish it from other Internet traffic. There is quite a simple fingerprint for the Mirai botnet. The Mirai botnet implements a stateless scanning technique, meaning that it does not need to maintain any state on the scanning system in order to establish whether or not a system has responded to its probe. It achieves this by taking advantage of a sequence flag in the TCP header. This header is normally used for connection integrity and flow control. It is a 32-bit number normally assigned a random value by the initiator of the connection[28]. When responding to a connection request, the recipient increments this value by the number of bytes they received. However, instead of setting this value as expected the author of the Mirai code set this value equal to the IP address of the destination host. Therefore, when the bot receives a packet it can deduce the source based on this property, allowing the scanning tool to use less memory by not maintaining state for a connection the remote host is unlikely to establish.

This property of the scanning technique allows for scans from the Mirai botnet to be fingerprinted. The probability of this fingerprint occurring accidentally is $1/2^{32}$. This fingerprint method outlined by Antonakakis et al. is still in popular use today[4] for detecting the Mirai botnet.

This fingerprinting technique does not distinguish between different variants of the Mirai botnet under control by different operators. For the purposes of this research all variants will be treated as one.

2.7.3 Recent Activity

The Mirai botnet has now outlived the activity of its three creators who were prosecuted but not jailed in 2018[29]. Although first appearing in late 2016, Mirai and its variants are still active in March 2019 with new attack vectors specifically targeting commercial and industrial signage IoT devices.[30].

2.8 Monitoring Internet Background Radiation

Researchers have been interested in monitoring unsolicited network behaviour since its inception. As far back as 1992 in his paper 'There Be Dragons.' Bellovin discusses the motivations and techniques for analysing the behaviour of 'crackers', 'hackers' and 'Goths'.

[4] `https://mirai.badpackets.net/about/`

Bellovin's motivations were to understand and mitigate these threats rather than to identify and prosecute the attacker.

Researchers are still interested in observing the strategies, tools and behaviours of attackers. While some researchers are still interested in observing the attacks against their own networks, others have begun looking at the Internet as a whole to detect patterns and observe large scale attacks. Monitoring Internet Background Radiation allows researchers to detect these large scale attacks, estimate the size of botnets[22] and to monitor the general state of the Internet.

In general, monitoring Internet background radiation involves the collection of unsolicited network traffic. There are two main methods of doing so, active responders and passive listeners. Active methods interact with the attacker and enable them to provide more information about their motives and techniques. Passive methods simply observe and collect data without any interaction or response.

2.8.1 Honeypots

Honeypots are classified as an active form of monitoring Internet background radiation. Honeypots are described as a decoy computer resource whose value lies in being probed, attacked or compromised.[32]. However, such systems were being deployed and developed before the use of the term Honeypot[31]. Honeypots can be deployed for a wide variety of purposes. They may be deployed within a corporate network to detect an intruder inside[33] or to act as a first line of defence much like a canary in a coal mine - signalling that something is wrong before a larger attack can take hold. A Honeypot may also be deployed in an Internet-facing manner in order to collect information about the ways in which attackers are scanning and probing hosts on the Internet.

Their aim is to be attractive to attackers with the goal of collecting as much detailed information as possible. They appear attractive to attackers by closely mimicking the behaviour of a vulnerable system.

They can be used to collect information about later stages of an attack. This follow-on attack data is simply not visible using passive approaches. This data can include credentials or exploits used to break in to systems at the very early stages of an attack. The latter stages of an attack can be also be observed. This might include techniques used to consolidate and maintain control over a system, payloads used to further attack systems, as well as techniques used to elevate privileges on the machine. These collected payloads can be further analysed and compared against malware samples found elsewhere. This collected information in the form of log files or traffic logs is often backed up to a logically separate machine. This helps to protect against an attacker deleting them in an attempt to hide their actions.

A set of Honeypots can also be configured into a network, a so called Honeynet. This can allow researchers to obtain a wider look at attack behaviour. A Honeynet may be deployed in a distributed manner, sometimes as a collaboration between researchers[5]. They can also be used to analyse propagation techniques used by attackers as they attempt to connect from one host to the next.

Honeypots have varying levels of interactivity. There are three levels of interactivity typically used: Low, high and medium. Interactivity refers to the amount of interaction an attacker can have with the system.

Low interaction

Most low-interaction systems emulate services such as FTP (File Transfer Protocol) or a Web server. Depending on the sophistication of the implementation a low interaction Honeypot may not look convincing to attackers and their tools. A low interaction Honeypot does not allow for further system interaction so not much new information can be gained.

High interaction

A high-interaction Honeypot provides access to real services and a real operating system with advanced logging. This opens up the possibility for the attacker to take control of a real system and its associated resources. This also opens up an ethical question around whether or not researchers should be providing resources which could be leveraged by attackers to launch DDoS or spam campaigns. Some Honeypot operators will limit the lifespan of their machine or the outward communication firewall rules in order to mitigate this risk.

Medium interaction

A medium-interaction Honeypot is a mix somewhere between a high-interaction system and a low-interaction system. The degree to which it is either depends on the specific configuration including the services which are enabled and available on the machine.

2.8.2 Network Telescopes

A network telescope is a passive method of analysing Internet background radiation. They have emerged as the main method for analysing security events such as DDoS attacks and worm contagion at Internet scale[2]. Network telescopes get their name from the apt

[5] `https://www.projecthoneypot.org/`

comparison to traditional light telescope, where a photon landing on a light telescope is analogous to a packet arriving on a network telescope[2]. They have also been referred to as network motion sensors[34], network sinks[35], darknets[36] and blackhole monitors[37].

A network telescope monitors a contiguous portion of unused IP address space that is not expecting to receive any legitimate traffic. Therefore any traffic it receives can be classified as unsolicited. The resolution of a network telescope refers to the size of the address space that it monitors. The larger the address space - the higher the resolution. A large network telescope typically monitors an IP address block of size **/8** (16.7M addresses) or **/16** (65K addresses) (See section: 2.3 CIDR Notation).

The resolution of a network telescope has an effect on its ability to observe smaller events as well as accurately determine the start time of events occurring on the Internet[2].

Figure 2.1 shows the traditional architecture of a network telescope. All IP addresses arriving at the router destined for the dark IP address are forwarded to the collection server according to the route table. Network traffic is shown flowing passively in one direction to server.

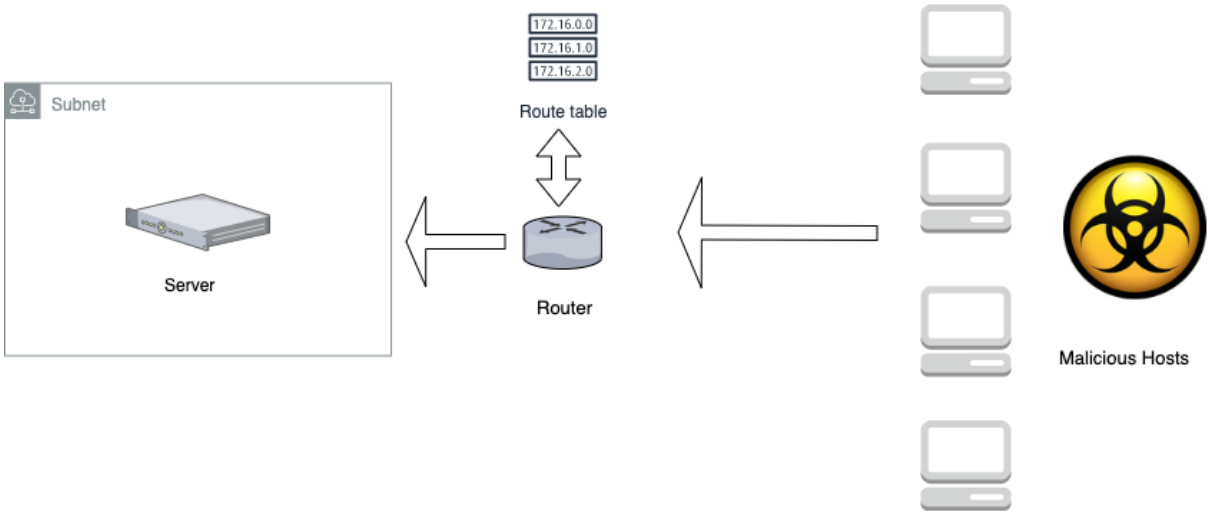

Figure 2.1: Traditional Network Telescope Architecture

Network telescopes are limited in their ability to observe all the stages of an attack. As a network telescope does not respond to any requests and does not engage with any attacks no further information can be gleaned from the attacker. Figure 2.2 below from Cooke et al.[34] shows firstly the behaviour of a Honeypot (Instrumented Live Host), secondly the behaviour of their 'Internet Motion Sensor' which is a more interactive network telescope and thirdly a traditional network telescope.

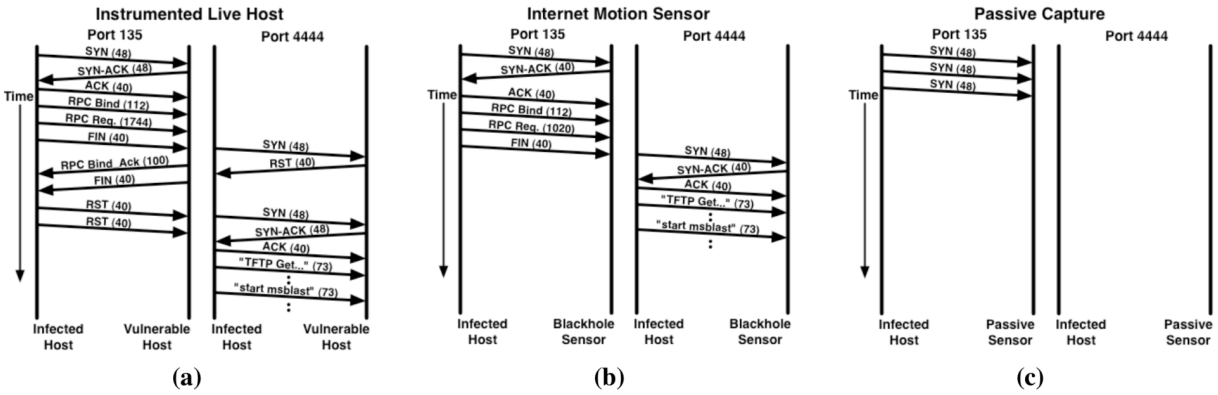

Figure 2.2: Figure 3 from Cooke et al.[34] **The Internet Motion Sensor: A distributed global scoped Internet threat monitoring system**. (a) shows a Honeypot type system. (b) shows the IMS, a slightly interactive network telescope and (c) a standard network telescope. Shows that the IMS has the depth necessary to see and capture all major transactions for the worm.

2.8.3 Greynets

A greynet is similar to a network telescope, but instead of using a contiguous block of IP address space from one dark subnetwork it uses individual or smaller blocks of dark IP addresses from 'lit' subnetworks[36]. Harrop and Armitage also propose the name 'sparse darknet'.

Chindipha et al. and Harrop and Armitage recognise the challenges faced by smaller organisations and researchers in obtaining larger blocks of IP address space for use as network telescopes[38][36].

The experimental greynet system developed by Harrop and Armitage was built on two subnetworks with varying amounts of free IP addresses. It contained 238 dark IP addresses making it approximately a /24 network.

Harrop and Armitage conclude that network operators can obtain useful levels of network scan detection with only the use of a small greynet.

2.8.4 Distributed Network Telescopes

Distributed network telescopes combine multiple smaller network telescopes observing different regions of the network address space. Distributed network telescopes observe significantly different traffic patterns than traditional network telescopes[37]. For this reason Moore et al. suggests the use of distributed network telescopes to overcome some of the issues associated with monitoring a contiguous IP address block. Some of the benefits of increased observed address space include improved detection time, duration precision, and targeting rate estimation over individual telescopes, reveal and overcome targeting bias in the event[2].

Distributed network telescopes also typically take the form of a Greynet(See Section 2.8.3) as the distributed design easily allows for monitoring of portions of IP address space from different subnetworks.

One example of a large distributed network telescope is the commercial and private one operated by Arbor[6]. This system was formally known as The Active Threat Level Analysis System (ATLAS)[39]. Moore et al. discuss some collaborative projects which make use of distributed network telescopes, however only one of these services appears to be still in existence and relies on firewall log submissions from contributors instead of raw network capture data. There is more collaborative activity in the deployment of distributed Honeypots(Section 2.8.1).

[6] https://www.netscout.com/global-threat-intelligence

3 Design

This section will detail the design of the systems related to this research. It will start with an overview of the system used to collect baseline data - the so called traditional network telescope. The design of the novel system will then be described along with some related terminology.

3.1 Design Objectives

The core objective of both systems is to collect network traffic directed at the system and to store it for further processing. The files will also be split in hourly intervals to facilitate easier processing. This baseline requirement means that the systems must have enough storage for an estimated one week worth of data collection. The systems must also have enough RAM in order to store the packets in memory before writing them to disk so as to not lose any data. The user of the system must have the required privileges to read packets.

3.2 Network Telescope

A traditional /16 network telescope was provided by HEAnet [1], Ireland's National Education and Research Network. This involved the creation of a virtual machine within the HEAnet network and the routing of the /16 block to that machine. The machine was configured and deployed by the network operator and provided to the researcher. This /16 address space had been unused for a number of years before this research. Access was provided to the researcher by means of Secure SHell(SSH). This was configured on an IP address outside of the /16 range in order to avoid polluting the collected data.

[1] http://www.heanet.ie

3.2.1 Hardware provided

The hardware specifications required for the machine were estimated by looking at the reported data rates of the CAIDA network telescope[2] for a /8 network telescope. CAIDA state that they receive 3TB of data per day to their system. The /16 network telescope is 256 times smaller than the CAIDA /8. Scaling down by the same factor works out to about 12GB per day of traffic or about 0.5 GB per hour. This lead to a final estimate of an average data rate of 1.1Mbps. The actual recorded data rate was 1.25Mbps. This estimate was used as a rough guide to ensure that the system had the correct magnitude of storage space made available - both in terms of size and write speed.

3.3 Novel Cloud-based System

The aim of building this novel cloud-based system is to deploy a /24 network in order to observe and collect Internet background radiation. This system has some of the properties of a distributed network telescope (Section: 2.8.4) and some of the properties of a greynet (Section: 2.8.3). The proposed system has a diverse range of IP addresses similarly to when a group of organisations collaborate to build a distributed network telescope.

The IP addresses assigned by the cloud provider are within a number of different subnetworks. This gives the system the property of a greynet by having multiple dark and lit IP addresses in multiple subnetworks.

The first concept for the design of this cloud-based system was a globally distributed network telescope. The main purpose for this was to obtain a diverse range of IP addresses. As cloud providers assign IP addresses from a diverse pool of IP addresses in every region it was decided to simplify the design by deploying in only one region while still maintaining the benefits of a diverse IP address range.

3.3.1 Choice of cloud hosting provider

The novel implementation was deployed on Amazon Web Services(AWS), a large and popular cloud provider.The concept for the architecture was arrived at when considering the network functionality provided by AWS. Mainly the ability to assign multiple IP addresses to a single instance.

Two other main cloud computing providers were also considered for use in this project.

[2] https://www.caida.org/data/passive/telescope-near-real-time_dataset.xml

Google Cloud, the Cloud computing service offered by Google was also evaluated. It was deemed not suitable as there is a limit of 100 IP addresses per machine and a limit on 5 ports per IP address. This would not provide the functionality of a network telescope.

The Azure cloud computing service offered by Microsoft was also considered. The Azure service offers better functionality for this task than AWS and also at a lower cost. The network topology would also have been simplified. However, due to the time constraints of this research project it was decided to prioritise the researchers familiarity with AWS over cost.

3.3.2 AWS Terminology

AWS provides a layer of virtualisation which abstracts a lot of the traditional network terminology away from the user. A explanation for some AWS terminology used is below in Table 3.1:

AWS Term	**Meaning**
EC2 Instance	Elastic Compute Cloud. Server. There are many varieties available for different purposes
Elastic IP Address	A static IP address associated with an account that can be mapped to an instance or network interface
Elastic Network Interface	Represents a virtual network card
VPC	Virtual Private Cloud. Logically isolated section of the AWS Cloud

Table 3.1: AWS Terminology

3.3.3 System Architecture

The cloud-based system consists of a single Amazon EC2 instance deployed in a VPC. The maximum number of network interfaces that can be attached to an instance are attached. This number depends on the instance type and specifications [3]. This has an impact on the overall cost of the system as well as the design. The ramifications of these limitations will be discussed in Section 3.3.4: Instance Pricing & Specifications.

A subnet in the VPC is created for each network interface. This avoids issues caused by asymmetric routing which may occur when a machine has multiple network interfaces attached to it. Each network interface is assigned the maximum number of possible private IP addresses[3]. Each private IP address then has an elastic IP addresses associated with it.

This architecture is shown in Figure 3.1. The individual subnets are omitted from this diagram for clarity. This architecture can be contrasted with Figure 2.1 which shows a traditional network telescope architecture.

[3] `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html#AvailableIpPerENI`

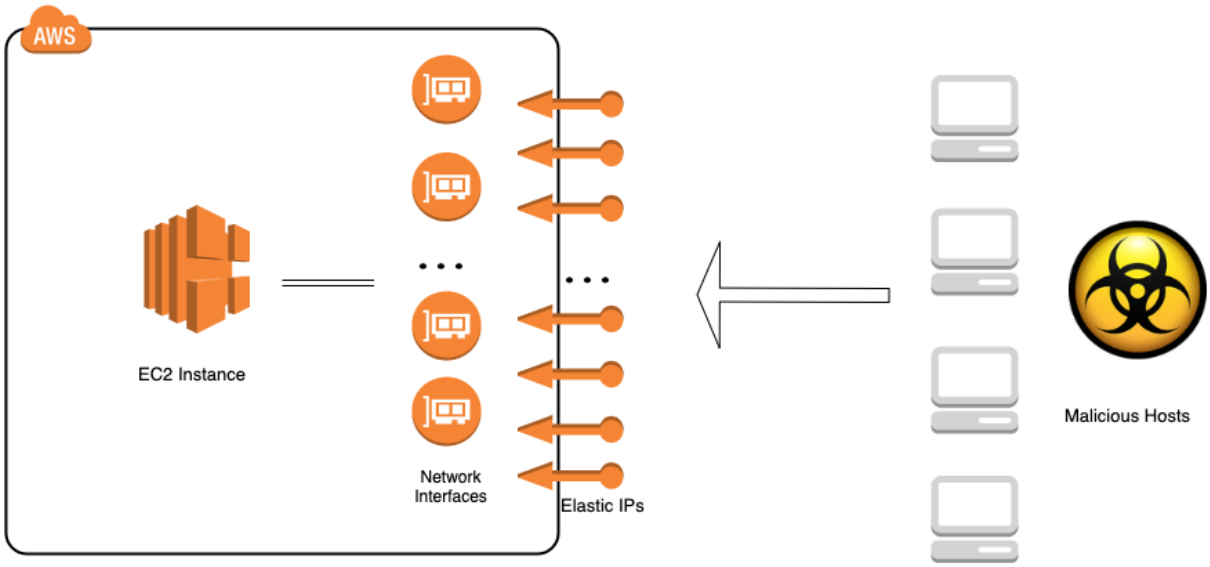

Figure 3.1: Cloud-based Network Telescope Architecture

3.3.4 Instance Pricing & Specifications

As mentioned above there is a limit on how many network interfaces can be assigned to each machine and how many IP addresses can be assigned to each network interface. AWS provide a table of these mappings in their documentation shown below in Figure 3.2.

Instance Type	Maximum Network Interfaces	IPv4 Addresses per Interface
a1.medium	2	4
a1.large	3	10
a1.xlarge	4	15
a1.2xlarge	4	15
a1.4xlarge	8	30
c1.medium	2	6
c1.xlarge	4	15
c3.large	3	10
c3.xlarge	4	15
c3.2xlarge	4	15
c3.4xlarge	8	30

Figure 3.2: Abridged AWS documentation showing maximum network interfaces per instance type

AWS offer over 130 instance types with different configurations and specialisations for different tasks. These instances are all priced at different hourly rates. In order to find the instance type with the best value per IP address a spreadsheet was created, combining the data shown in Figure 3.2 and the pricing information made available elsewhere in the AWS

documentation[4]. An excerpt of that collated data is shown below in Figure 3.3. The cheapest method of obtaining the desired amount of IP addresses would have been the `t3.nano` instance type. However, as can be seen in column H in Figure 3.3, this would have required 64 instances. This would have added significantly to the work required to build and manage the system. Instead the instance type `a1.4xlarge` was chosen. This allowed the system to be created with 240 IP addresses, which is close to the targeted 256. Choosing a smaller number of IP addresses is not an issue as the system is of the correct magnitude. A 238 IP address /24 greynet was used by Harrop and Armitage in their seminal paper on greynets[36].

The `a1.4xlarge` system is running am ARM chip custom designed by Amazon. It is designed to be cost effective at large scale loads. The `a1.4xlarge` system is the highest specification system in the `a1` range. It has 16 vCPUs and 32 GB of memory. This is much more than is required to simply record network traffic. This inefficiency is a big drawback of this design. As discussed in Section 3.3.1 Azure could be used as they do not have a tiered system of network cards per machine type. This wasted CPU time could have also been used for a productive task while also collecting network traffic, such a contributing to the SETI@home project[5] or by mining cryptocurrency.

Other costs

AWS charges users for using additional elastic IP addresses after one per instance. This is a somewhat low fee of $0.005 per hour per IP but quickly adds up when using so many IP addresses. This charge is intended to encourage users to not waste IP addresses and to release them back into the pool. There is no traditional use-case that calls for the behaviour of assigning hundreds of IP addresses to one instance as used in this research.

	A	B	C	D	E	F	G	H
1	Name	Interfaces	Ips Per Interface	Price / Hour	Total IPS	Price Per IP / Hour	Price for 10 Days	No Instances Needed
2	t3.nano	2	2	0.0057	4	0.001425	87.552	64
3	t2.nano	2	2	0.0063	4	0.001575	96.768	64
4	t3.small	3	4	0.0228	12	0.0019	116.736	21.33333333
5	a1.large	3	10	0.0576	30	0.00192	117.9648	8.533333333
6	a1.xlarge	4	15	0.1152	60	0.00192	117.9648	4.266666667
7	**a1.4xlarge**	**8**	**30**	**0.4608**	**240**	**0.00192**	**117.9648**	**1.066666667**
8	t2.small	3	4	0.025	12	0.002083333333	128	21.33333333
9	t3.medium	3	6	0.0456	18	0.002533333333	155.648	14.22222222
10	t3.large	3	12	0.0912	36	0.002533333333	155.648	7.111111111
11	t2.medium	3	6	0.05	18	0.002777777778	170.6666667	14.22222222

Figure 3.3: Excerpt of spreadsheet generated showing cost per IP address per instance type. Highlighted row showing chosen instance type

[4] `https://aws.amazon.com/ec2/pricing/on-demand/`
[5] `https://setiathome.berkeley.edu/`

3.3.5 IP Address Diversity

None of large cloud providers were in operation in the early 1990's when much of the IPv4 address space was being given away to corporations, educational institutions and US government departments without much thought of running out of space. As a result, to cater to their growing customer bases the cloud providers need to acquire more IP address space. They do this in a piecemeal fashion, acquiring new blocks as they come on the market. One large example of this is Amazon's purchase of 3.0.0.0/8 from General Electric. Amazon publish a list of all the IP addresses that they use for AWS[6]. Looking at this list shows the variety of IP addresses available.

The pool of IP addresses obtained from AWS for this research came from 28 different /16 blocks. This offers much more IP address diversity than the one /16 used in the traditional network telescope.

[6] https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html

4 Implementation

This chapter will detail the operation of the network telescope provided by HEAnet and the implementation of the novel cloud-based design. The final costs and some proposed alternative designs will also be discussed.

4.1 Traditional Network Telescope

The traditional `/16` network telescope provided by HEAnet required minimal configuration as it had been configured in collaboration with the network engineers at HEAnet after the design process outlined in Section 3.2.1.

Access was granted to the machine via `SSH` on a separate IP address outside of the `/16` space. `tcpdump` was configured along with file rotation as described in Section 4.3 below.

4.2 Cloud-based System

The novel cloud-based system was initialised through the AWS web console. This created the *a1.4xlarge* system as described in Section 3.3.4. Python scripts were created in order to add the additional 7 subnets & network cards required and to assign the 240 IP addresses. These scripts made use of the official AWS Python SDK, *boto3*[1]. They are described below and also printed in Appendix A1.1

4.2.1 Scripts to setup system

The main script `aws.py` operates as follows:

Create subnets and assign network interfaces:

[1] `https://github.com/boto/boto3`

- Start at subnet `10.0.0.0/24`
- Add subnet if it doesn't exist
- Add network interface if it doesn't exist
- Increment to subnet 10.0.1.0/24
- Repeat until `10.0.7.0/24`

Allocate private IP addresses to interfaces:

- Get all interfaces on the machine
- Get number of IP addresses currently assigned
- Add IPs until the max is added (30)

Allocate public Elastic IP addresses to interfaces:

- Get all interfaces on the machine
- Get all private IP addresses that aren't assigned a public IP
- For each private IP without public allocate & assign a new elastic IP

Release any Elastic IPs that are not assigned to the system to avoid additional charges.

4.2.2 Networking

Adding additional network interfaces and IP addresses requires a lot of configuration. This is especially true when network interfaces are added whilst the machine is turned on. Thankfully AWS provide a customised distribution of Linux called Amazon Linux[2]. This has additional tools installed which can communicate with the Amazon APIs automatically to detect that a network card has been attached and to configure it based on the route tables defined. However, it was hard to tell that this process was occurring as there is a 25 minute delay for the DHCP servers in the VPC to update and assign IP addresses to the new network interfaces. This caused a significant amount of confusion and delay during setup.

The `SSH` service was running on the telescope machine in order to facilitate setup and monitoring. If this service was publicly accessible on all IP addresses port 22 would not behave as a network telescope as attackers could detect the presence of the service and attempt to attack it. To prevent this, the `SSH` service was configured to only listen on one IP address (the primary IP). Access to this port was then further restricted by limiting access to only IP addresses within TCD's IP address range of `134.226.0.0/16`.

[2] `https://aws.amazon.com/amazon-linux-ami/`

The networking of the additional IP addresses was tested by choosing random IP addresses from each subnet from the list of public IP addresses and attempting to connect to them from a remote host using the UNIX tool `netcat`. If the connection attempt was shown in the output of `tcpdump` the system was confirmed to be working.

4.2.3 Actual Costs

Estimates for the cost of the system were given in Section 3.3.4 along with Figure 3.3. The estimated cost of just the server for 10 days was $117.96. This system was actually run for approx. 7.2 days. The final cost of running the `a1.4xlarge` system was $80.11. As mentioned in Section 3.3.4, there is an additional cost for each elastic IP address assigned to a server. This rate was '$0.005 per Additional Elastic IP address attached to a running instance per hour (prorated)'. This resulted in 35,475 'IP address hours' at a cost of $177.38. There was additional charges of $18.27 for data storage but this was mostly related to backing up of data from the larger telescope system. The total cost directly related to the operation of the network telescope for 7 days of data collection was $257.49 .

The high cost of the IP addresses makes the Azure service discussed above in Section 3.3.1 appealing as an alternative provider. The cost per additional IP address with Azure is $0.0031 (€0.0035 @ €1 = $1.12) making it 30% cheaper.

4.3 Traffic collection

In order to collect traffic for analysis the tool `tcpdump` was used[3]. `tcpdump` is installed in most Linux distributions by default and is a popular tool for network traffic collection. The output of `tcpdump` is stored in a `.pacp` file for further analysis.

4.3.1 `tcpdump` Configuration

`tcpdump` was configured with a number of flags in order to improve performance and to manage the data stored. The flags `-G 3600 -s 65535` were used. `-G` indicates after how long the `.pcap` file should be rotated in seconds. The `-s` flag limits the maximum size of each packet that is saved by the tool. For the purposes of this project we are only interested in the initial data and headers received, thus superfluous data can be discarded. The default value for this flag is `262144 bytes`. Reducing this value improves the performance of the tool as less buffer space is taken up reading larger packets and less storage space is used.

[3] `https://www.tcpdump.org/`

The interface to record on was specified using `-i <interface>`. On the large HEAnet system this flag was configured to one specific interface. On the AWS system the argument `any` was used to configure `tcpdump` to listen on all 7 interfaces (as well as system loopback interfaces) simultaneously.

Files were written to a local folder in the format `{system}_YYYY-MM-DD-HH.pcap` for example, `heanet_2019-03-21-11.pcap` or `aws_2019-03-21-11.pcap`. The `-G 3600` flag indicates in seconds that the file should be rotated every hour. This creates a new `.pcap` file in the specified directory.

In order to ensure that the `tcpdump` program continues running in the case of an error or crash a Linux service was created. A service allows a program to run in 'daemon' mode. It is managed by the `systemd` tool, an operating system tool. To create this service a file was placed in the `/etc/systemd/system` directory. This service file along with the flags described above can be seen in Appendix A1.2. The service was enabled using the command `systemctl enable telescope.service` and started using `systemctl start telescope.service`.

4.4 Implementation Issues

The astute reader will have noticed that in Section 4.2.3 above the 35,475 IP address hours for 7.2 days of system time does not add up to 240 IP addresses as designed in Section 3.3.4, but instead adds up to 204 IP addresses. This was in fact due to only 204 IP addresses being assigned to the instance. This issue was not noticed until the end of the collection period before the analysis of the data when collating the public IP addresses.

As per the behaviour detailed in Section 4.2.1 the script successfully assigned all subnets and private IP addresses. This was verified by viewing screenshots taken of the system configuration before termination as well as by reviewing the network capture data. The network logs shows local activity on these subnets throughout the capture period. This local activity was DHCP traffic between the gateway and the router in the subnet. The script failed to assign public IP addresses to 36 private IP addresses, from the 'top' of the `10.0.6.0/24` subnet and the whole of the `10.0.7.0/24` subnet.

This may have been caused by a bug in the script used to assign the elastic IP addresses. This bug may have been related to not catching an error thrown by the AWS SDK. It is possible that the maximum number of elastic IP addresses for the account was reached. This value was set to 256. If the existing number of IP addresses in use before the setup script was ran was greater than or equal to 52 this maximum could have been reached, thus preventing the creation of additional IP addresses. However, this should have been mitigated

by the use of the function which releases any unassigned IP addresses.

Unfortunately, due to budget constraints it was not feasible to run the data collection period again. As discussed in the design phase in Section 3.3.4 when choosing 240 IP addresses over 256 the exact number of IP addresses in the telescope is not a major issue once it is of the correct magnitude.

5 Analysis

5.1 Tools and methods used for analysis

A number of tools and techniques were used to analyse the pcap files. The data collected from both systems was stored on one server in AWS where it was analysed. This chapter will discuss the tools used for the analysis of the captured data as well as other tools that were considered. The default resolution of the output was also increased.

5.1.1 tshark

tshark is a network protocol analyser tool which is very similar to tcpdump, the tool used to collect the network traffic. The tool can be used to capture data from the network or to process existing pcap files. The tool is from a suite of tools called the Wireshark Network Analyser[1]. The tool was used because it has easy to use filtering options. The output of tshark can be easily limited to show only specific fields. This is done with the -T and -e flags. For example `-T fields -e ip.src -e tcp.dstport` would show only the source IP address of the packet along with the destination port of the packet. This can be used to generate lists of IP addresses.

5.1.2 capinfos

capinfos is another tool from the Wireshark suite. It prints a summary of the information about a capture file. That data includes the average data rate, the number of packets received and the total date size. The data can be exported in a computer friendly format using the `-T -m -Q` flags (T for table, m for comma separated, Q for wrapping the data in quotes). It can be used on multiple files at once. Using this method a report for all of the data sets can be generated quickly.

[1] https://wireshark.org

5.1.3 Bash and Python Scripts

A number of bash scripts were created to process the collected data. The scripts are included in Appendix A1.3 and described below.

Scripts to get all IPs

When doing some preliminary analysis of the capture data it was realised that the destination IP address of the packets was the local IP address and not the public IP address. This was due to the OS level packet routing. In order to filter data based on the Mirai fingerprint discussed in Section 2.7.2 the public IP address, the public IP address as a long and private IP address must be known. This filtering process is detailed further in below. Converting the IP address to a long simply means converting it from the familiar dot-decimal representation to a 32-bit integer representation. This was done by using the python sockets API to build a packet and then extracting the calculated 32 bit value.

The script `ip-addr.py` behaves as follows:

- For each network interface:
- Get Private IP address
- Get associated public IP address
- Calculate IP address as a Long
- Output as tab separated file: {priv_ip} {public_ip} {long_ip}

Filtering Mirai probes

As detailed in Section 2.7.2, the Mirai botnet has a distinct fingerprint. This involves TCP sequence number field being equal to the IP address.

In order to filter the Mirai traffic from the other background radiation in the AWS data a number of steps were required:

- Create a `tshark` filter query to select only packets with the fingerprint
- Run the query against each hourly file
- Store filtered data in new files

Creating the `tshark` query string required the use of the script described in Section 5.1.3 to first generate a list of all IP address/long combinations. This file was then processed by another Python script `gen_aws_query.py` which outputted a long query string which could

be used later. The output of this script looked like `(tcp.seq == 315257032 && ip.dst == 10.0.0.87) || (tcp.seq == 315257272 && ip.dst == 10.0.3.125) . . ..` Note that the sequence number is the long representation of the *public* IP address and the destination IP address is the *private* IP address. Just the sequence number could have been used, as discussed in Section 2.7.2 the probability of these values occurring are quite low but the probability of false positives is vastly reduced by specifying both. There is a performance penalty due to this bigger query string. This is reduced by putting the least likely to match portion of the query on the left hand side of the `&&` operand.

This generated filter query was added to another bash file `filter_aws_mirai.sh`. This query was ran against all of the individual `.pcap` files using another file `all_mirai_aws.sh`. This additional script preserves the time-stamping and naming of the files. The flag `-o tcp.relative_sequence_numbers:FALSE` was needed when filtering the data, by default the `tshark` tool treats the sequence numbers as relative values. This means the first relative sequence number is 0, the second 1 and so on. Without this flag no results would be returned.

This process was repeated for the larger telescope capture data. However, the larger system does not require a complex filter string to filter the data. As the larger system uses a contiguous block of IP addresses a simple query string to determine if the sequence number lies in the correct range can be used: `tcp.seq >= FIRST_IP && tcp.seq <= LAST_IP`. As this does not check that the IP address matches the sequence number, some false positives may have occurred. For example, if a packet destined from another IP addresses in the block had a sequence number equal to that of another IP address in the block. It was calculated that of the 11659 Mirai packets received by the larger system, 0.353 of them were false positives. This filtering was done using `filter_heanet_mirai.sh` and `all_mirai_heanet.sh`.

The counts of unique Mirai hosts which probed the systems every hour is generated using the script `count_mirai_hosts.sh`. This script makes use of another script `count_unique_hosts.sh`. This script combines the use of `tshark` along with the the `UNIX` utilities `sort`, `uniq` (unique) and `wc` (word count).

IP Geolocation

IP geolocation is the process of mapping IP addresses to approximate geographical locations. IP address geolocation is not an exact science, a well-known and used database and Python library was used[2]. A Python script `geo-ip.py` was created. This script processes all the IP addresses seen both network telescopes. The data was output in CSV format. The data

[2] `https://github.com/maxmind/GeoIP2-python`

included the IP address, country, latitude, longitude, source. The source is either 0 for the small system, 1 for the large system or 2 for both.

The source country of the infected Mirai hosts was used to count the number of attacks from each country and the approximate latitude and longitude were used to create a map showing the source of attacks.

5.1.4 `datamash`

The tool `datamash`[3] was used to manipulate other files. This program was used to help calculate the most popular port count. One program iterated over each file using `tshark` to output the port number accessed by each connection attempt. This was piped to `sort | uniq -c | sort` which sorts the file, counts the number of time each port number occurs and then sorts again. This output was appended to a file in the format `count port`. This program was used to group data by the first column and sum the second column. `datamash` was then used to group the data by the second column containing the port numbers and sum the first column for each port. This resulted in a final file containing a sorted list of the most popular ports accessed. The command used was: `datamash -W groupby 2 sum 1 < all_ports_counts.txt > reduced_counts.txt`

5.1.5 `QGIS`

The tool `QGIS` was used to create maps of the data. `QGIS` is an open-source tool for creating graphical visualisations. The output from the geolocation described in Section 5.1.3 was used to generate 3 maps. Showing the locations of attacks received from the `/16 system`, the `/24` system and all. `QGIS` was configured to use a Mercator projection with a black background and grey landmasses.

5.1.6 Google Sheets

Google sheets was used as a spread-sheeting and data visualisation tool. It was used to manage and manipulate the data obtained by other tools and scripts and to create graphs visualising the results.

[3] `https://www.gnu.org/software/datamash/`

5.2 Alternative data analysis tools

An alternative method of analysing the results was considered. The tool `moloch`[4] was discovered during the research phase of the project and appeared to have many desirable features. Moloch is an open-source network packet capture and indexing tool. The architecture of the tool is designed for use for ongoing collection and analysis of network traffic for intrusion detection in networks. It is designed to handle large amounts of data, and so, was well placed to handle the large amount of data generated by the `/16` system. The indexing feature allows for searching and generation of reports. This was desirable as it would facilitate more detailed analysis of specific events observed in the data as well as the creation of maps.

The tool requires a machine running an instance of Elasticsearch, a data search and analytics engine. AWS provides Elasticsearch as a service and an instance can be easily created. The data would be indexed and stored in Elasticsearch using a tool in the `moloch` suite. This import process was very slow due to the large size of the data and the insufficient specifications of the Elasticsearch server. The import process failed multiple times due to exhausted system resources, and when restarted duplicates were entered into the database with no easy way of cleaning the data. The Elasticsearch service operated by AWS is expensive and due to budget constraints the specifications of the server could not be increased. It also transpired that the default indexing of the tool did not allow for searching based on the `tcp.seq` flag required to identify traffic from the Mirai botnet. This would have required significant engineering work to alter the indexing used by `moloch`. It was then decided to use the other tools described above in Section 5.1 to perform analysis of the data.

[4] `https://github.com/aol/moloch`

6 Evaluation

Data was collected from the two systems for a period of one week using the methods described in chapter 4. This data was then analysed using the techniques described in chapter 5 in order to achieve the objectives outlined in Section 1.2. This chapter will detail and evaluate the results obtained and discuss their implications.

6.1 Evaluation Metrics

Number of unique Mirai hosts per hour

The two systems were compared based on the number of unique hosts that scanned the system in each hour period. The list of IP addresses that connected to each server were sorted and duplicates were removed. This list was then counted, giving the number of unique hosts for an hour. The novel system is evaluated based on its ability to detect Mirai probes.

Internet background radiation data volume & data rates

The amount and rate of Internet Background Radiation received by the two systems was calculated for both data sets. This is calculated by using the `capinfos` tool described in Section 5.1 to measure the size of all the packets received by both systems for the collection period.

6.2 Results

This section will look at the results obtained in order to evaluate the system as a tool to monitor the Mirai botnet as well as supplemental results obtained related to the general operation of a network telescope.

6.2.1 Expectations

Before analysing and looking at the data it was expected that the larger /16 network telescope would receive approximately 256 times more traffic than the smaller /24 cloud-based system. This was not quite an accurate expectation as Moore et al. show that the performance of different sized network telescopes do not scale linearly, but they state this difference is slight in practice[2]. This difference in behaviour was expected in terms of general traffic load as well with the detection of specific events. It was also expected that the diversity of the IP addresses in the cloud-based system would have some advantages for event observation as discussed in Section 2.8.4.

However, as these results will show this was not the case. The results show that the larger system collects more data in general but not for all types of events.

6.2.2 Data Volume & Data Rates

As expected, the larger system received more Internet background radiation than the smaller system. However, not quite as much as was expected. As per the estimates described in Section 3.2.1 the larger system was expected to receive about 1.1Mbps but actually received 1.25Mbps. By scaling using the same method, one would expect the /24 system to receive about 619 `bytes per second`. The /24 system actually received 1828 `bytes per second` on average. The data rates are shown below in figures 6.1 and 6.2. This shows that the /24 novel cloud-based network telescope receives about 3 times more traffic per IP address than the /16 system.

Figure 6.3 shows the vast difference in data rates between the two systems, but when normalised, as in figure 6.4 the general patterns of background radiation can be seen in both.

System	Total Data Received (GB)	Average Data Rate Mb/s
/16	95.6	1.25
/24	1.9	0.0146

Table 6.1: Data and data rate received by each system over the period of one week

6.2.3 Mirai Botnet Data

Requests received from the Mirai botnet were extracted using the techniques described in Section 5.1. The analysis of the filtered traffic resulted in some surprising results. The results showed that, on average the smaller /24 system received 57.6 times more pings from

System	Unique Hosts
/16	2210
/24	104601

Table 6.2: Number of unique hosts seen by both systems

unique hosts per hour than the /16 system. This result is a complete reversal of the results seen above for the general data rates in Section 6.2.2.

The two systems saw a vastly different number and set of hosts. The total number of hosts is shown in Figure 6.2. Of these hosts there were only 18 common between the two systems. This low overlap demonstrates how many events the larger but less diverse /16 system missed.

The peak observed by both systems on 2019-03-19 corresponds with the news coverage of the Mirai botnet discussed in Section 2.7.3. This may be due to a general resurgence of interest in the Mirai malware following the news coverage.

6.2.4 Possible reasons for differences

The difference between the data received by the two systems was very unexpected. A number of scenarios were considered to account for the differences.

Poorly routed address space

The Border Gateway Protocol (BGP) is used to communicate between autonomous systems (AS) in order to route packets through the Internet.

The address space for the larger /16 system had not been used for a number of years before this experiment. It was possible that the operator HEAnet had not advertised this address space until soon before this research began. This could have resulted in some hosts on the Internet not being able to get packets to the address space. The operator verified that this was not the case and could demonstrate that the IP address space was perfectly propagated and accessible globally and had been for a number of years.

Some worms use BGP routing tables to only enumerate routable portions of the Internet[40]. While this was not the case in the public versions of the Mirai source code it was possible that new strains of the worm used this optimisation.

Bias in sequence of IP addresses generated

The method by which the Mirai botnet chooses IP addresses to scan was detailed in Section 2.7.1. The Mirai botnet uses a pseudo-random number generator (PRNG) to generate IP addresses. It was possible that there was a bias in the output of this random generator function. This could have been a negative bias away from the `/16` address space or a positive bias towards the IP addresses used in the `/24` system. It was established by Riegel that there was no bias in the PRNG used in the Mirai bot[27]. The blacklist of hardcoded IP address blocks mentioned in Section 2.7.1 was also checked and the `/16` range was verified to not be in this list.

Information was also sought from a number of IT security researchers who did not report knowing of any Mirai variants which were doing selective scanning.

Incorrect data collection

It was possible that the data filtering methods used did not correctly filter the collected data. It was thought that processing the large files from the `/16` system could have caused the `tshark` tool to fail and only output results from the start of the capture file. To verify that this was not the case the small filtered data was checked using `tshark` to output only the timestamps. These timestamps were reviewed and showed that there were results for the beginning, middle and end of the hour window. It was also verified that the filtering was working as expected by manually reviewing a number of packets using the `Wireshark` tool.

As none of these possibilities offer an explanation for the differences observed, it has been concluded that these differences are due to the general characteristics of the systems.

6.2.5 Observations

Additional analysis was performed on the captured traffic as well as the filtered Mirai traffic from both systems. These results are not related to comparison of the performance of the systems but instead give some insights typically obtained from network telescopes.

Top countries

Geolocation of IP addresses was performed as described in chapter 5. This was done to identify which countries have the most number of hosts compromised by the Mirai botnet. The data was plotted on a map in Figure 6.8 and shown in graph form Figure 6.7.

This data accounts for every IP address seen over the collection period. This may cause the results to skew in favour of countries with a high rate of DHCP churn. DHCP (The Dynamic Host Configuration Protocol) is used by Internet providers to assign an IP address to a customer. The rate at which a provider changes the customers IP address is called the DHCP churn rate. Moura et al. show that IP addresses are typically stable for about 10-60 hours in the Internet providers they studied[41], this could cause a single host to be counted multiple times over the one week collection period.

Top Ports

The top 20 most common TCP ports attacked are shown in figures 6.9 and 6.10. The difference between the two systems is notable. It appears that in the /24 system there is a bias towards standard HTTP ports 80 and 8088. This may be due to the IP addresses being in the AWS range, as AWS is typically used to host web content. This range could be targeted by attackers as they know they are likely to find web services at these IP addresses. It may also be due to misconfiguration of devices connecting to services that were once hosted at the IP addresses used.

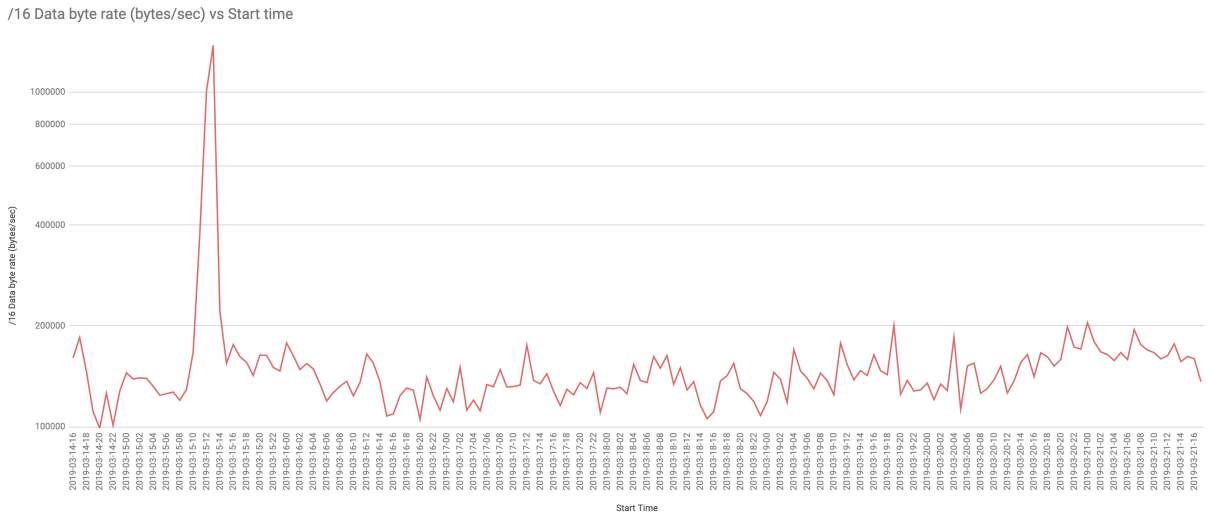

Figure 6.1: Data rate of /16 system (Log Scale)

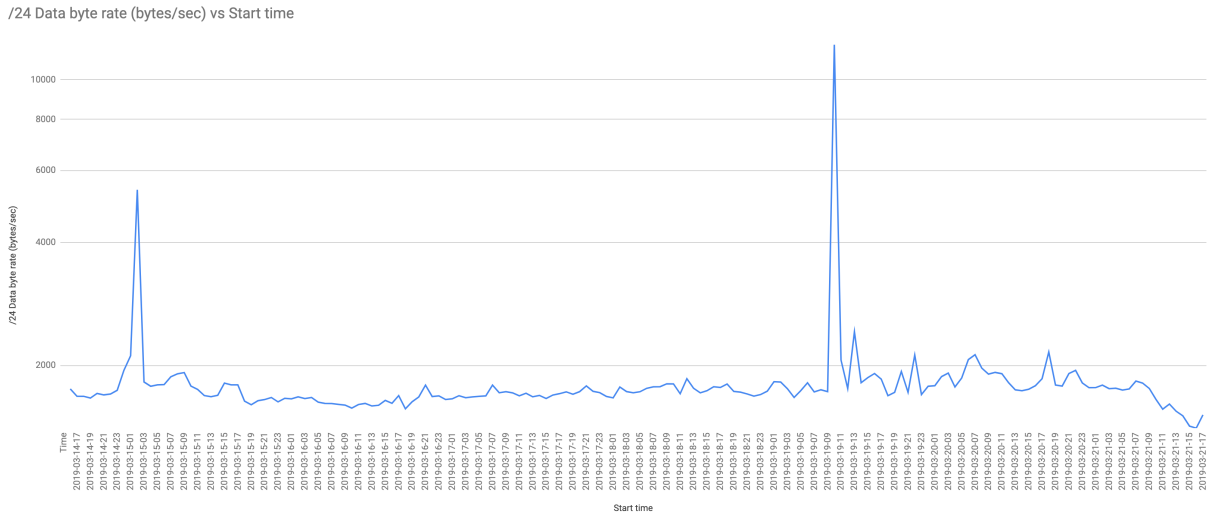

Figure 6.2: Data rate of /24 system (Log Scale)

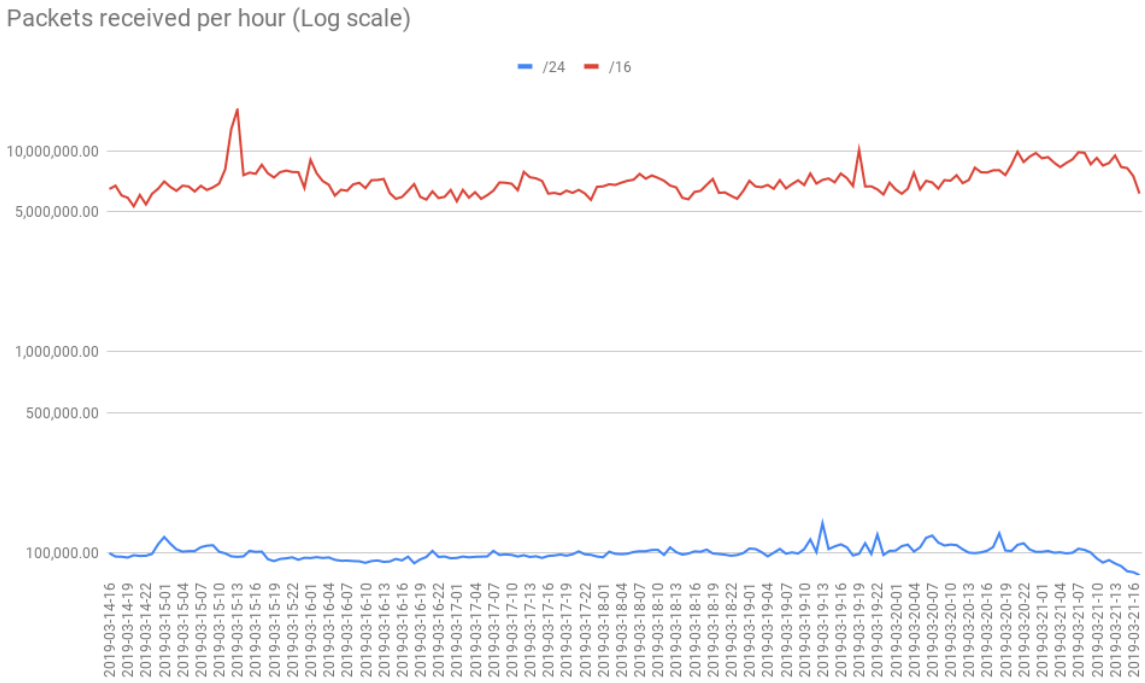

Figure 6.3: Packets received per hour for both systems (Log Scale)

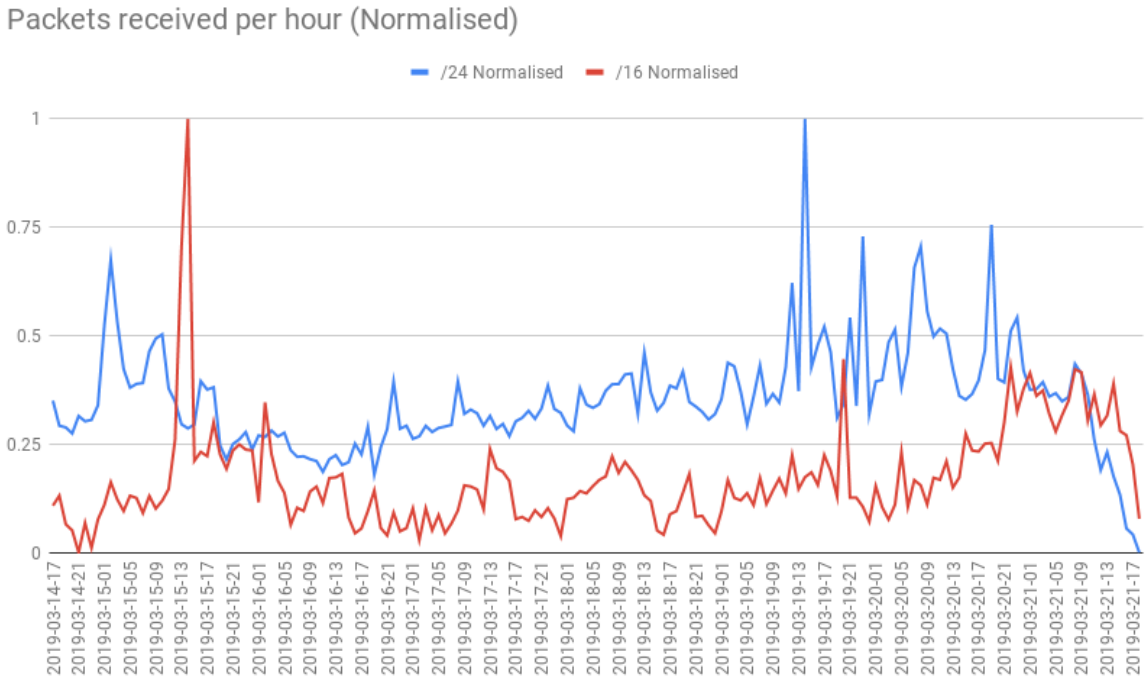

Figure 6.4: Packets received per hour for both systems (Normalised)

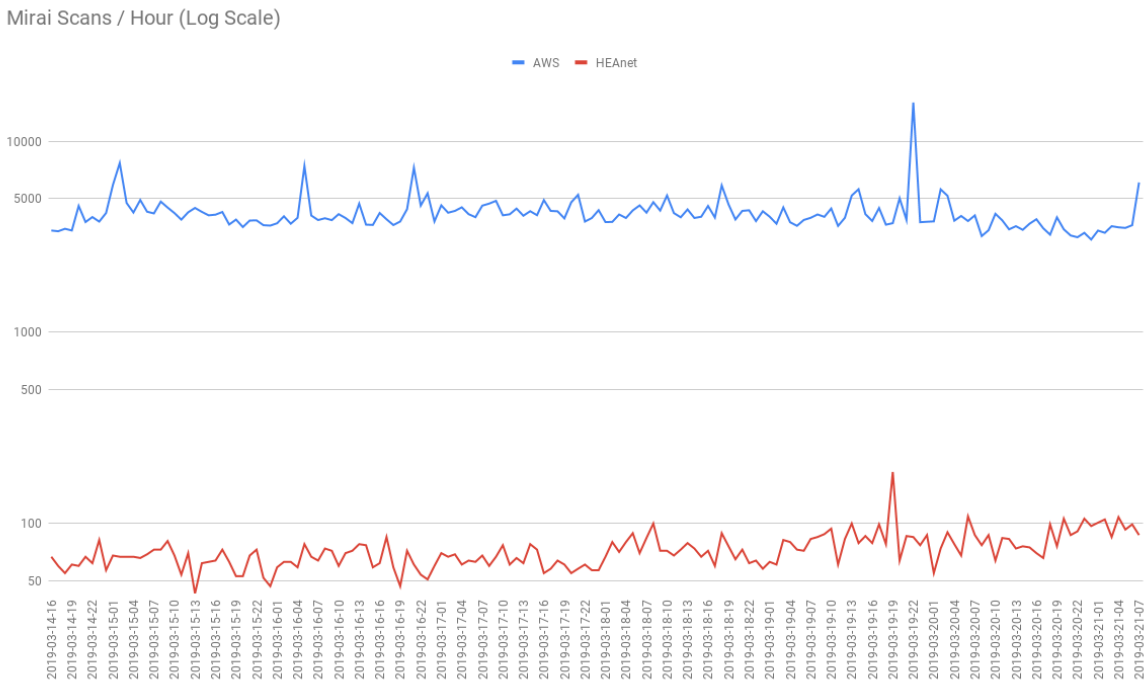

Figure 6.5: Mirai packets received per hour for both systems (Log Scale)

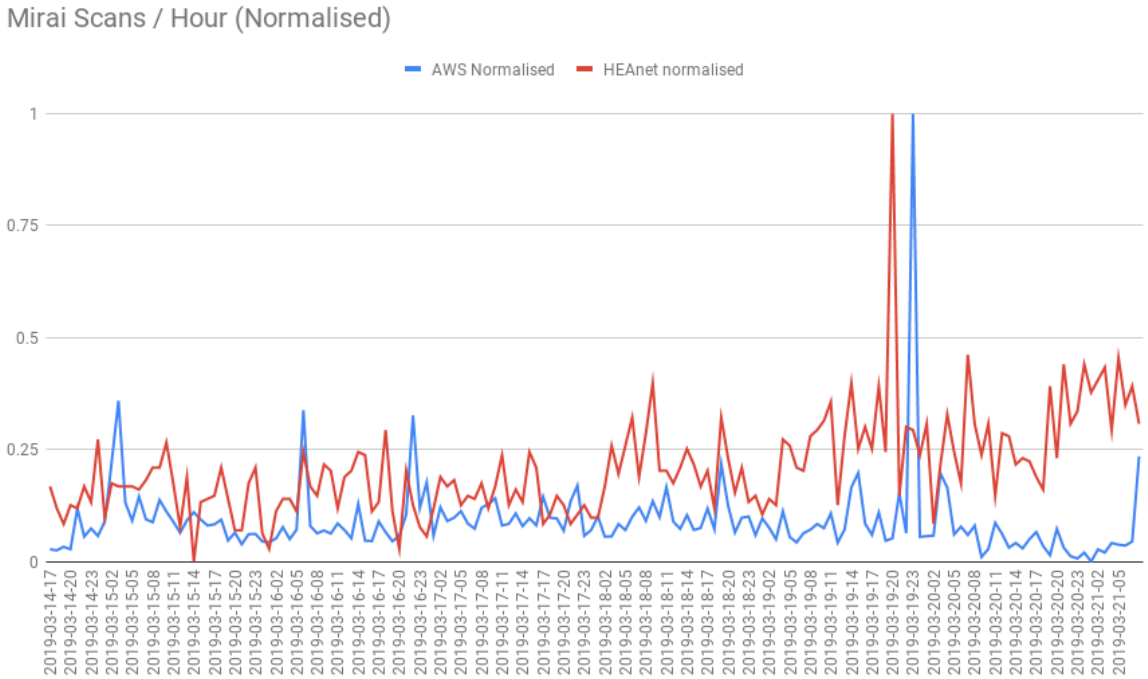

Figure 6.6: Mirai packets received per hour for both systems (Normalised)

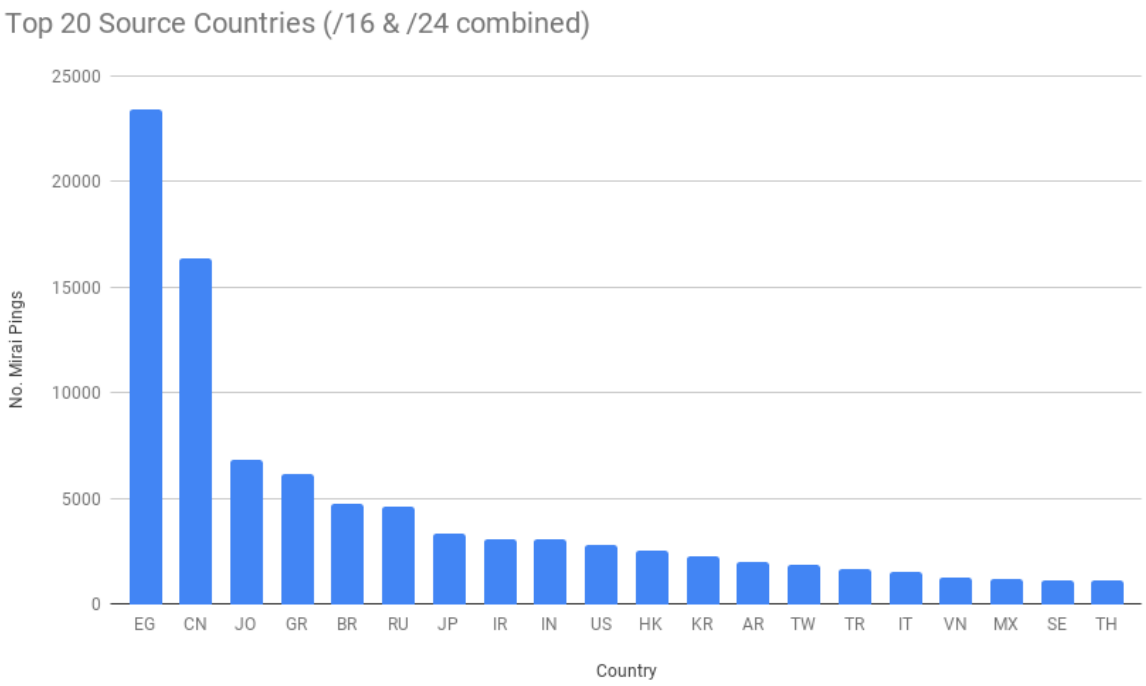

Figure 6.7: Top 20 ports as observed by /16 and /24 systems

Figure 6.8: Map of the world showing the location of Mirai infected hosts

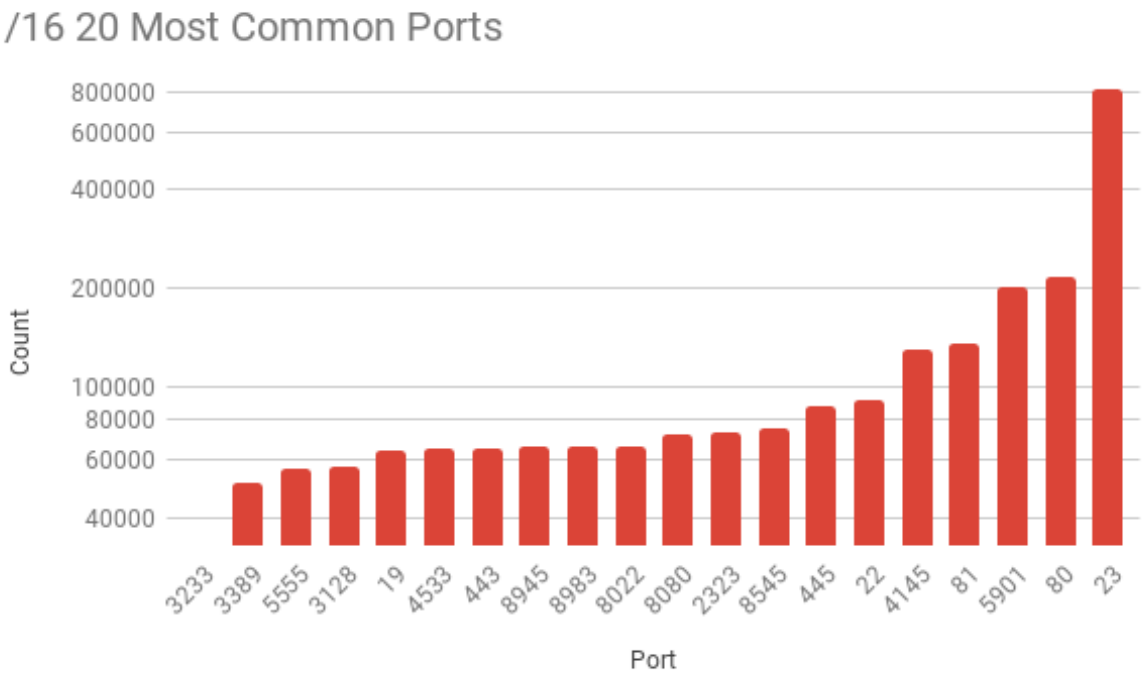

Figure 6.9: Top 20 ports as observed by /16 system (Log Scale)

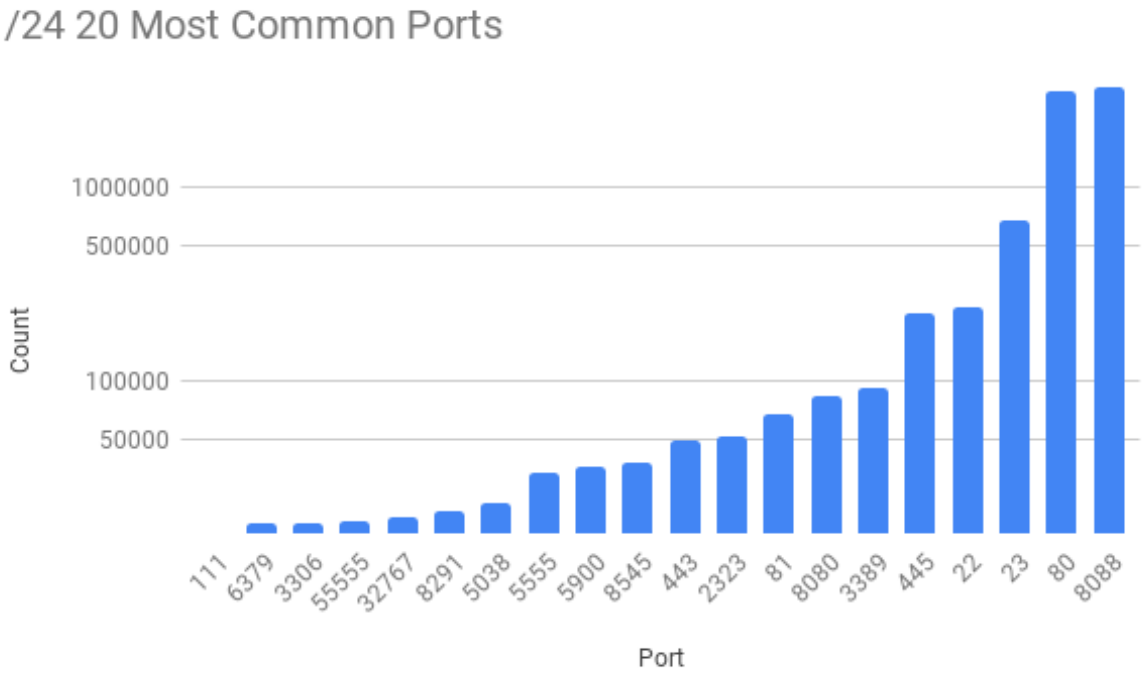

Figure 6.10: Top 20 ports as observed by /24 system (Log Scale)

7 Conclusion

This research aimed to determine if a small cloud-based network telescope with a very diverse IP address range can perform better at the task of botnet analysis than a traditional and larger network telescope.

The research aimed to achieve this by:

- Comparing the number of botnet hosts seen by a large telescope and the proposed telescope
- Number of scans from botnet hosts
- Overlap in botnet hosts seen by each system
- Comparing traffic patterns observed by the two systems

This research has shown that a smaller network telescope with a significantly more diverse IP address range can receive much more data for specific types of events than a larger but less diverse network telescope.

More research is needed to determine how these results can be used to determine broader network behaviour.

7.1 Future Work

As this is novel system design, much more work is needed to determine the best approach to build and operate this type of system. Some areas for additional research are:

Cycle through available IP addresses on AWS to achieve a desired level of diversity - and measure the difference between more and less diverse networks of this kind.

Build a similar system on Azure as it may be cheaper and easier.

Chindipha et al. shows that the starting IP addresses get more hits than the latter IPs[38]. This bias in IP selection could be tested by selectively choosing AWS IP addresses.

New research is needed comparing a totally mixed greynet to a contiguous block of the same size and evaluate the performance.

The design proposed in this paper could be configured as a higher interaction system to act as a sinkhole for misconfigured devices pointing to AWS IP addresses space. These interactions could be from systems pointing to IP addresses no longer under the control of their previous owner. Such a system could collect credentials or other sensitive information for analysis.

Bibliography

[1] Center for Applied Internet Data Analysis. Historical and near-real-time ucsd network telescope traffic dataset. URL `https://www.caida.org/data/passive/telescope-near-real-time_dataset.xml`.

[2] David Moore, Colleen Shannon, Geoffrey Voelker, Stefan Savage, et al. Network telescopes: Technical report. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), 2004.

[3] Steven Michael Bellovin. Packets found on an internet. 1993.

[4] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Peterson. Characteristics of internet background radiation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 27–40. ACM, 2004.

[5] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Huston. Internet background radiation revisited. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 62–74. ACM, 2010.

[6] Eugene H Spafford. The internet worm program: An analysis. *ACM SIGCOMM Computer Communication Review*, 19(1):17–57, 1989.

[7] V. Fuller and T. Li. Classless inter-domain routing (cidr): The internet address assignment and aggregation plan. BCP 122, RFC Editor, August 2006. URL `http://www.rfc-editor.org/rfc/rfc4632.txt`.

[8] Mark Allman, Vern Paxson, and Jeff Terrell. A brief history of scanning. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 77–82. ACM, 2007.

[9] J Reynolds and Jon Postel. Rfc 1340-assigned numbers. *Internet Engineering Task Force*, 1992.

[10] Zakir Durumeric, Michael Bailey, and J Alex Halderman. An internet-wide view of internet-wide scanning. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 65–78, 2014.

[11] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pages 605–620, 2013.

[12] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647. ACM, 2009.

[13] Brian Krebs. Israeli online attack service 'vdos' earned $600,000 in two years — krebs on security. https://krebsonsecurity.com/2016/09/israeli-online-attack-service-vdos-earned-600000-in-two-years/. (Accessed on 03/30/2019).

[14] Ryan Brunt, Prakhar Pandey, and Damon McCoy. Booted: An analysis of a payment intervention on a ddos-for-hire service. In *Workshop on the Economics of Information Security*, 2017.

[15] Arman Noroozian, Maciej Korczyński, Carlos Hernandez Gañan, Daisuke Makita, Katsunari Yoshioka, and Michel van Eeten. Who gets the boot? analyzing victimization by ddos-as-a-service. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 368–389. Springer, 2016.

[16] Max Goncharov. Criminal hideouts for lease: Bulletproof hosting services. *Forward-Looking Threat Research (FTR) Team, A TrendLabsSM Research Paper*, 28, 2015.

[17] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[18] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.

[19] Manyika James, M Chui, P Bisson, J Woetzel, R Dobbs, J Bughin, and D Aharon. The internet of things: Mapping the value beyond the hype. *McKinsey Global Institute*, 3, 2015.

[20] A Keranen and Carsten Bormann. Internet of things: Standards and guidance from the ietf. *IETF J.*, 2016. URL `https://www.ietfjournal.org/internet-of-things-standards-and-guidance-from-the-ietf/`.

[21] Malware Must Die! Mmd-0056-2016 - linux/mirai, how an old elf malcode is recycled.. · malwaremustdie!, August 2016. URL `http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html`.

[22] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *USENIX Security Symposium*, pages 1092–1110, 2017.

[23] Brian Krebs. Krebsonsecurity hit with record ddos. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, 2016. (Accessed on 03/30/2019).

[24] Islam N. Bertino, E. Botnets and internet of things security. *Computer*, (2):76, 2017. ISSN 0018-9162.

[25] Dan Goodin. Record-breaking ddos reportedly delivered by> 145k hacked cameras, 2016. URL `https://arstechnica.com/information-technology/2016/09/botnet-of-145k-cameras-reportedly-deliver-internets-biggest-ddos-ever`.

[26] February 28th ddos incident report - the github blog. `https://github.blog/2018-03-01-ddos-incident-report/`. (Accessed on 03/30/2019).

[27] Meghan Riegel. Tracking mirai: An in-depth analysis of an iot botnet. Master's thesis, Pennsylvania State University, 2017.

[28] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. URL `http://www.rfc-editor.org/rfc/rfc793.txt`. `http://www.rfc-editor.org/rfc/rfc793.txt`.

[29] Mirai botnet authors avoid jail time — krebs on security. `https://krebsonsecurity.com/2018/09/mirai-botnet-authors-avoid-jail-time/`. (Accessed on 04/01/2019).

[30] Spadafora Anthony. Mirai botnet returns to target iot devices | techradar. `https://www.techradar.com/news/mirai-botnet-returns-to-target-iot-devices`. (Accessed on 03/30/2019).

[31] Steve Bellovin. There be dragons. In *USENIX Summer*, 1992.

[32] Lance Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 99(2):15–23, 2003.

[33] Lance Spitzner. Honeypots: Catching the insider threat. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 170–179. IEEE, 2003.

[34] Evan Cooke, Michael Bailey, David Watson, Farnam Jahanian, and Jose Nazario. The internet motion sensor: A distributed global scoped internet threat monitoring system. *Technical Report CSE-TR-491-04*, 2004.

[35] Vinod Yegneswaran, Paul Barford, and Dave Plonka. On the design and use of internet sinks for network abuse monitoring. In *International Workshop on Recent Advances in Intrusion Detection*, pages 146–165. Springer, 2004.

[36] Warren Harrop and Grenville Armitage. Defining and evaluating greynets (sparse darknets). In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 344–350. IEEE, 2005.

[37] Evan Cooke, Michael Bailey, Z Morley Mao, David Watson, Farnam Jahanian, and Danny McPherson. Toward understanding distributed blackhole placement. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 54–64. ACM, 2004.

[38] Stones Chindipha, Barry Irwin, and Alan Herbert. Effectiveness of sampling a small sized network telescope in internet background radiation data collection. 09 2018.

[39] Robin Berthier, Dave Korman, Michel Cukier, Matti Hiltunen, Gregg Vesonder, and Daniel Sheleheda. On the comparison of network attack datasets: An empirical analysis. In *2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 39–48. IEEE, 2008.

[40] Cliff C Zou, Don Towsley, Weibo Gong, and Songlin Cai. Routing worm: A fast, selective attack worm based on ip address information. In *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 199–206. IEEE Computer Society, 2005.

[41] Giovane CM Moura, Carlos Ganán, Qasim Lone, Payam Poursaied, Hadi Asghari, and Michel van Eeten. How dynamic is the isps address space? towards internet-wide dhcp churn estimation. In *2015 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2015.

A1 Appendix

A1.1 Python Code

A1.1.1 aws.py

```
from botocore.exceptions import ClientError

from utils import *

import boto3

client = boto3.client('ec2')

DryRun = False
MaxInterfaces = 8
MaxIpsPerInterface = 30

InstanceId = 'i-06ea4a12f5622fbb5'
VpcId = 'vpc-0fa32ae9050503bf3'
SecurityGroupId = 'sg-06ea256f9f1be1c66'


def get_interfaces(instance_id):
    instances = client.describe_instances(Filters=[
        {
            'Name': 'instance-id',
            'Values': [instance_id]
        }
    ])
```

```
    instance = instances['Reservations'][0]['Instances'][0]

    return instance['NetworkInterfaces']


# Creates an elastic IP for any private IP addresses that do not
    currently have a public IP
def allocate_and_assign_address(instance_id):
    interfaces = get_interfaces(instance_id)

    for interface in interfaces:
        for private_ip in interface['PrivateIpAddresses']:
            if 'Association' not in private_ip:
                new_ip = client.allocate_address(Domain='vpc')
                client.associate_address(NetworkInterfaceId=
                    interface['NetworkInterfaceId'],
                                         AllocationId=new_ip['
                                             AllocationId'],
                                         PrivateIpAddress=
                                             private_ip['
                                             PrivateIpAddress'])


# Allocates the max amount of private IPs to every interface on
    an instance
def allocate_private_ips(instance_id):
    interfaces = get_interfaces(instance_id)
    for interface in interfaces:
        numPrivIps = len(interface['PrivateIpAddresses'])
        if numPrivIps < MaxIpsPerInterface:
            response = client.assign_private_ip_addresses(
                AllowReassignment=False,
                NetworkInterfaceId=interface['NetworkInterfaceId
                    '],
                SecondaryPrivateIpAddressCount=
                    MaxIpsPerInterface - numPrivIps
            )
```

```
def check_subnet_exists(cidr):
    subnets = client.describe_subnets(
        Filters=[
            {
                'Name': 'vpc-id',
                'Values': [VpcId]
            },
        ])
    for subnet in subnets['Subnets']:
        if subnet['CidrBlock'] == cidr:
            return True

    return False


def get_subnet_id(cidr):
    subnets = client.describe_subnets(
        Filters=[
            {
                'Name': 'vpc-id',
                'Values': [VpcId]
            },
        ])
    for subnet in subnets['Subnets']:
        if subnet['CidrBlock'] == cidr:
            return subnet['SubnetId']

    return ''


def create_subnet(cidrBlock):
    subnet = client.create_subnet(CidrBlock=cidrBlock, VpcId=
       VpcId)
    subnet_id = subnet['Subnet']['SubnetId']
    client.associate_route_table(SubnetId=subnet_id,
       RouteTableId="rtb-0c9575e2f57fac800")
    print("Created subnet %s and associated route table",
       cidrBlock)
    return subnet_id
```

```
# Create up to max interfaces for the instance
def allocate_and_assign_interfaces(instance_id):
    num_interfaces = len(get_interfaces(instance_id))

    for i in range(num_interfaces, MaxInterfaces):
        cidr = '10.0.' + str(i) + '.0/24'
        subnet_id = get_subnet_id(cidr)
        if subnet_id == '':
            subnet_id = create_subnet(cidr)

        network_interface_res = client.create_network_interface(
           SubnetId=subnet_id, Groups=[SecurityGroupId])
        network_interface_id = network_interface_res['
           NetworkInterface']['NetworkInterfaceId']
        res = client.attach_network_interface(
            DeviceIndex=i,
            InstanceId=instance_id,
            NetworkInterfaceId=network_interface_id
        )

    return


def deallocate_unused_addresses():
    ips = client.describe_addresses()
    for ip in ips['Addresses']:
        try:
            res = client.release_address(AllocationId=ip['
               AllocationId'], DryRun=DryRun)
            print("Deallocated␣IP:␣%s" % ip['PublicIp'])

        except ClientError as e:
            if e.response['Error']['Code'] == 'InvalidIPAddress.
               InUse':
                print("Failed␣to␣deallocate␣IP:␣%s" % ip['
                   PublicIp'])
                print(e.message)
```

```
            else:
                print("Unexpected error: %s" % e)

    return


def print_all_addresses():
    pretty_print(client.describe_addresses())
    return


def start_instance(instance_id):
    res = client.start_instances(InstanceIds=[instance_id])
    pretty_print(res)


def describe_instances():
    pretty_print(client.describe_instances())


allocate_and_assign_interfaces(InstanceId)
allocate_private_ips(InstanceId)
allocate_and_assign_address(InstanceId)
deallocate_unused_addresses()
```

A1.1.2 ip-addr.py

```
import socket, struct
import boto3

"""
    Print the Public/Private IP Pairs and the public IP as a
       LONG.
    Used for mapping back to Mirai probes in .pcap files.

"""
client = boto3.client('ec2')

DryRun = False
```

```
MaxInterfaces = 8
MaxIpsPerInterface = 30

InstanceId = 'i-06ea4a12f5622fbb5'
VpcId = 'vpc-0fa32ae9050503bf3'
SecurityGroupId = 'sg-06ea256f9f1be1c66'


def print_address_pairs():
    address_res = client.describe_addresses()
    addresses = address_res['Addresses']
    for address in addresses:
        if address['InstanceId'] == InstanceId:
            print address['PublicIp'] + '\t' + address['
               PrivateIpAddress'] + '\t' + str(ip2long(address['
               PublicIp']))


def ip2long(ip):
    """
    Convert an IP string to long
    """
    packedIP = socket.inet_aton(ip)
    return struct.unpack("!L", packedIP)[0]
```

A1.1.3 gen-aws-query.py

```
import csv

"""
        Generates a tshark query string that can be used to
           filter .pcap files for Mirai probes
        Input is from ip-addr.py
"""

with open('ips.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',', quoting=csv.
       QUOTE_NONE)
    for row in reader:
```

```
        print "(tcp.seq == " + row[2] + " && ip.dst == " + row
            [1] + ") ||",
```

A1.1.4 geo-ip.py

```
import geoip2.database
from geoip2.errors import AddressNotFoundError


def generate_line_string(ip_addr, geo, source):
    return ip_addr + "," + geo.country.iso_code + "," + str(geo.
        location.latitude) + "," + str(geo.location.longitude) +
        "," + str(source)

reader = geoip2.database.Reader('GeoLite2-City/GeoLite2-City.
    mmdb')



ips = dict()

f = open("aws_all_hosts_no_dupes.txt", "r")
for ip in f:
    ip = ip.rstrip()
    response = reader.city(ip)

    if response is not None and response.country.iso_code is not
        None:
        if ip in ips:
            ips[ip] = generate_line_string(ip, response, 2)
        else:
            ips[ip] = generate_line_string(ip, response, 0)
    else:
        print "failed to map " + ip


f = open("heanet_all_hosts_no_dupes.txt", "r")
for ip in f:
    ip = ip.rstrip()
```

```
    try:
        response = reader.city(ip)
    except AddressNotFoundError:
        print "Couldn't find ip in DB " + str(ip)
        continue

    if response is not None and response.country.iso_code is not
        None:
        if ip in ips:
            ips[ip] = generate_line_string(ip, response, 2)
        else:
            ips[ip] = generate_line_string(ip, response, 1)
    else:
        print "failed to map " + ip


for line in ips.values():
    print line

reader.close()
```

A1.2 tcpdumpd.service

```
[Unit]
After=network.target

[Service]
Restart=always
RestartSec=20
Environment="TCPDUMP_FORMAT=%%Y-%%m-%%d-%%H"
ExecStartPre=/bin/mkdir -p /home/johara/dumps/
ExecStart=/usr/sbin/tcpdump -i all -G 3600 -s 65535 -w '/home/
   johara/dumps/heanet_${TCPDUMP_FORMAT}.pcap'
ExecStop=/bin/kill -s QUIT $MAINPID

[Install]
WantedBy=multi-user.target
```

A1.3 Bash Scripts

A1.3.1 all_mirai_aws.sh

```
#!/bin/bash
for filename in /home/johara/telescope/data/cap/aws_*.pcap; do
        ./filter_aws_mirai.sh $filename
done
```

A1.3.2 all_mirai_heanet.sh

```
#!/bin/bash
for filename in /home/johara/telescope/data/cap/aws_*.pcap; do
        ./filter_heanet_mirai.sh $filename
done
```

A1.3.3 filter_aws_mirai.sh

```
#!/bin/bash
filter='(tcp.seq == 315257032 && ip.dst == 10.0.0.87) || (tcp.
   seq == 315257272 && ip.dst == 10.0.3.125) ....truncated '
tshark -r $1 -o tcp.relative_sequence_numbers:FALSE -Y "$filter"
    -w /home/joharatelescope/data/mirai/"$(basename␣$1)"
```

A1.3.4 filter_heanet_mirai.sh

```
#!/bin/bash
#MIN and MAX removed for privacy reasons
tshark -r $1 -o tcp.relative_sequence_numbers:FALSE -Y "tcp.seq␣
   >=␣MIN_IP␣&&␣tcp.seq␣<=␣MAX_IP" -w /home/johara/telescope/
   data/mirai/"$(basename␣$1)"
```

A1.3.5 count_mirai_hosts.sh

```
#!/bin/bash
for filename in /home/johara/telescope/data/mirai/{system}*.pcap
   ; do
        ./count_unique_hosts.sh $filename
done
```

A1.3.6 count_unique_hosts.sh

```
#!/bin/bash
count=$(tshark -r $1 -T fields -e ip.src | sort -n | uniq | wc -
    l)
echo $(basename $1)      $count
```