University of Dublin

Trinity College

Design Characteristics of Honeypots – with an Aim to Study Botnets

Student - Nigel Farrelly Supervisor – Stefan Weber

Abstract

The security of internet-connected devices is a topic of ever increasing interest. This is true not solely for people who are involved in research in the field, nor is it limited to people who work within the area of network security, but for a majority of people who own and operate such a device. Botnets are at the heart of much of the malicious and illegal activity that occurs online. They are responsible for a wide range of criminal acts, varying from seemingly more minor fraudulent acts to more high profile incidents involving prominent components of the internet infrastructure. The distributed denial of service, (DDoS), attack on the American domain name system server *Dyn* in October 2016 is an illustrative example of the potential scale of these attacks. This particular botnet powered attack led to sites such as Twitter, Reddit and PayPal finding themselves temporarily offline. [1] Honeypots are a way for researchers and professionals to interact with these botnets, to monitor their behaviour and evolution and to help in the development of new strategies for limiting the damage these botnets can cause, both to companies and to individuals.

When undertaking this project, there were a few important aims. The first aim was to analyse the design and efficacy of currently available honeypots, to deploy and experiment with a selection of these honeypots, and to examine how well these designs are suited to dealing with the current threat posed by the ever evolving botnet family. Studying the architecture and development of botnets was the second foremost aim of this project. This would be achieved by working with relevant and current literature on the topic, then using this knowledge to progress to deploying a range of honeypots in order to enable real interaction with active botnets. Lastly, as the project progressed the focus became centred on the Internet of Things and the rapidly developing security threat these devices pose. Understanding the gulf between currently available open source honeypots and what is required in order to properly track the threat of these IoT botnets became an area of particular interest. Using this information as a platform, the intention was to design and deploy an IoT honeypot capable of interacting with IoT devices in a convincing manner, with the hope of obtaining malware binaries for further analysis.

Table of Contents

Chapter 1

Introduction

October 2016 witnessed one of the largest DDoS attacks in internet history. The security site *KrebsOnSecurity* was targeted by a botnet harnessing the power of thousands upon thousands of compromised IoT devices. Without the use of any amplification techniques, traffic reached up to 620Gbps. The threat presented by botnets shows no signs of abating, quite the opposite. We are in an age where the majority of people in developed countries own and regularly operate internet connected devices. Users with no technical background, nor security awareness, operate these devices unaware of their intrinsic security flaws. Internet of Things devices are internet connected at all times, they have global IP addresses and more often than not have weak login credentials. They make for the ideal botnet target. The idea behind honeypots is to allow researchers and professionals alike the ability to interact with these botnets in a, depending on the level of interaction, controlled environment. The ability to monitor the attackers in this way is a valuable resource in gaining insight into how these botnets operate.

1.1 Project Goal

The goal of this project is to gain an in depth understanding of the design characteristics of actively maintained honeypots, with a view to studying botnets. This will be achieved through a combination of literature research and active experimentation with honeypot deployment. The literature will be from both academic and respected professional security sources, which can often offer valuable insight in a fast moving field. As well as deploying some of the more prominent currently well-maintained honeypots, the aim is to deploy a dynamic research Internet of Things honeypot. The aim will be to monitor all interactions between the deployed honeypots and malicious users. A final aim is for the honeypot to be infected with malware, and to analyse the malware's network communication.

1.2 Report Layout

This report begins with an overview of literature relating to the field of honeypots and botnets. This is intended as an introduction to the reader to the current state of the art in this area of network security. In the first section of the literature review we will discuss what a honeypot is, the differing types of honeypots, and from there we will move on to examine some examples of the more prominent currently available honeypots. In the second section we will be discussing botnets and the current trends apparent in botnet fueled attacks. From here we will discuss the more recent security issues that have developed around the Internet of Things. We will examine this in terms of how it has impacted on the current environment, from a botnet perspective and also from the point of view of honeypot developments. From this point we will progress to discussing the initial setup for the deployed honeypots. We will then discuss the issues encountered when considering how to deploy a honeypot to monitor IoT botnets. We will discuss the design and the eventual implementation of such a honeypot. The report is concluded with an evaluation of both the process and the results of the honeypot deployments.

Chapter 2

State of the Art

This section is intended as an introduction to honeypots and botnets. We discuss the state of the art, how the landscape for both has changed in recent years, while also examining specific well-known instances of both. We include a self-contained section dedicated to the Internet of Things. The intended outcome of this section is to give the reader an overview of the growing role these devices are playing within the spectrum of botnet-powered attacks. We discuss the inherent security flaws apparent in many of these devices, how these flaws make them ideal targets for botnets, and the rise of botnets created to target IoT devices. We look at examples of such botnets and some of the higher profile attacks they have been responsible for. We then examine a honeypot, IoTPOT, which was designed specifically with this new threat in mind.

2.1 Honeypots

A honeypot is a system that is created with the intention of enticing botnets and human attackers to attempt to access and interact with the system. A honeypot system is set up in such a way that makes it appear to be vulnerable to malicious attacks. This can be achieved in various ways, for example the honeypot may run an outdated operating system with known security issues, or run services with known security issues, that an attacker will intend to exploit. There are numerous categories of honeypots and potential honeypot set ups that we will discuss in more detail in the coming sections. There are various reasons for an individual or organisation to deploy a honeypot. They may be deployed as a component in an overall security system or simply as a means of monitoring and interacting with attackers as a form of research. Honeypots are classified at the deployment level as well as the interaction level.

2.1.1 Honeynets

A honeynet is a network of high interaction honeypots. It simulates a production network, monitoring and recording all incoming and outgoing traffic. Typical production systems are included within the honeynet offering real services, thus making it more difficult to fingerprint, and more likely to entice malicious users. [2]

2.1.2 Categories of Deployment

There are two types of deployment categories we associate with honeypots, production honeypots and research honeypots.

I) Production Honeypot

A production honeypot is a honeypot deployed as a component of a company's overall security infrastructure. Honeypots are not generally useful as a means of preventing attacks on a company's network, however, they can prove to be extremely useful in detecting unwanted traffic within the network. Their obvious value in this sense is that any connections, or attempts at connection, to the honeypot system are more than likely from a suspicious source. In a company dealing with large amounts of incoming traffic, suspicious sources are not always easily detected by an intrusion detection system (IDS), in this scenario a honeypot can add great value to the company's security set up. Production honeypots can also be useful in the aftermath of an attack. If the honeypot was compromised during the attack it is easy to take the honeypot offline, examine how the system was penetrated and how the malicious user interacted with the system. Naturally this is done much more easily with an isolated honeypot where most of the interactions are with a malicious user than with an active production component of the company's network, while taking the honeypot offline has no adverse effect on the service provided by the company. A production honeypot can also be deployed within a company's network to operate as a decoy 6

system. The goal in this scenario is to attract the attention of would be attackers away from components of the system with real production value. Production honeypots tend to be in the medium or low-level interaction category. [3] [4]

II) Research Honeypot

A research honeypot is deployed, as one might expect, purely as a tool for research. It is not used as a means of improving a company's security infrastructure as discussed in the previous section, but as a means of monitoring the behaviour of attackers. The aims when deploying a research honeypot are, among other, to examine the types of attacks and intrusion attempts, to monitor the malicious user's interaction with the system if compromised, to monitor the traffic generated by the user and often to obtain and analyse malware downloaded to the honeypot system by the attacker. The honeypots deployed and implemented for this project are research honeypots.

2.1.3 Levels of Interaction

Historically the levels of interaction of a honeypot system have been divided into three categories, high-level, medium-level and low-level. The difference between low-level and medium-level interaction honeypots can often be ambiguous. The level of interaction is an important consideration when deciding on the honeypot one wishes to deploy. Setting the level of interaction is a way of controlling the extent to which the attacker can interact with the honeypot system. There is a trade off to be considered when choosing the level of interaction of one's honeypot. We shall discuss the advantages and disadvantages of each in the issues and tradeoff section. [5]

I) High Level Interaction

A high-level interaction honeypot tends to be a real fully-fledged operating system, with expected services and applications available. The experience for the user should be almost identical to what they would expect from interacting with a standard release of the system. High interaction honeypots are used by organisations or researchers wishing to obtain detailed analysis of how malicious users and botnets operate.

There are obvious issues with deploying a high interaction honeypot. The apparent vulnerabilities of the system are not emulated as with medium interaction honeypots, they are real vulnerabilities and as such are open to real exploitation by an attacker, with infection by malware a dangerous possibility. There needs to be careful consideration taken when deciding on the set up and configuration of such a honeypot. One needs to be careful that in attempting to garner data relating to attacks the honeypot itself does not become involved in malicious activity, such as DDoS attacks or propagating malware to other devices.

There are steps that an organisation can take in order to minimise the risks involved when deploying a high interaction honeypot, which we will discuss in greater detail in a later section.

Suffice to say there is a careful balancing act between making the honeypot as attractive to attackers and as useful as possible while also ensuring its security.

While possible, it should prove much more difficult for an attacker to deduce that they are interacting with a honeypot when the honeypot is of high interaction rather than medium interaction.

II) Medium Level Interaction

A medium-level interaction honeypot is generally not a real system but a piece of software that emulates components of a real system. Medium interaction honeypots are usually designed with the intention of monitoring a specific type of threat. We will look more closely at examples of real world medium honeypots shortly but as an idea, SNARE is designed to monitor attacks on web applications, Conpot is a honeypot dedicated to dealing with attacks on industrial control systems and so on. A medium interaction honeypot is designed to appear vulnerable to certain types of attacks. This can be achieved in a variety of ways, for example by emulating vulnerable protocols, including services for which known exploits exist, or emulating an outdated operating system.

When a malicious user gains access to a medium interaction honeypot she is effectively interacting with a sandbox, designed to appear to the user as a real system. This can lead to issues whereby the attacker, botnet or human, discover they are not interacting with the actual system they were expecting and upon this discovery exit the system. This is often referred to as *fingerprinting*. When interacting with the system the malicious user may attempt to execute a command specific to the architecture of the device they believe they are interacting with, if the user does not receive the specific response they are expecting they may deduce that they are not interacting with a real system and will choose to exit. This is one of the main challenges faced by honeypots creators. Creating a dynamic, reactive honeypot that manages to convince multiple attackers that they are indeed interacting with an actual system is not a straightforward task. We will discuss this issue in greater detail later in the report.

III) Low Level Interaction

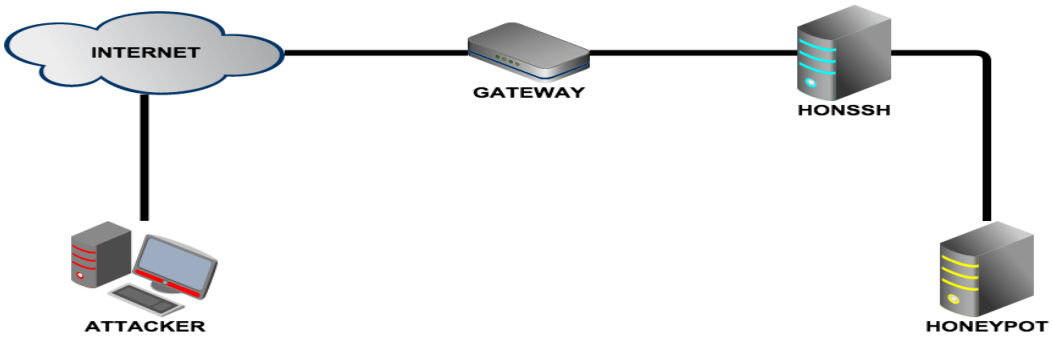
Typically a low interaction honeypot has very few if any services available to the user to interact with. A low interaction honeypot might present itself as a system simply to monitor log in attempts but not actually emulate any services or allow a user to log in, an example of such a honeypot is HoneyD. There are also many such honeypots on Github designed simply to monitor log in attempts on certain protocols such as Telnet, without offering the user any further interaction. As we already alluded to the difference between medium and low interaction honeypots is often quite ambiguous and there can be arguments about the correct classification of each. For this reason, from here on, we shall categorise medium and low interaction together under the medium interaction title.

2.1.4 High Interaction Honeypots

In this section we will look at some instances of well-maintained high interaction honeypots.

I) HonSSH

HonSSH provides a high-interaction honeypot solution. The diagram below is intended to give the reader an idea of the solution's architecture. HonSSH is a proxy placed between the malicious user and the honeypot. The proxy creates two separate SSH connections, accepting a connection from the attacker and forwarding a separate connection to the honeypot. HonSSH captures all connection attempts and saves them to a text file or database. HonSSH can spoof a login if the desire is to let the attacker login regardless of whether they have guessed the correct credentials. All the attacker's interactions with the system are monitored and stored in a TTY log. HonSSH also has an advanced networking feature used to avoid fingerprinting. With this feature enable it is possible to create fake IP addresses on the connection between the HonSSH box and the honeypot. This will ensure that it appears as if all packets are coming from the attacker and not the HonSSH box. HonSSH uses IP tables to achieve this. The IP tables are used to do network address translation (NAT) to make it appear as if all packets are coming from the attacker. HonSSH is inspired by the Kippo honeypot, which we will discuss in the medium interaction section, and takes some features directly from the Kippo project. [6]



HonSSH Architecture

II) Sebek / Qebek

Qebek and Sebek although no longer maintained were notable high interaction honeypots. Qebek was a direct successor to the Sebek honeypot. Qebek built upon the idea of using system calls as a way of monitoring an attacker's interaction with the honeypot as well as addressing some of the fingerprinting issues that arose with the Sebek honeypot. The monitoring of the malicious user in the Sebek honeypot was carried out in the operating system itself. Qebek addressed this by moving the monitoring to the hardware virtualisation component. Qebek uses QEMU, an open source hardware emulator, for the hardware virtualisation. Qebek is no longer maintained but was seemingly influential in the development of later honeypots, for example the IoTPOT, which we will discuss later in the report, uses QEMU as a means of emulating multiple CPU architectures.

2.1.5 Novel / Research Honeypots

In this section we will discuss two honeypots better classified under the title novel or research rather than their level of interaction. Neither honeypot is a real system, however, both are very reactive, dynamic honeypots designed to interact with a range of attacks and to give the appropriate responses to specific attackers. Both of these honeypots are developed by the same group of researchers. We will discuss the AmpPot honeypot in greater detail in this section as the IoTPOT will be covered at length later in the report.

I) AmpPot

The AmpPot honeypot, like the IoTPOT we will discuss in a later section, was developed by researchers at Yokohama National University and Saarland University. The source code for AmpPot was not made available online nor upon request but its design and deployment is detailed in the referenced paper by the researchers involved in its development. [7] The AmpPot honeypot was designed solely to monitor amplification type DDoS attacks. Please refer to the DDoS section later in this report, part VII, for a description of what an amplification attack entails. Briefly, the aim of an amplification attack is to consume the network bandwidth of its victim. Amplification attacks target vulnerable UDP protocols such as DNS using reflection techniques and IP spoofing to carry out an attack.

Operation

There are numerous UDP-based protocols that are susceptible to abuse by attackers. The most well-known is DNS on port 53, but others include NTP [123], SNMP [161], SIP [5061], Net-Bios [137] among others. The AmpPot honeypot supports these protocols by listening on the appropriate ports for incoming UDP packets. AmpPot provides three modes of response that govern the type of response the AmpPot sends upon receiving a request from a client, emulated, proxied and agnostic. In *emulated* mode upon the receipt of a valid request AmpPot will respond with a pre-defined set of protocol specific responses. There are some cases for which this not sufficient such as with DNS, which requires a dynamically generated response specific to the request. This is dealt with by recursively resolving the requested resource before sending the response. In *proxy* mode AmpPot behaves like a proxy, forwarding the requests to internal servers that actually operate the protocol. Proxy mode requires configuration of servers, for example, as an NTP-time server. This main benefit of running in proxy mode is bypassing the need for emulation of services. In *agnostic* mode AmpPot is not concerned about the validity of the request. It will reply with a response either 100 times the size of the request or with the maximum transmission unit (MTU). The response is made up of random bytes. Agnostic mode will work for attackers that are only concerned with finding serves that send large replies and are unconcerned with the validity of the reply. This mode will not work for more sophisticated botnets or skilled human attackers. For all the response modes excepting agnostic mode, the honeypot was carefully designed in order to ensure that popular clients software for each protocol would successfully parse the responses sent by AmpPot.

The AmpPot developers faced a familiar ethical issue when considering the design of the honeypot, the balance between appearing as an attractive target for attackers while avoiding becoming a part of a DDoS attack. To this end the AmpPot includes a rate limiting feature, if a client sends more than ten requests per minute, AmpPot will block this IP address for a time period so as not to be included in a reflection attack. Blacklists are reevaluated after an hour.

By their own admission AmpPot can be fingerprinted relatively easily by a skilled human attacker or well-designed botnet, particularly if all the UDP ports are listening. Configuring AmpPot to listen on just one port, however, makes it more difficult to detect. A malicious user might notice discrepancies in expected payload responses and actual responses, also leading to detection, or

observe denied requests due to the rate-limiting feature.

Overall the AmpPot appears to be an interesting honeypot project with some unique features. The researchers involved in its development obtained and published interesting results relating to amplification DDoS attacks. Although the author did not gain access to the AmpPot source code, it was possible to gain access to traffic and malware data for the IoTPOT, a later honeypot designed by the same researchers. Considering this and the fact that the researchers involved in the project are recognised in their field, it is safe to assume the information provided in the AmpPot paper is true and trustworthy.

II) IoTPOT

The IoTPOT is a high interaction honeypot designed specifically to monitor the threat posed by IoT botnets. Contact was made with the researchers who created the IoTPOT in the duration of the project. While no scripts were available, it was possible to deploy a proxy sensor and to gain access to a server for two weeks with all the data relating to the IoTPOT data procurement for that given time. We will discuss the honeypot, the deployment of the proxy sensor and the data obtained from the IoTPOT server in detail in later sections.

2.1.6 Medium Interaction Honeypots

In this section we will look at some instances of medium interaction honeypots. As stated already some authors may categorise some of these instances as low interaction.

I) Kippo

Kippo is no longer under active development but it is worth discussing here due to the influence it has had on the development of later honeypots. Kippo, written in Python, is a medium interaction SSH honeypot designed to log a malicious users' entire interaction with the system. As with most medium interaction systems Kippo is not a real operating system but a software emulation of such a system. Kippo emulates the SSH protocol logging all brute force login attempts. Kippo also emulates a shell and file system for the attacker to interact with after a successful login attempt. It emulates a Debian operating system, with a fake filesystem that allows the user to add or remove files. Configuration options allow for setting passwords that when guessed correctly will allow access to the system. The user is allowed to download but not execute files on the system. These downloaded files are saved if needed for further analysis. All login attempts are saved and all the users' interaction with the system is saved after successful login. [8]

Kippo has had a significant influence on the HonSSH honeypot and is a direct predecessor of the Cowrie honeypot. Kippo was inspired by the Kajoney honeypot. [6][8]

II) Cowrie

Cowrie is a direct successor to Kippo and is under active development. Cowrie is also a medium interaction SSH and Telnet honeypot offering all the features that the Kippo honeypot provides. It extends the shell system supporting a number of extra commands. It allows for ssh-proxying and forwarding of SMTP connections to another SMTP honeypot. It supports logging in the JSON format for easier visualistion of data obtained. The Cowrie honeypot was used for this project, both as a Modern Honey Network sensor and as a standalone honeypot on an amazon web services (AWS) Ubuntu image. We will discuss the results of these in a later chapter.

There were a number of ways in which Kippo could be fingerprinted that the Cowrie honeypot has fixed. For example echoing the SSH server banner in Kippo will lead to Kippo printing "Protocol mismatch", this will not happen on a real system with OpenSSH installed. This issue has been fixed in Cowrie. [9]

Kippo

```
$ nc -w localhost 2222
SSH-2.0-OpenSSH_5.1p1 Debian-5
SSH-2.0-OpenSSH_5.1p1 Debian-5




Protocol mismatch.
```

OpenSSH

```
$ nc -w3 localhost 22
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2
SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2
[illegible]curve25519-sha256@libssh.org...
...
```

III) Dionaea

Dionaea is another medium interaction honeypot. It is not a real system but rather like Cowrie and Kippo an emulation of one. Dionaea emulates a vulnerable Windows operating system that runs services such as SSH, HTTP and FTP. The stated goal of Dionaea is to trap malware and obtain a copy of it. In order to capture malware it uses the LibEmu library to detect and evaluate payloads sent by the malicious user. The Dionaea honeypot does not execute the obtained malware. If further analysis of the malware is desired the Dionaea needs to be used in conjunction with malware analysis tools. Like the Cowrie honeypot, a Dionaea honeypot was used for this project, both as a Modern Honey Network sensor and as a standalone honeypot on an amazon web services (AWS) Ubuntu image. We will discuss the results of these in a later chapter. [10]

IV) HoneyPy

HoneyPy is a well-maintained medium/low interaction honeypot, as you might suspect, written in Python. HoneyPy comes with a few plug ins emulating services. One of its objectives, however, is to be easily extendible by allowing users to add new service emulation for TCP and UDP by building their own plugins. An interesting feature of HoneyPy is its log handling integration with third party tools such as Twitter. While all monitored activity is saved to a log file as one might expect, posting the honeypot activity to a web end point can be easily achieved also. [11]

V) Glastopf / SNARE

Glastopf is a dynamic low-interaction web application honeypot written in Python. Glastopf is capable of emulating numerous known vulnerabilities in order to collect data in relation to attacks that target web applications. The main concept behind Glastopf is that the honeypot should be capable of replying to the attacker with the expected response to a specific exploit. Glastopf is no longer under development but is maintained by the MushMush organisation. Glastopf has been replaced by SNARE, which has all the features of Glastopf with many additional ones. SNARE allows for the conversion of existing web pages into attack surfaces. SNARE is actively maintained by the MushMush group. [12] [13]

VI) Conpot

Another honeypot developed and maintained by the MushMush organisation is the Conpot honeypot. Conpot is an Industrial Control System (ICS) honeypot designed to collect data relating to the methods and motives of malicious users. It provides the user with a range of common ICS protocols enabling the user to emulate a real ICS system in order to entice attackers. Conpot provides a central logging feature using HPFeeds which enables users to contribute to the data they collect along with numerous existing Conpot contributors. Conpot is dedicated to monitoring ICS systems only. [14]

VII) Honeytrap

Honeytrap is a low interaction honeypot daemon that is designed to observe attacks on network services. The description from the Honeytrap github is as follows “Honeytrap is a network security tool written to observe attacks against TCP or UDP services. It runs as a daemon and starts server processes dynamically on requested ports. A server emulates a well-known service by simply sending captured network traffic to a connected host.” Honeytrap development started over ten years previous to the time of this report but the github repo has recent updates and there is also a dockerised version of Honeytrap maintained as part of the Deutsche Telekom T-Pot project. [15]

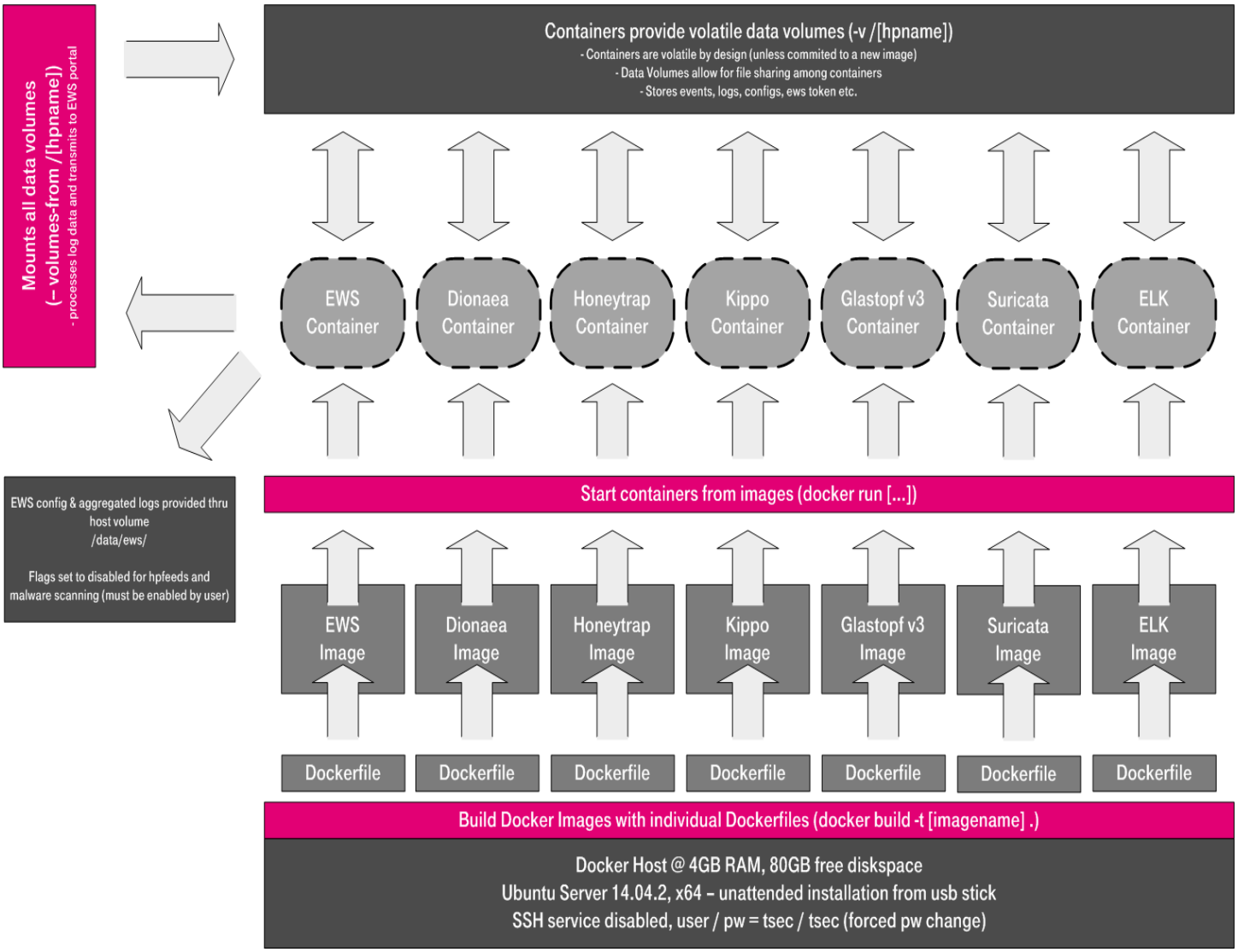
2.1.7 Additional Software

In this section we will look at some additional honeypot related software.

I) T-Pot

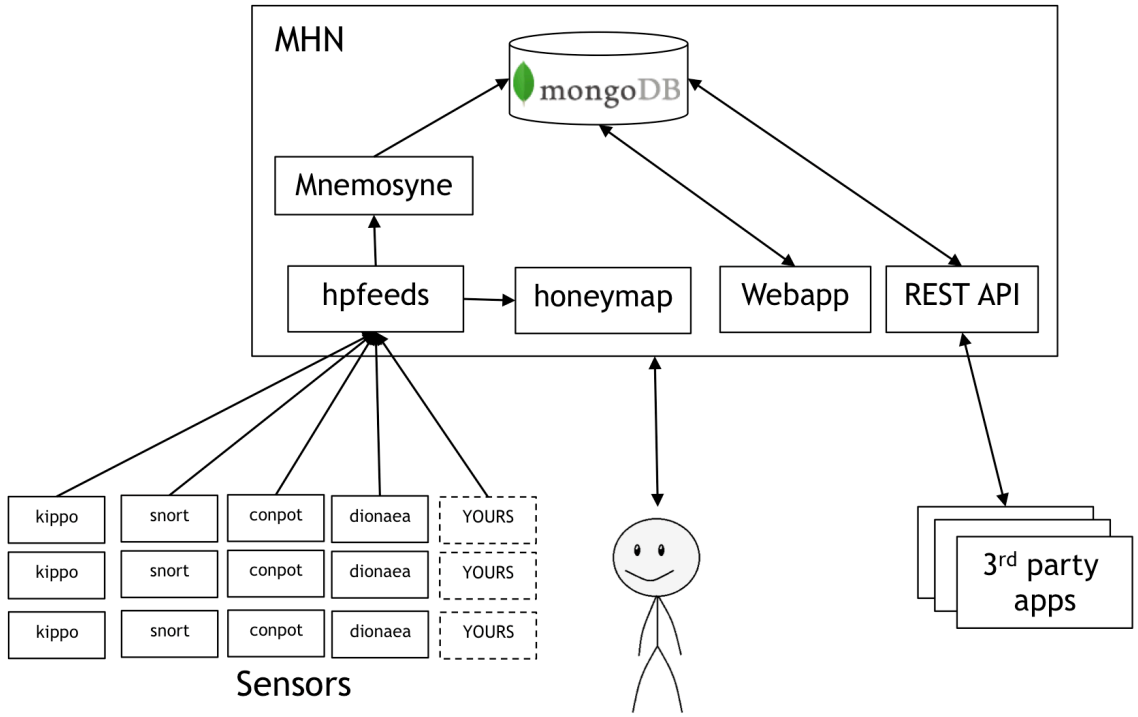
T-Pot is a honeypot solution provided by Deutsche Telekom that provides multiple docerkised versions of popular and well-maintained honeypot projects. The idea for the project arose from Deutsche Telekom's experience with deploying a number of honeypots throughout their global network as well as at their partner's locations, the data from which was used to power their early warning systems. This data was also used as the stream for their graphical and statistics gathering website Sicherheitstacho. [16] Their aim with the T-Pot project is to make the deployment of honeypot technology easier for organisations.

T-Pot uses Docker to create paravirtualized versions of honeypots such as Cowrie, Dionaea and Glastopf. Docker is an application or software container platform designed to overcome platform dependency issues. Using Docker for T-Pot allows the developers to run separate deamons of each honeypot in isolated containers. This separation of each honeypot deamon enables the developers to easily update each and ensures the overall system maintenance is kept low. The overall T-Pot system is a combination of the docerkised versions of the honeypots, the network intrusion detection system (IDS) application Suricata, T-Pots data submission tool Ewsposter and the monitoring and visualization ELK stack which we will discuss in greater detail in the design section of this report. On the next page you can see a visual representation of this architecture provided by the T-Pot developers.


Mounts all data volumes
(– volumes-from /[hpname])
- processes log data and transmits to EWS portal
Containers provide volatile data volumes (-v /[hpname])
- Containers are volatile by design (unless commited to a new image)
- Data Volumes allow for file sharing among containers
- Stores events, logs, configs, ews token etc.
EWS Container
Dionaea Container
Honeytrap Container
Kippo Container
Glastopf v3 Container
Suricata Container
ELK Container
EWS config & aggregated logs provided thru host volume
/data/ews/
Flags set to disabled for hpfeeds and malware scanning (must be enabled by user)
Start containers from images (docker run [...])
EWS Image
Dionaea Image
Honeytrap Image
Kippo Image
Glastopf v3 Image
Suricata Image
ELK Image
Dockerfile
Dockerfile
Dockerfile
Dockerfile
Dockerfile
Dockerfile
Dockerfile
Build Docker Images with individual Dockerfiles (docker build -t [imagename] .)
Docker Host @ 4GB RAM, 80GB free diskspace
Ubuntu Server 14.04.2, x64 – unattended installation from usb stick
SSH service disabled, user / pw = tsec / tsec (forced pw change)

II) Modern Honey Network

Modern Honey Network (MHN) is a somewhat similar idea to T-Pot developed by security firm ThreatStream. It is an overall honeypot management system with the aim of making it easy for ogranisations to deploy honeypots as part of their network defense infrastructure. Once MHN is installed on a system deploying one of the honeypot sensors the user need only run a script provided by MHN. The system supports a number of the honeypots we have encountered already including Cowrie, Dionaea and so on. It enables communication via HPFeeds and uses as MongoDB as its database. It allows for easy visualisation of attack data such as geolocation and type of attack. MHN was set up on an AWS Ubuntu server early in this project with Dionaea and Cowrie sensors deployed. While it proved an interesting tool in terms of monitoring attack types no malware was captured during the week it was set up. Below you can see a visual representation of this architecture provided by the MHN developers.



III) HPFeeds

HPFeeds was originally developed by the Honeynet project as a means to transfer real-time data between members of the Honeynet project from various honeypot software components. HPFeeds is a data sharing publish-subscribe protocol used by many Honeypot/Honeynet projects in order to share the data collected by multiple deployed sensors. The honeypot sensors send their collected data to a HPFeeds server which in turn forwards this data to authenticated HPFeeds clients (subscribers). HPFeeds is used by MHN and T-Pot among other existing honeynet projects as their data sharing mechanism. HPFeeds provides a secure way of sharing the data by using Authkeys or by allowing for the protocol to be run on top of SSL/TLS. HPFriends is a more recent project that is built upon HPFeeds. It uses the same wire protocol as HPFeeds but aims to make the sharing of data easier and more natural for the user. [17]

2.1.8 Exploitation & Malware Delivery

In this section we will discuss some of the ways that attackers manage to exploit a targeted device.

I) Brute Force Authentication

Brute force authentication is a technique commonly used by botnets in order to gain access to a vulnerable system. It is the technique used by the Mirai botnet. Many systems allow remote access to a system via the SSH protocol. In many cases to access a system remotely via SSH the user need only supply a username and password. If the system has not been configured in a professional manner there will be no limit on the number of attempts the user has to gain access. In such a scenario a malicious user may attempt as many username / password combinations as she wishes. Brute force login attempts are often carried out using a lookup dictionary of well-known username / password combinations.

The predecessor to the SSH protocol was the Telnet protocol. Telnet does not encrypt traffic enabling attackers to easily capture authentication details. SSH was developed as a more secure solution to remote system access and it is generally considered wise to disable Telnet on a system for security purposes. Telnet is, however, still used by many IoT devices with an estimated 16 million internet-connected devices accessible via Telnet. [https://securityintelligence.com/telnet-an-attackers-gateway-to-the-iot/] Mirai, Hajime and other IoT botnets scan for systems running Telnet in order to brute force their way into the system.

Ideally a system should disable Telnet and use SSH if remote access is required. Other techniques to improve the security of the system would be to require two-factor authentication or the use of a cryptographic key pair as is the case with AWS instances. At the very least a system should be configured to place a limit on the number of potential login attempts.

II) Social Engineering

At the time of writing numerous systems throughout the world, including Telefónica in Spain and the NHS in the United Kingdom, recently suffered from a ransomware attack. One of the primary techniques for spreading this specific ransomware was through links sent via email to unsuspecting victims. Tempting victims into clicking on a malicious link via email is a form of social engineering. Social engineering in this context is the targeting of people in order to propagate malware and infect devices. There are many forms of this type of exploitation ranging from attempting to extract information in person or via telephone to using torrents sites as a way of delivering a disguised payload. While these types of attacks usually garner most success when targeting people with limited knowledge of the field, sometimes there can be surprising victims of such attacks. In April 2017 both Facebook and Google fell victim to a phishing attack temporarily costing the companies as much as $100 million. [18]

III) Software vulnerabilities

Finding and exploiting a vulnerability in a piece of software is a common technique used by malicious users for spreading malware. This type of exploitation involves the malicious user reverse engineering the software in question or in the case of open source software, examining the source code in an attempt to discover potential weaknesses that may be exploited. This requires a level of skill and knowledge from the attacker's perspective. Software companies continuously preach the importance of installing newly released updates, these updates are often ones addressing a known security flaw in the software. Referring again to the WCry attack, Microsoft had released a patch for this vulnerability prior to the attacks in May 2017. Any Windows operating system with the patch installed was not vulnerable to this exploit. Researchers and "white-hat hackers" often attempt to discover vulnerabilities themselves for research purposes as well as for financial rewards. Companies such as Google and Facebook run bug bounty schemes with financial rewards varying depending on the importance of the vulnerability discovered. [19]

The discovery of a previously unknown exploit in a widely used application can lead to a proliferation of infected systems over a relatively short time period. The security company

FireEye provides interesting and detailed analysis of some recent zero-day exploits they have encountered in the wild. [20]

IV) Hardware Backdoor

[21] discusses the security risks of companies using third party testers in the chip development process, potentially leading to the introduction of a hardware backdoor to a system.

2.1.9 Issues & Tradeoffs

In the levels of interaction section we touched briefly on some of the issues associated with the various levels of deployment. There are certain issues, advantages and disadvantages related to each level that a researcher needs to take into account when choosing a honeypot for a particular task. For example, a researcher looking to observe authentication attempts by the Mirai botnet need only employ a simple low interaction honeypot such as MTPot, which is a honeypot dedicated to this specific purpose. A researcher looking for an in depth analysis of how a particular botnet operates will need to look at employing a high interaction honeypot solution. The choice of a high interaction honeypot for deployment, however, is accompanied by certain risks. We will look at these logistical, ethical and legal issues in the design chapter, while discussing the potential tradeoffs involved.

Another issue that needs to be considered and contended with is what is known as fingerprinting. We discussed this briefly also earlier in the chapter, fingerprinting is the scenario whereby an attacker, automated or otherwise, is able to detect that the system she is interacting with is not in fact a real system but a honeypot. Fingerprinting is not always an intentional act by the botnet, it can often occur when the botnet does not receive the expected (or any) response to a certain command. This will regularly lead to the botnet exiting the system. The honeypot deployer needs to be aware of this issue, particularly when deploying a medium interaction honeypot, and take steps to counteract it where necessary.

2.2 Botnets

A botnet is a network of compromised devices that have been infected with malware, which is controlled by a single owner, often referred to as the bot-master or bot-herder. This network of compromised devices is used by the bot-master to carry out malicious coordinated tasks. There are a number of functions a botnet can serve. Some of the more common reasons include DDoS attacks, spam and information theft. We discuss these among other purposes in greater detail in the next section. Botnets can be designed to target various systems, from ICS devices to standard personal computers. As discussed in chapter three and various other sections of this report, recent botnets are increasingly targeting vulnerable IoT devices. Botnet designs are constantly evolving, particularly with the release of code online, as we noted with the Mirai botnet, it allows other malicious users to deploy the botnet as is, as well as leading to variations of the original one. The two main types of botnet architecture are centralised command and control and peer to peer. We will discuss these in more detail in a later section.

2.2.1 Purpose of a Botnet

Botnets are deployed by users for various reasons, commonly with the intention of carrying out some form of malicious activity. In this section we will look at some of the common reasons for botnet deployment. Botnets can have a significant detrimental impact on an economic sector. Taking as an example the digital marketing sector, according to the Association of National Advertisers and security firm White Ops, in 2016 advertisement fraud was expected to cost marketers more than $7 billion. [22] In this section we will detail some of the current common uses for botnets.

2.2.1.1 Distributed Denial-of-Service (DDoS) Attacks

We have already discussed some high profile examples of DDoS attacks. They are a common method of attack deployed by botnets. DDoS attacks can be divided broadly into three categories of attack, volume based, protocol and application layer attacks. DDoS attacks are normally carried out at the transport layer of the OSI model (UDP, TCP) or at the application layer. In the following section we will discuss specific types of DDoS attacks.

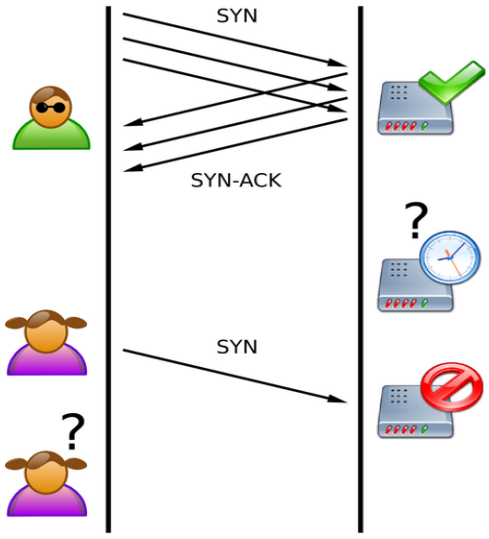
I) HTTP Flood

HTTP Floods are a commonly deployed method of DDoS attack in which the attacker uses apparently valid HTTP requests in order to overload the target server. The two types of HTTP request methods typically used when a client connects and communicates with a server are GET and POST methods. A HTTP GET request is sent in order to obtain static such as an image, while a HTTP POST request is used to access dynamically generated content provided by the server. There is a tradeoff from the attacker's perspective in terms of which type of method to use when initiating an attack on a server. The POST method requires more processing at the server side and as a result is ideal in terms of overloading the server's resources, however, GET attacks are easier for an attacker to create and scale as the Botnet grows.

HTTP Floods are difficult to detect and defend against as they use standard URL requests, do not employ any of the common reflection attack techniques and do not use malformed packets. This makes it difficult to differentiate traffic generated by an HTTP attack from genuine traffic attempting to connect with a server.

II) SYN Flood

The SYN flood attack is a common method of DDoS attack that exploits the standard TCP three-way handshake, the basis of every TCP connection. With a legitimate three-way handshake we have three steps, the client sends a SYN message to the server requesting a connection be established, then the Server responds to the client with a SYN-ACK message, and finally the client replies to this with an ACK message and the connection between client and server is now established. The diagram below illustrates what occurs during a SYN flood attack. The malicious user keeps sending repeated SYN packets to every open port on the server. The server, believing the requests to be valid, responds to each with a SYN-ACK packet. The server, however, does not receive the expected ACK packet from the client, as the client either chooses not to send this ACK, or if the original IP address was spoofed, does not receive the SYN-ACK. In both cases the server is left waiting for an ACK packet that will not arrive. While the server is waiting for the ACK response the connection stays open, and attacker sends another SYN packet before the connection times out. The result of this flood of SYN requests to the server is a growing number of open connections. As this continues the server 's resources are consumed and the server is no longer capable of providing normal service to legitimate clients. [23]



SYN Flood Attack

III) Generic Routing Encapsulation Attack (GRE)

Generic routing encapsulation (GRE) is a tunneling protocol developed by CISCO systems. The protocol "specifies a means of performing encapsulation of an arbitrary network layer protocol over another arbitrary network layer protocol". [24] GRE essentially allows two peers to establish a direct point-to-point connection, enabling the peers to share data directly via a virtual GRE tunnel between the two nodes. In order to create this tunnel the payload on the sender side is encapsulated in the a GRE packet and not decapsulated until it reaches the other node. This allows the two connected peers to share data that they would not be able to share over a public network.

A large portion of the Mirai attack on the KrebsOnSecurity site in September 2016 was in the form of traffic designed to appear as if it were GRE data packets. Up until this point in time, that level of attack coming from GRE was very unusual, particularly at the volume recorded in this specific attack. The diagram below illustrates how the Mirai botnet used GRE to flood the target, encapsulating 512 bytes of random UDP data in the GRE packet and sending this to the victim, thus placing a strain on the target's resources, particularly at the scale of traffic involved.

It's also worth noting that GRE traffic cannot be faked in the same way as we discussed previously with DNS traffic. The implication of this is that this attack was carried out using an extremely large number of devices, mainly IoT devices.

IV) Internet Control Message Protocol (ICMP) Flood / Ping Flood

A Ping or ICMP flood is an attack whereby the attacker overruns the server with numerous ICMP echo requests (pings). The attacker pings multiple servers, faking the target's IP as the source IP address. This leads to the servers replying with a ping echo response packet to the target, consuming bandwidth and CPU cycles. To execute an ICMP attack the attacker needs to know the IP address of the target. ICMP attacks can be broadly divided into three categories. An attack on a single device in a local network, in this case the attacker is already aware of the device's IP address. An attack on a router in an attempt to disrupt communication between devices on the network, in this scenario the attacker is already aware of the internal IP address of the router. Lastly, a blind ping flood requires the use of a program to detect the IP address of the victim before initiating the attack.

V) User Datagram Protocol (UDP) Flood

The final type of DoS flood attack we will reference here is the UDP flood. Unlike TCP the UDP protocol is a connectionless protocol that requires no initial handshake like the three-way handshake used in TCP. This can make UDP useful as an alternative with comparatively less overhead. This lack of an initial handshake can, however, make UDP more vulnerable to exploitation by malicious users. A significant amount of UDP traffic can be sent to a target device without any checks to establish the validity of this traffic. In a UDP flood the attacker targets random ports on the victim's device, sending IP packets containing UDP datagrams. The receiving device will then check for applications associated with the sent datagrams and reply with a "Destination Unreachable" packet upon finding no application to deal with the sent packet. If the number of packets sent and received grows to a large number the resources of the victim system will be overwhelmed and the system will cease to operate as expected.

VI) Reflection / Amplification Attacks

A reflection attack is one in which an attacker sends a certain type of request to a number of devices with a spoofed source address of the target, leading to multiple responses to the target. The attacker's aim is to overwhelm the target with these responses. There are a variety of well-known reflection attacks many of which use amplification techniques also to increase the impact of the attack.

A Domain Name System (DNS) amplification attack exploits known vulnerabilities in DNS servers whereby the attacker is able to elicit a large response to a relatively small query. The reflection component of a DNS attack is achieved by the attacker sending a DNS query, with the spoofed IP address of the victim, to an open DNS resolver. An example of such a DNS attack would be an attacker requesting as much zone information as allowed in order to amplify the DNS record response to the target.

A Network Time Protocol (NTP) attack is another type of reflection attack that employs amplification techniques in order to overwhelm the victim. A NTP server includes a facility for a server administrator to request the traffic count using the "get *monlist*" command. This command will receive a response with the list of the last six hundred hosts that connected to the server. In an NTP attack, the attacker will send the "*get monlist*" query to the NTP server with a spoofed IP address of the target, which will receive the much larger response.

A Simple Network Management Protocol (SNMP) attack works out under the same premise as the two previously described attacks. The attacker sends out a number of SNMP queries with the spoofed IP address of the target leading to the target been overrun with responses from the queried devices.

The AmpPot honeypot we described previously was designed specifically to monitor the threat of amplification based attacks. [25] [26]

It is worth noting that the Mirai botnet attacks discussed in this report did not rely on reflection or amplification techniques. The level of traffic was generated due to the number of compromised devices involved.

VII) IP Fragmentation Attack

IP fragmentation attacks exploit known vulnerabilities in datagram fragmentation mechanisms. ICMP and UDP fragmentation attacks occur when the attacker sends fake ICMP or UDP packets that are larger than the target network's maximum transmission unit (MTU). The target will attempt to reassemble the packets but will be unable to as the packets are fake, this can lead to an overwhelming of the victim's resources. TCP exploit the mechanisms involved in reassembling a fragmented datagram in order to stop the correct reassembly of datagrams at the target server. This can lead to an overlapping of datagrams and a strain on the target's resources.

VIII) Multi-Vector Attack

Multi-Vector DDoS attacks are attacks which combine a number of DoS attack types in a effort to overcome a system's defense mechanisms. Multi-Vector attacks can be made up of simultaneous attacks at the network layer and application layer. They can be difficult for an organisation to defend against as the attacks may target a number of various network resources. In a multi-vector attack the attacker may also initiate decoy attacks in order to inflict greater damage with another type of attack. In a 2016 report by Neustar the percentage of DDoS attacks that were a combination of two or more types was estimated to be 52%. This figure is expected to continue to grow in the coming years. [27]

2.2.1.2 Information Theft

If a bot-herder is interested in retrieving personal information from an infected device she may use a *traffic sniffing* or *keylogging* in order to achieve this. A botnet may deploy packet sniffing software on an infected device and use this to monitor outgoing and incoming traffic. This can be useful for finding any information that has not been encrypted. For example, if the user of the device enters a user name and password on a site that uses the HTTP protocol and not HTTPS, the packet sniffer can be used to retrieve the details entered. If a device has been compromised by more than one botnet, packet sniffing can also be used to gather information on the other botnet and if desired used to infiltrate the other botnet.

In the password scenario mentioned above, the packet sniffing software is useful as the user is using an insecure protocol. Often, however, this is not the case and this can render the packet sniffer redundant in efforts to gather sensitive information. In such a scenario a botnet may deploy keylogging software. This will allow the attacker to monitor all of the victim's keyboard input. The attacker may then use this in conjunction with custom filters in order to obtain the information she is interested in acquiring. An attacker may also wish to obtain files stored locally on the device. There are various reasons for this, from hoping to discover personal information relating to the devices' owner to retrieving digital currency such as bit coin.

2.2.1.3 Advertising Fraud

As mentioned in the overview of this section, advertisement fraud due to botnets has a significant financial impact on the sector. This type of fraud exploits the fact that income from advertisement on a website is relative to the number of clicks the advertisement receives. A malicious user can deploy a botnet to automatically click the advertisements in order to fake this data. The botnets designed for this purpose can be sophisticated and the fraud is difficult to detect. The source of the traffic will be from real infected devices in disparate locations, will click on other sections of the website and as such is difficult to differentiate from human traffic. The cost of this to the industry is an indicator as to how difficult it is to detect this type of fraud.

2.2.1.4 Ransomware

Just before the completion of this project the National Health Service in the United Kingdom suffered an attack from a new strain of Ransomware known as WannaCrypt. It was believed at the time of writing that WannaCrypt was spread using P2P exploitation of SMB. The exploitation had been leaked as part of the Wikileaks release of classified NSA data in early 2017. Spreading ransomware is a previously observed motivation for botnet deployment. The P2P Gameover Zeus botnet was used to spread Cryptolocker ransomware. [28]

2.2.1.5 Banking Fraud

The Gameover Zeus botnet mentioned in the previous subsection was also used for banking fraud purposes. Gameover Zeus uses a type of "man in the middle" attack adapted for a web browser in order to intercept a victim's transactions during an online banking session. "Once an infected user visits their banking website through a compromised computer, Gameover intercepts their online session using a technique commonly known as man-in-the-browser (MITB). It can bypass two factor authentication and display fraudulent banking security messages to the user to obtain information for transaction authorization. As soon as the attackers get these details, they can modify the users' banking transactions and steal their money." [29] Another example of a recent botnet used for banking fraud purposes is the Dridex botnet. [30]

2.3 Botnet Architecture

The first botnets employed a centralised architecture with one command and control server communicating with and directing many bot clients. There has been an evolution in botnet architecture as attackers attempt to increase the difficulty of tracking and shutting down deployed botnets. This has seen the development of botnets that move beyond a more straightforward centralised architecture. In this section we will look at the three main types of architecture, centralised, semi-distributed and peer-to-peer. [31]

I) Centralised

In a centralised architecture there is usually one command and control server communicating with multiple infected clients. The server is used by the bot-master as a way of delivering commands to the network of compromised devices. The first botnets used the Internet Relay Chat (IRC) protocol for communication. As botnet development evolved many centralised botnets have adapted to use the HTTP protocol. The advantage of using HTTP is that it is more difficult for a target to differentiate between traffic generated by the botnet and expected legitimate traffic. IRC traffic is more easily identified and filtered by blocking the IRC port with a firewall.

An IRC botnet operates in the following manner. A compromised target establishes a connection to an IRC C&C server, joining a specific channel that has been chosen by the bot-master. The bot-master may then deliver commands to the botnet via this channel, with each command message broadcast to all members of the channel. The compromised client will respond to commands on the same channel.

The obvious problem with a centralised architecture from the attacker's perspective is the fact that there is a single point of failure. If the command server is discovered the botnet is easily shut down. It is also straightforward to map the whole network once the command server is known, as all traffic streams directly from the C&C server to the network of compromised devices. In order to circumvent this issue botnets with a peer-to-peer architecture were developed.

II) Semi-Distributed

Semi-distributed architectures were developed as a reaction to issues with centralised architectures, namely, the issue of one point of failure. The idea is to operate with a client / server architecture but instead of operating with just one CnC server the botnet will operate with multiple servers. The obvious advantage of this strategy is, the takedown of one CnC server does not shut the entire botnet down. If one C&C server is shut down, the bots will communicate with another C&C in its place. Semi-distributed architectures may also implement a domain name generation algorithim (DGA) in order to become more resilient to take down attempts. The idea is to use the DGA to calculate the CnC server hostnames, using deterministic inputs such as the current date (new strains of the Mirai botnet employ this DGA). This leads to a scenario whereby a defender attempting to take down the botnet must consider a multitude of potential domains that may be hosting the server. The defender may need to register hundreds of frequently changing domains in order to disrupt the botnet. [31]

III) Peer-to-Peer

P2P networks are distributed networks in which nodes of the network communicate with other nodes on the network directly, without the need for a centralised server to coordinate the nodes actions. Nodes in a P2P network act as both a server and client, supplying data to other nodes as well as receiving data from other nodes in the network. P2P networks were popularised in the late 1990s by distributed file sharing systems such as Napster. Popular modern day file sharing systems such as BitTorrent use a P2P protocol. There are two main types of P2P botnet architecture, structured and unstructured.

Structured P2P

Structured P2P botnets are similar to popular existing P2P networks such as BitTorrent. Structured botnets use what is known as a Distributed Hash Table (DHT) for routing purposes. A DHT is similar to a normal hash table, essentially it is a look up service with key, value pairs stored in the DHT, if a node in the network wishes to retrieve a value it uses the associated key to look this value up. In structured botnets the bot-herder will store commands in the DHT. Nodes on the network may retrieve this information periodically from other peers on the network. Individual bots may also store information to the DHT, such as personal information taken from a compromised device.

Unstructured P2P

Unstructured P2P botnets do not use a DHT for storing or retrieving information. The type of communication in an unstructured P2P botnet is known as gossiping. Unstructured botnets may use a push-based or a pull-based system for communication. Nodes in the network communicate directly with the peers in their peerlist, propagating commands in this manner. In a push-based system individual bots forward commands received from neighbor peers to other peers in their peerlist. In a pull-based system bots periodically request commands from their peers. Commands in push-based networks usually propagate quicker than on pull-based systems as the nodes do not need to wait for a request to send commands on.

Hybrid P2P

Hybrid P2P botnets use aspects of P2P architectures and centralised architectures. The motivation for this type of architecture is to take advantage of the resilience to take downs offered by P2P architectures while using aspects of a centralised architecture to enable easier control of the botnet.

2.3.1 Botnet Examples

In this section we will describe briefly a high profile botnet operating with a centralised architecture and two high profile botnets operating with a P2P architecture.

I) Mirai

The Mirai botnet uses a centralised architecture. We will discuss the Mirai botnet in more detail in the next chapter.

II) Zeus P2P / Gameover

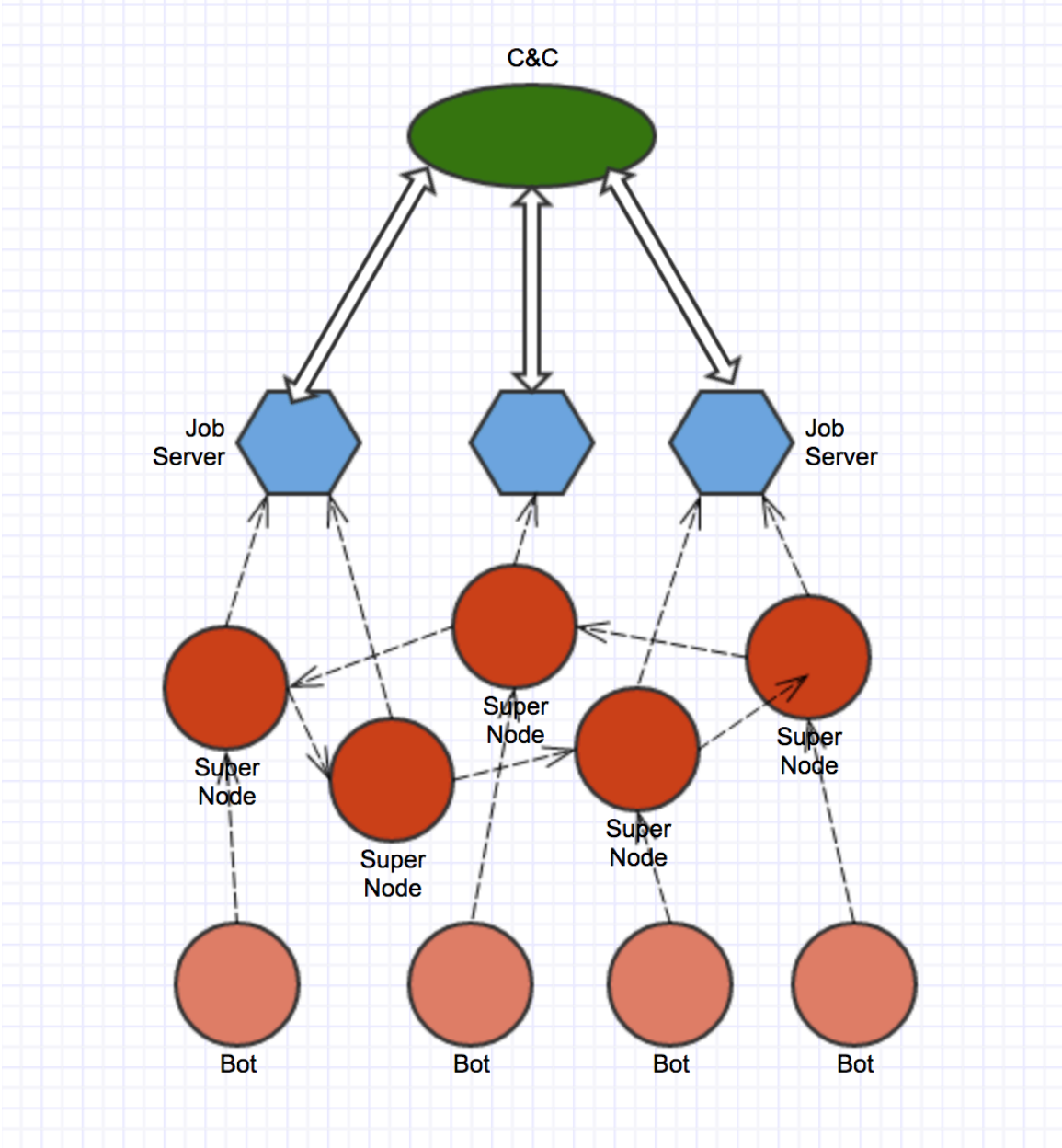
The original Zeus botnet had a centralised architecture. This was later adapted after the source code became available online, in August 2011, to a P2P architecture in order to become more resilient to take down. Zeus P2P uses an unstructured network. Individual nodes use gossiping to push dropzone location for data obtained from compromised devices. In order to gain access to the most recent updates to configuration files peers pull files from peers in their peerlist. [32] gives a detailed analysis of the Zeus botnet's operation.

III) Storm

The Storm botnet was first detected in 2007. Storm targeted PCs running Windows as their OS. Storm was built on a structured P2P network, using the existing Overnet protocol. Storm used a DHT to enable peers to look up commands, stored by a bot-herder, by an ID.

IV) Kelihos

Kelihos is an example of a hybrid P2P botnet. The diagram on the following pages gives an overview of the architecture of the Kelihos botnet. The diagram shows the four types of actors participating in the operation of the Kelihos botnet. We have the C&C server that controls the overall operation of the botnet. The Job servers are proxy servers used to relay requests to the actual C&C server. The Supernodes are infected devices capable of accepting direct incoming connections and communicating directly with other Supernodes. The Bots are equivalent to infected devices in a typical centralised architecture. When a device is infected with the malware it will have access to a list of Supernode IPs hardcoded in the Kelihos binary. The bot will establish a connection with a Supernode and they will exchange lists of Supernodes adding any new entries to their current list in order to maintain an up to date list of live Supernodes. In order to obtain attack instructions or any particular type of instruction the bot will send an online Supernode a job request over the HTTP protocol. The Supernode will forward this job request to one of the Job servers and upon receipt of a response the Supernode will relay this back to the bot. [33]



Kelihos Hybrid P2P Architecture

V) Hajime

Hajime is a structured P2P botnet dedicated to infecting IoT devices. We will look at the Hajime botnet in the following chapter.

2.4 Summary

In this chapter we looked at the characteristics of honeypots. We reviewed the terminology, interaction levels and categories of deployment. We detailed examples of a specific research honeypot as well as looking at a number of currently well-maintained open source honeypots. We gave an overview of botnets including some of the important purposes of them. We examined various types of DDoD attacks, looked at common types of botnet architecture and looked at a few specific examples. As the reader may have observed, the subject of botnets, and honeypots by extension, is a broad field with many interesting facets. It is also an ever evolving, particularly fast-paced field. In the next chapter we will look at the Internet of Things, an area that has become of particular interest in recent times. We could have placed this as a subsection of this state of the art chapter, but as it became a main focus of the project, it seems sensible to place it in a standalone chapter.

Chapter 3

Internet of Things

In October 2016 the *KrebsOnSecurity* website, a site authored by renowned American investigative journalist Brian Krebs dedicated to investigating network security and cyber crime, fell victim to a DDoS attack. The attack was one of the largest known attacks in internet history, with traffic analysis measurements putting the attack at 620Gbps. To put this into context, the largest attack the security company protecting the site, Akamai, had seen in the year previous, 2015, measured at 363Gbps. [34]

This huge attack on the *KrebsOnSecurity* site was achieved without deploying any of the amplification techniques usually associated with large scale DDoS attacks. A typical example of such amplification techniques is DNS amplification. This is a type of reflection attack in which the attacker sends a DNS query with the intended victim's forged IP address to an open DNS resolver, this leads to a DNS response being sent to the victim's IP address. This attack can then be amplified by the perpetrator creating DNS queries that will lead to much larger responses. Using certain techniques a DNS request message of just 60 bytes can be crafted to receive a response message of over 4000 bytes to the victim's address, an amplification of 70:1. [35]

This October 2016 attack, however, did not need to rely on such amplification techniques. This attack used SYN floods, GET and POST HTTP floods and a technique that uses generic routing encapsulation (GRE) data packets, to generate the level of traffic already referenced. We discussed these types of attacks along with others in greater detail in a previous section. The important thing to note about these types of attacks, in this scenario, is that the traffic cannot be faked in the same manner as occurs in DNS amplification attacks. This attack was powered by a compromised network of IoT devices, the exact number of which was unknown, but estimates suggest possibly hundreds of thousands. [34] Whatever the exact number, to generate such traffic

we can safely say it took, until recently, an unprecedented number of infected devices.

When we refer to IoT devices we are referring to anything from wireless routers to IP cameras, to smart thermostats. They make for ideal targets for attackers for a number of reasons. Unlike most PCs they are permanently online, they have global IP addresses, they often have weak login credentials that are easily cracked, quite often using default passwords, and they have no defense software such as anti-virus installed. They are, in many ways, an attacker's dream. [36]

In October 2016 the U.S. government agency, computer emergency readiness team (CERT) also published a warning relating to vulnerabilities in IoT devices. Among some of the suggested preventative steps, were suggestions to disable the vulnerable universal plug and play (UPnP) protocol on routers and to monitor TCP port 23 (Telnet) for unauthorised log in attempts. [37]

In February 2017, Gartner estimated that 8.4 billion IoT devices will be in use worldwide in 2017, a rise of up to 31 percent from the previous year, with total spending on endpoints and services reaching almost $2 trillion. Gartner estimated that the number of IoT devices in use world wide will reach 20.4 billion by the year 2020. [38]

3.1 IoT Botnets

The *KrebsOnSecurity* DDoS attack discussed in the previous section was carried out by a botnet known as the Mirai botnet. Mirai was first detected by the whitehat malware research group, MalwareMustDie, in August 2016. [39] The source code for Mirai was uploaded online in October 2016 to *Hackforums*, a hacking community site by a user with the pseudonym Anna-senpai. [40] An analysis of the source code provided an overview of how Mirai operates. We will go through this in detail in the next section. In the previous chapter we looked at the Hajime botnet, which as we observed is also a botnet that targets IoT devices.

3.1.1 IoT Botnet Case Study - Mirai

We have discussed the Mirai botnet previously in this report. We have referenced some of the significant attacks for which Mirai was used. In this section we will look at the design and operation of Mirai, referencing the source code that was uploaded online last year.

I) Scanning

The original Mirai targets devices running the Busybox operating system, which is a reduced version of Linux used on embedded devices. The Mirai source code contains a scanner file written in c. The scanner is used to scan a range of public IP addresses while using SYN scanning (half open scanning) to check if a device is vulnerable. The scanner generates a random IP to target making sure to avoid certain IP ranges such as the Department of Defense and US postal services. The SYN scanning works in the following way, the attacker will send a SYN packet to multiple ports on the target, if the port is open the target will respond with a SYN-ACK. The attacker responds to this SYN-ACK with a RST packet, not completing the handshake but closing the connection. Not completing the interaction usually avoids the interaction being logged by any logging application on the target device. Mirai will send a packet to port 23 (TELNET) to test if it's open on the target device. After these steps are complete Mirai will establish a TCP connection with the target device. In the scanner file we can see the hardcoded passwords in hexadecimal form in the "add_auth_entry" function. Mirai uses these passwords when prompted for login credential, using a brute force dictionary method. As an example the following is used to test the user name "admin" combined with the password, also "admin".

"add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7); "

This combination, along with others found in scanner, were regularly encountered in the honeypots deployed during this project, which we will discuss in detail in the evaluation section. If the attacker manages to connect to the target device it will report the new infected device to a

server, listening on port 48101, with the target's IP address, the port number and the authentication credentials. This is the scanListen process which stays listening on port 48101. The server will then send the details of the device to the Loader that will load the malware to the newly compromised device.

II) Loader

The loader code is also written in C. The loader uses *wget* to upload the malware to the device. In the case that *wget* is not supported by the version of *Busybox* the TFTP will be used instead to obtain the malware. After the malware has executed on the target device it will delete itself to avoid detection but the process will stay running. Telnet is also disabled after the device has been infected, presumably to prevent infection attempts by other attackers. Once the target device has been infected with the malware it will also start scanning for new devices to infect.

III) C&C

To initially establish a connection to the CnC the compromised device resolves a domain name, connecting to this IP. The connection to the C&C server from a client/bot is established on port TCP/23. If a TCP connection is established the IP of the bot establishing the connection is added to the clientList, this device is now a part of the botnet. The C&C interface is written in the GO language. Commands may be issued from the admin interace. From here it is possible to manage all the clients, monitor their state, send commands directly to individual bots and to initiate attacks.

IV) Attack Vectors

Mirai has the capacity to initiate a number of DDoS attack types after compromising a network of devices. The attack code is written in the google language GO. Most of the attack types provided by Mirai we have already discussed in the previous chapter. The list of attack types is, SYN flood, UDP/UDP plain flood, DNS resolver flood, SYN flood, ACK flood TCP stomp flood, GRE IP/Ethernet flood and HTTP flood.

3.1.2 Conclusion

In this section we've looked at the operation of the original Mirai botnet. Since the release of the original source code a number of variations of the original have been found. Modifications include changing the attack port from 23 to 7547 or 5555. [41] Another notable change was the inclusion of a domain name generation (DNG) algorithim whereby a new domain for the infected device to interact with the C&C is generated periodically. The Mirai code also includes a link to the following youtube video in a comment, the meaning of which I've yet to decipher. https://youtu.be/dQw4w9WgXcQ

3.1.3 Hajime Botnet

The Hijame botnet is an IoT botnet that operates in a very similar manner to Mirai in the early stages. Like Mirai it scans for Telnet devices, using brute force dictionary methods to gain access once a device has been discovered. The network topology, however, is different and more sophisticated than Mirai. The Hajime botnet creates a structured P2P network using BitTorrent's DHT protocol for peer discovery and uTP for data exchange. [42]

In April 2017 there was estimated to be roughly 300,000 devices in the Hajme botnet. [43] Hajime has become known as the "Vigilante Botnet". The reason for this is Hajime has not been known to be used for any malicious purposes, but instead it gains access to a vulnerable IoT device and shuts down the ports that make the device vulnerable. Hajime does not contain any attacking capabilities beyond its propagation abilities. The following message is displayed periodically on infected devices "Just a white hat, securing some systems. Important messages will be signed like this! Hajime author. Contact CLOSED. Stay sharp!" [44]

3.2 IoT Honeypots

The creation of botnets designed specifically to target IoT devices has naturally led to researchers attempting to find a way to monitor and examine these botnets effectively. A number of medium interaction honeypots were created as a direct reaction to the Mirai botnet such as MTPot, created by Cymmetria, a network security start up. [45] Well-maintained SSH/Telnet honeypots such as Cowrie can also be used for this purpose. We will not look at a research honeypot designed specifically with this threat in mind.

3.2.1 IoT Honeypot Case Study - IoTPOT

IoTPOT is a novel honeypot created by researchers at Yokohama National University, Japan, and Saarland University, Germany. The IoTPOT is a honeypot designed specifically to track and monitor IoT botnets by emulating the interactions of the Telnet protocol and various IoT devices. [36]

During the course of the project I made contact with the researchers involved in the IoTPOT project. Although they were unable to share the scripts for the honeypot at this point in time, they kindly shared their data sets with me. This data included unique malware binaries that they were able to procure with their honeypot and a record of incoming and outgoing honeypot traffic contained in pcap files.

The IoTPOT team also gave me the opportunity to deploy a proxy. The issue that arose with the deployment of the proxy sensor in this case was one of time and expense. The IoTPOT team required that the sensor be deployed for at least a period of one year. With the amount of disk space required to run the proxy (>=64GB) this would have left the author maintaining the proxy, long after the end of the project, at a personal cost. Unfortunately this made the deployment unfeasible. The data provided, however, from the research team's active honeypots was significant and played an important role in analysing threats along with the honeypots deployed

during the project. This data will be discussed later in the report in the evaluation chapter.

In this section we will discuss the design of the IoTPOT along with the findings published by the researchers.

Design & Operation

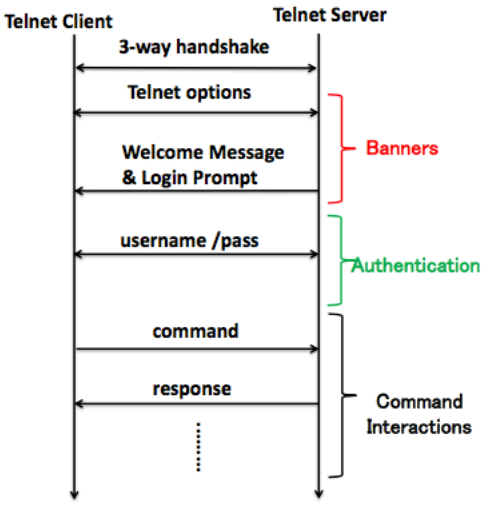
The IoTPOT as described by the developers is “a novel honeypot that emulates interactions of the Telnet protocol and a variety of IoT devices”. The goal of IoTPOT is to attract attackers, monitor their interactions with the system and obtain malware from these attacks. An important aspect of the IoTPOT is that it is designed to be a dynamic honeypot in the sense that it is able to emulate the behaviour of multiple embedded devices and to adapt to what the attacker expects, from initial greeting to responses to commands once access to the shell has been gained. Below we have the diagram of the Telnet protocol provided by the researchers in their paper. They split the interactions involved in the Telnet connection process into three distinct sections. They refer to the initial connection establishment followed by login welcome message as the *banner interaction* phase, the sending of the username and password as the *authentication* phase and lastly the interaction between the client and the system as the *command interactions* phase. Clearly to perform the banner interaction phase in a dynamic manner as opposed to one generic response to any connection attempt, the honeypot needs to be able to provide the correct initial welcome message. An example given by the developers is “BCM96318 Broadband Router “. In order to achieve this the group collected appropriate banners from the internet using Telnet scans with the Mascan tool. This step ensures that the banner interaction phase was capable in most cases of providing the attacker with the appropriate welcome message.

In the authentication process phase the IoTPOT may be configured in a variety of ways that one would expect for a well designed honeypot, similar to honeypots like Cowrie and Kippo the IoTPOT may be configured to accept specific usernames and passwords only, to accept any credentials or to reject the first number of attempts but eventually allow login after a certain number of attempts.

In the final command interaction phase the *Frontend Responder* of the IoTPOT will reply to the attacker’s commands with the appropriate response, however, if the command is not known it is

forwarded to a backend known as IoTBOX which emulates a number of embedded Linux operating systems for various CPU architectures. These means that the IoTPOT can handle the interactions expected of a number of different CPU architectures. This behaviour differentiates it from other honeypots I have looked at through the course of the project.

Telnet Protocol



Telnet Protocol Diagram [36]

IoTPOT Architecture

The diagram below, provided by the researchers in the referenced paper, shows the architecture of the IoTPOT. The two main components are the *Frontend Responder* and the *IoTBOX* . We also see the *Profiles* section, each device has a related profile that determines the appropriate interactions and responses for the device, split into the three phases of the Telnet interaction we mentioned already.

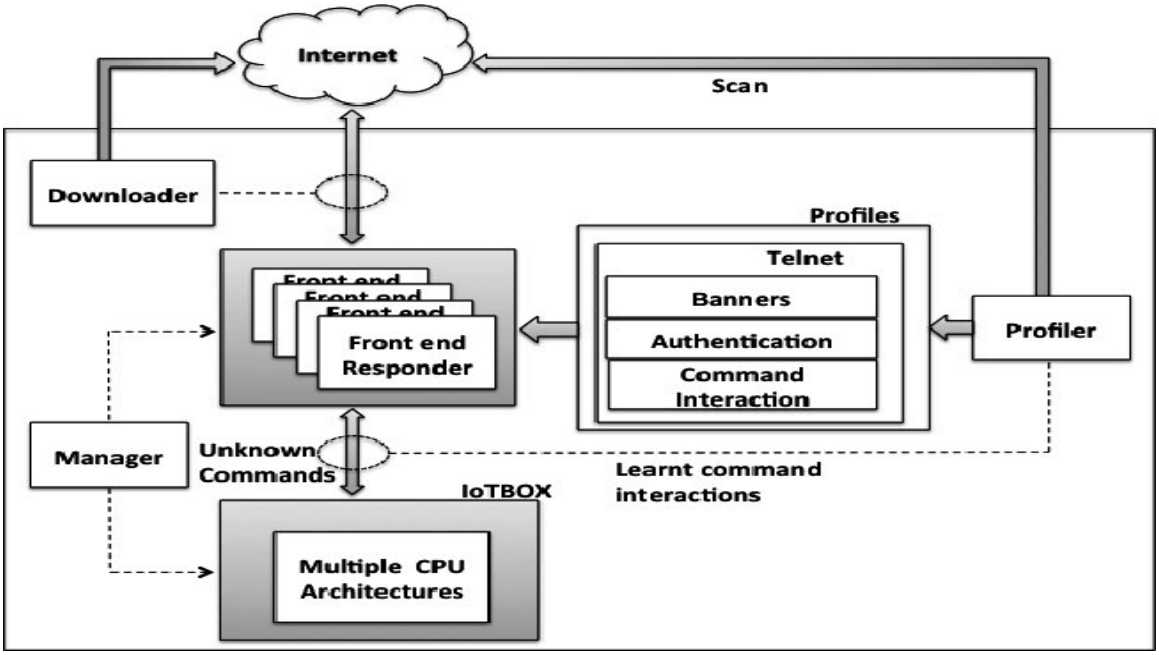
When an unknown command is received the *Frontend Responder* establishes a Telnet connection with the IoTBOX and forwards the unknown command to the IoTBOX. The IoTBOX as we mentioned already consists of a set of sandbox environments running Linux for embedded devices with varying CPU architectures.

Specifically, if the *Frontend Responder* encounters an unknown command with a specific device profile relating to some IoT device, this unknown command is then forwarded to the emulated sandbox that runs the CPU architecture of the specific IoT device. The *Frontend Responder* will then send the reply received from IoTBOX onto the client. The IoTBOX is periodically reset in order to protect the honeypot from malware infection. The IoTBOX is also used as a standalone in order to analyse malware captured by the system. After this interaction the profiler will update the *Frontend Responder* enabling it to handle this command in future.

The *Profiler* can be run in two operation modes, *active scan* and *visitor scan*. When run in active scan mode the honeypot proactively scans networks in order to gather the banners relating to various IoT devices. When run in visitor scan mode the honeypot connects back to hosts that attempted connection with the honeypot in order to obtain the banner for this device.

The *Downloader* is used to observe the attempts by botnets to download malware to the system using commands such as wget and tftp, both used by Mirai. After observing these commands the honeypot will then download the files from the URLs specified with these commands in order to gather and analyse potential malware.

The last component of the architecture below is *Manager* which uses IP tables in order to handle communication between the *Frontend Responder* and the *IoTBOX*.



IoTPOT Architecture Diagram

3.2.2 Conclusion

As mentioned already the IoTPOT is not open source and as of yet it is not possible to obtain the scripts for the honeypot upon request. The researchers did however provide access to traffic and malware data that we will discuss along with other data obtained through my own honeypot deployments in the evaluation section. The way in which the IoTPOT appears to go beyond other currently available honeypots is the way in which in deals with multiple types of devices, from the banner interaction stage to the command stage providing the appropriate responses. A well designed Telnet / SSH honeypot such as Cowrie could be used as a starting point in order to build a similarly dynamic honeypot, using QEMU to emulate the varying CPU architectures as is achieved with the IoTBOX.

3.3 Summary

In this chapter we looked at the increasing threat posed by IoT devices. We described the operation of the Mirai botnet, referencing the source code that was made available online in 2016. We detailed another research honeypot designed and developed specifically for the IoT threat. In the coming chapters we will look at the authors' experiments with honeypots in the duration of this project. We will discuss design considerations, implementation decisions and the evaluation of results from these experiments.

Chapter 4

Design

From the literature review we have seen that there is a multitude of options available for honeypot deployment. There are also a number of implementation decisions to consider when choosing a honeypot design. In this chapter we will focus on the design considerations and how they related to the goals of this project.

4.1 Design Overview

There are a number of important decisions to contemplate when choosing a design for one's honeypot deployment. For this project over time the focus became fixed on tracking and observing IoT botnets, particularly the Mirai botnet. The design of the honeypot, therefore, needed to reflect the features of the Mirai botnet, and the particular type of system targeted by the botnet.

I) Operating System

The choice of operating system for a honeypot depends on the type of attack the person deploying the honeypot wishes to attract. The two most common choices of operating system are various versions of Windows and Linux. The Windows operating system is often a target for botnets looking to exploit known vulnerabilities, particularly older Windows operating systems such as Windows XP. Microsoft ceased support for Windows XP in April 2014, when notifying users of this Microsoft included a security warning, making customers aware that continuing to use the operating system would result in their device becoming increasingly vulnerable. [46] A recent example of malware targeting the Windows system is the WanaCryptor (WCry) ransomware attack in May 2017. WCry exploited a known vulnerability in the Windows operating systems. [47]

Linux operating systems are also common targets for botnets. Web servers usually tend to be run on variants of Linux operating systems as are many industrial systems, in addition to this, many IoT devices use reduced versions of Linux as their OS. As we have seen from the IoT chapter the original Mirai botnet targets devices running the Busybox operating system. Busybox is a variant of the Linux operating system, therefore for it was necessary to use a Linux operating system, or an emulation of one, in order to track the Mirai botnet and other potential IoT botnets.

II) Interaction Level

In the literature review chapter we discussed the importance of interaction level and how the interaction level relates to the particular goals of a honeypot deployment. In certain papers it is contended that in order to observe fully botnets in the wild it is necessary to deploy a high interaction honeypot. Deploying such a honeypot is certainly an attractive option purely in terms of allowing the botnet free reign over the system while monitoring fully the results of this. The problem with this approach, however, is the system is often at risk of becoming a part of a malicious attack. There are legal and moral issues to consider before deploying such a honeypot.

The IoTPOT laid out a design for an IoT honeypot that suggested an alternative to this. The idea of capturing the malware, downloading this malware and then executing this malware in a separate safer environment is an attractive option and one that was considered ideal for the study of the Mirai botnet. There are sophisticated sandbox tools under active development that are built solely for this purpose. To execute captured malware in a sandbox environment in order to analyse the results. We will look at the specific requirements for a honeypot used for tracking IoT botnets in the requirements section. A well-maintained medium interaction honeypot, that enables an attacker to interact with a realistic Linux shell, that can be modified to include various binaries and commands as needed, and that can monitor the commands executed by the botnet, is required.

III) Monitoring

Monitoring the botnet's interactions with the system is of vital importance. We want to be able to monitor the user's access attempts, their geo location and credentials used in brute force login attempts. After an attacker has successfully entered the system we need to be able to monitor her interaction with the system. We need to be able to observe the shell commands entered. We also need to observe how the attacker initiates contact with the C&C server. Of particular importance is enabling the attacker to execute the wget and tftp commands used by the Mirai botnet in order to download the malware to the system. Monitoring these commands and the servers that the botnet is in contact with is important when considering the goals of the project.

IV) Location

Another aspect to consider in the overall design is the location of the honeypot. Location may refer to either the geographical location or the location of the honeypot on the network. In terms of the geographical location for this project one of the aims was to observe the regularity of Mirai attacks in various locations, also to observe if there were any significant differences between the interactions based on geographical location. To this end, honeypots were deployed on AWS instances at various locations in Europe, the US and Asia.

Network location refers to where the honeypot sits on the network. As we discussed at the beginning of chapter two, there are two main categories of honeypot deployment, research and production. Production honeypots are usually deployed as part of an overall production network as an added security measure. As such they are usually placed behind a firewall among the production services. The fact that it is placed behind a firewall means that honeypot traffic will be limited. In the case of this project we wanted the honeypots to be internet-facing similar to many of the vulnerable IoT devices targeted by IoT botnets.

4.2 Design Choices

In this section we will first present the requirements for the honeypot. We will then look at the choices available for the design of such a system.

I) Requirements

The main goal for honeypot deployment in this project was to monitor and analyse the threat posed by IoT botnets. In order to achieve this the system needed to include the following features.

The honeypot should be accessable via the Telnet protocol and/or via the SSH protocol. As we have discussed in previous chapters the main targets for IoT botnets such as Mirai are vulnerable devices with Telnet enabled. Therefore it was vital for the honeypot to advertise these protocols to botnets scanning for vulnerable devices.

The attacker must be able to gain access to the system and be presented with a realistic system in which expected interaction is possible. It must be possible to choose what credentials will be accepted or otherwise refused.

The honeypot must be able to accept and respond to the commands run by the botnet once it has gained access. If the system cannot do this it will lead to the botnet disconnecting from the system ensuring that no further analysis is possible.

We need to be able to capture the commands entered by the botnet while interacting with the system.

We need to be able to log interaction conveniently and visualise important data when desired.

Finally, we hope to analyse the botnet's communication. In order to do this we need to execute malware obtained by the honeypot. We then wish to capture and analyse the traffic generated

4.2.1 Software / Interaction Level

In the literature review we looked at a number of various honeypot software solutions. The majority of these were low / medium interaction honeypots. We also looked at two high interaction honeypots, and two research honeypots. A number of these could be ruled out immediately. The high interaction honeypots along with a few of the medium interaction honeypots are not actively maintained while the research honeypots are not yet open source. This narrows down the available options drastically. The one solution that appears to be well suited to tracking IoT botnets that focus on exploiting the Telnet protocol is the Cowrie honeypot.

4.2.2 Host System

The question of on what type of system the honeypot should run was a relatively straightforward one to answer. Initial considerations included running the honeypot on VMWare on a home network. There were obvious issues with this option. Firstly the issue of attracting attackers to a home network that is also shared with other users. There are ways to circumvent this problem by isolating the machine on a virtual LAN, but there are still risks involved. In terms of running a honeypot such as Cowrie, on their Github is the recommendation to run "on a dedicated well firewalled Virtual Machine as the non-root user."

Having considered this option, even in the scenario where one felt fully secure in deploying the honeypot on a home network, it would not lead to obtaining results relating to the geographical location of the machine. Running the honeypot on a VPS with a service such as Amazon Web Services or DigitalOcean certainly seemed to provide the best solution. As well as providing great flexibility in terms of location and type of OS and number of deployments, it circumvented any security concerns relating to running a honeypot on a home network.

The option of having two servers, one server used solely as the honeypot system, and the other as the graphical system was the most appealing option. We will discuss the implementation choices for this in the next chapter.

Another option that was never in real consideration was to use a physical honeypot. Again, this idea would have lacked the flexibility of the VPS solution, required the purchasing of hardware equipment and would have been run on a home network.

4.3 Conclusion

After considering the available options the most logical choice was to run the honeypots on a VPS system. This choice provided various advantages particularly in terms of flexibility and allowing room for experimentation. In the next chapter we will look at the implementation of the honeypots and how the design requirements shaped the implementation process.

Chapter 5

Implementation

In this chapter we will look at the implementation of the honeypots based on the design options and the system requirements discussed in the previous chapter. We begin with a brief discussion of initial experiments.

5.1 Initial Experiments

To begin the project and to gain experience with honeypot deployment, the Modern Honey Network application was installed on an Ubuntu server. As we discussed in the literature review, the MHN allows the user to easily deploy a choice of sensors. After installing the MHN server manager a Dionaea as well as a Kippo sensor were deployed. MHN includes an out of the box visualisation of attacker's credentials and geographical location as well as the type of attack vector. While this was interesting to view no malware was captured during the deployment of the sensors. Another piece of honeypot software that was experimented with was T-Pot. We discussed T-Pot in the literature review section. Again, although interesting to experiment with, no malware was captured and T-Pot was not considered for use in an overall solution.

5.2 Software / Tools

The main aim set out at an early point of this project was to observe and analyse the threat posed by vulnerable IoT devices and the IoT botnets that target them. In the design section we laid out the requirements for a honeypot system that would help achieve this goal. In this section we will look at the software solutions found in the research stage of the project that were best suited towards achieving these goals.

I) Cowrie

In the literature section we discussed the Cowrie honeypot. Cowrie is a medium interaction honeypot that began as a fork of the Kippo project. It is open source and actively maintained and developed by its original creator Michael Oosterhof with many other contributors. When looking at potential honeypot solutions it was a matter of checking the features of each solution against the requirements set out in the design section.

The first issue that one needs to be sure of is that the software in question is actively maintained. There are various interesting honeypot solutions that we discussed in the literature section that have not been maintained for some time, honeypots such as Kippo and Dionaea would fall into this category. These solutions were therefore immediately ruled out as viable options. Cowrie, however, is well maintained with an active Github where issues raised are invariably responded to in a timely manner.

The next requirement was that the solution enables a malicious user to gain access to the system via the Telnet or SSH protocols. These ports should be visible to a botnet scanning for new victims. Again Cowrie was well suited for this purpose.

The Cowrie system is also an emulation of a Linux operating system, this was again a prerequisite for our honeypot solution. It provides an out of the box file system that crucially can be amended to suit the purposes of the honeypot, we will discuss this in more detail in a later section.

Cowrie is actively maintained with the developers aware of issues relating to fingerprinting and the importance of providing accurate responses to commands executed by an attacker. This is a strength of the Cowrie honeypot, however it is not possible for any honeypot software to suit the needs of all users, this is why the open source nature of the project is also of great importance. This allows the honeypot deployer to make amendments to the source code based on observations of the attacker's interaction with the system.

Monitoring the attacker's interaction with the system is also a requirement for tracking Mirai. The Cowrie honeypot records all the interactions, storing an attacker's session in a few different formats including json files, which can be used as a source for displaying the data graphically.

Another important goal of this project is to capture malware that the honeypot attempts to upload to a compromised system. The cowrie honeypot allows us to do this, storing the malware as its SHA-256 checksum, while avoiding storing duplicates of the one malware.

Finally, we wish to execute the malware saved in order to analyse the results of this. Cowrie is not a high interaction honeypot and does not provide this service. For this we will need to look at alternative solutions. We will discuss one such solution, Cuckoo, in a following section.

II) Platform

After choosing Cowrie as the most suitable solution for the project's aims the next decision to be made was where to run the software. Cowrie is a medium interaction honeypot and does not call any external software, it is certainly safer than running a fully high interaction solution. On the Cowrie Github page however, the response to the question "Is Cowrie secure?" is as follows

"*Cowrie is written in Python, and doesn't call any external software, so it's probably somewhat secure*.

However, Cowrie has not had any real security audit done on it, and it's definitely vulnerable to some DoS attacks, as there are no limits on how many people can connect to it, or how many files they can download.

It's my recommendation to run Cowrie on a dedicated well firewalled Virtual Machine as the non-root user."

The first option considered was to run it on a VMWare virtual machine on my home network. After taking into account the potential security issues this entailed this was deemed to be a less than ideal solution. Running the honeypot on my home network also wouldn't have satisfied the requirement of observing interaction with the honeypot in different geographical locations. The most logical solution for running the honeypot was to use a Virtual Private Server (VPS). There are various options available for such a service, the obvious two being Amazon's Amazon Web Service (AWS) and DigitalOcean.

Both AWS and DigitalOcean offer various pricing levels relating to memory size, disk space and CPU power. Both offer Ubuntu images which is the recommended operating system for running the Cowrie honeypot. The decision to use AWS was made easy with the option of a free tier available.

III) Visualisation – ELK

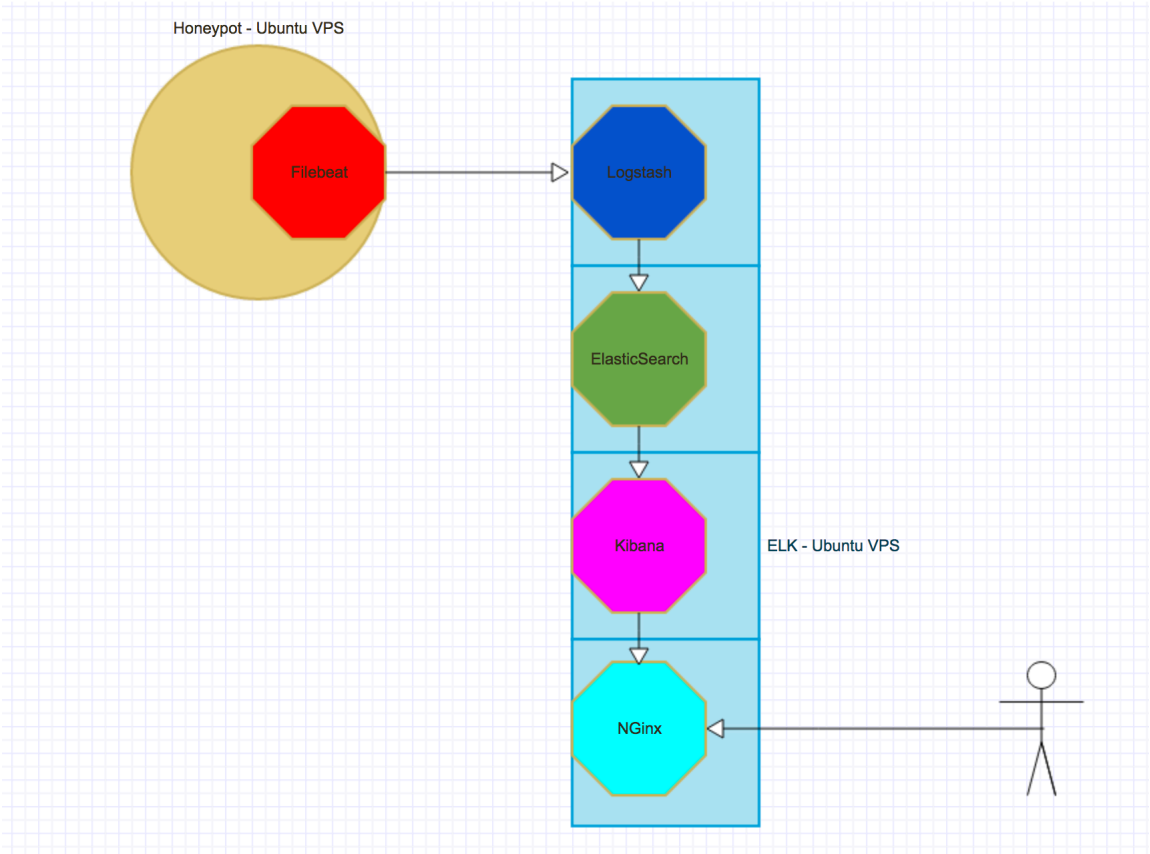
Another important aspect of the honeypot setup is the visualisation of data collected from the honeypot. As we previously mentioned the logs for Cowrie are stored in json format making it a relatively straightforward task to visualise the data. There are various choices available for such a task but the option that seems to be standard is the use of what is known as the ELK stack. The ELK stack consists of the three applications, ElasticSearch, Logstash and Kibana. For the setup here Elk was used in conjunction with the Filebeat application. We will discuss briefly what each of these applications does.

The diagram below shows the setup employed. The Cowrie honeypot is placed on one AWS server with the ELK visualisation stack placed on another AWS server, with private networking set up between the two servers. The Filebeat application is used as a means of sending the honeypot data from the honeypot server to the ELK server.

On the ELK server side Logstash is used to accept and process the incoming cowrie logs on a user defined port. Logstash forwards the now filtered data to Elasticsearch which is listening on localhost:userdefinedPort. Elasticsearch is JSON search and analytics engine that in this scenario is used to store the logs forwarded by Logstash.

The final part of the ELK stack is Kibana. Kibana is a web interface for visualising the Cowrie logs. It is possible to search the logs and tailor the visualisations depending on the requirements. The NGinx reverse proxy is there to allow outside access as Kibana is set to listen on localhost for security reasons. This is an optional step as Kibana can also be setup to listen for external conenctions.

A simple custom Python server was also used for visualsation and making files available for download.



Honeypot -> ELK Setup

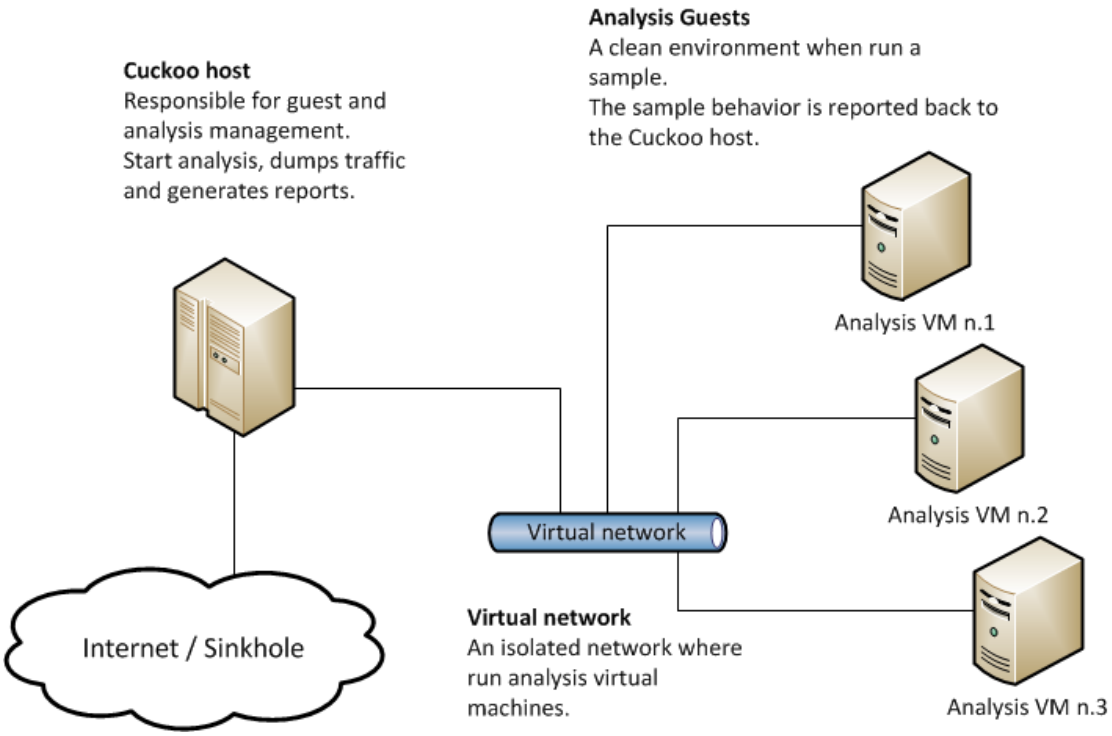
IV) Cuckoo Sandbox

In order to execute and analyse the malware captured in a safe environment the Cuckoo software was chosen. The Cuckoo Sandbox is an open source tool for malware analysis written in Python. The description of what the Cuckoo Sandbox can do on the Cuckoo website is as follows -

"*Cuckoo Sandbox is an advanced, extremely modular, and **100% open** malware analysis system with infinite application opportunities.*"

It is capable of analysing various types of malicious files from executables to document exploits. It can dump and analyse network traffic, even if the traffic is encrypted. It can trace the API calls and general behaviour of the analysed file, among other features including memory analysis of the infected virtual system.

The Cuckoo sandbox can be run on any virtual machine software such as VMWare or VirtualBox.



Cuckoo Sandbox Architecture

5.3 Setup & Installation

The setup and installation of Cowrie on the AWS server was a straightforward process. The Cowrie code and documentation are updated regularly so to install the system on the AWS Ubuntu image was just a matter of following the install instructions. This involved nothing unusual, cloning the git repo, creating a new user on the system as Cowrie cannot be run as root for security reasons, and following other steps such as installing additional Python libraries.

There are a couple of important configurations to be made as soon as Cowrie has been set up and tested. By default Cowrie runs on port 2222, so it is vital to modify this so that Cowrie receives all incoming traffic on port 22 (SSH). There are two specific ways that this can be achieved. The first option is to use IP tables creating a rule whereby all incoming traffic on port 22 will be forwarded to port 2222.

$ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222

The other option is to use Authbind in order to listen on port 22 as non-root. Both of these options were used on various deployed honeypots and both worked without issues. When making this change on the AWS instance it is important to edit the sshd_config file to change the SSH port for the instance. If this step is not taken it will not be possible to access the instance after exiting, as making an SSH connection via port 22 will no longer be possible. These steps will set up the Cowrie honeypot to listen for incoming connections on port 22, and once they have been configured correctly, the user will find a steady stream of access attempts which can be viewed by running the command "*tail -f ~cowrie/cowrie/log/cowrie.log*".

In order to enable the Telnet protocol the user must follow similar steps, using Authbind to listen as non root on port 23. Again, once Telnet was set up and enabled an almost constant stream of login attempts could be observed.

The next section in the configuration process is choosing which credentials should be accepted or rejected. Cowrie is set up to accept certain username / password combinations that can be easily edited in the *userdb* text file stored in the data directory. You can allow specific combinations or accept all attempts from a specific username. For the purpose of tracking Mirai and capturing IoT botnets, considering the Mirai source code, it was important to at least allow access to username

root.

At this point in the setup it was already possible to observe malicious users consistently attempting to gain access to the system.

5.4 Tracking Mirai

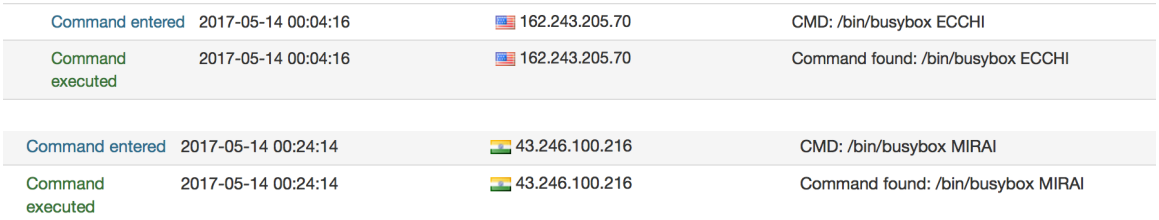
An SSH honeypot and a Telnet honeypot were now set up, the SSH on a DigitalOcean instance and the Telnet honeypot on an AWS instance. After a day or two the SSH honeypot had successfully captured malware, however, the Telnet honeypot which was of more interest for this project had failed to capture any malware samples. From observing the attacker's interactions with the honeypot system it was clear that the attacker would enter the system and upon attempting to execute certain commands would then exit. There were specific commands that were observed regularly that the honeypot was unable to deal with, this led to a process of observing the interactions of the botnet with the system, and then making appropriate amendments to the honeypot in order to prolong the botnet's session, with the aim of capturing malware.

As we mentioned in the section on the Mirai botnet, Mirai targets devices running the Busybox operating system. The following is a description of Busybox taken from the official Busybox website -

"*BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides minimalist replacements for most of the utilities you usually find in GNU coreutils, util-linux, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts.*"

A command often executed at the beginning of the session is the /bin/busybox ECCHI or /bin/busybox/MIRAI command. This is used as a fingerprinting method by the attacker in order

to test that they are inside the system they expect and not a honeypot or a device run on a more complete version of Linux. If the attackers are inside a honeypot or a device running another version of Linux they will expect an incorrect response, quite often the response to an unknown command will be a help screen. If the attackers receives this response they will know they are not inside the system they expected and will invariably exit the system. The response the attacker is expecting from running this command is “ECHHI: applet not found” or “MIRAI: applet not found”. It is important to give the attacker this response to these commands if one hopes to observe a full session. It is important to ensure that the honeypot is giving this correct response and to modify the Busybox module otherwise. This is easily tested by running “bin/busyboxy ECCHI” from the command line.



Command entered	2017-05-14 00:04:16	162.243.205.70	CMD: /bin/busybox ECCHI
Command executed	2017-05-14 00:04:16	162.243.205.70	Command found: /bin/busybox ECCHI
Command entered	2017-05-14 00:24:14	43.246.100.216	CMD: /bin/busybox MIRAI
Command executed	2017-05-14 00:24:14	43.246.100.216	Command found: /bin/busybox MIRAI

Busybox command run during session

Once the honeypot is capable of responding correctly to the /bin/busybox command the attacker will stay connected and attempt more commands. A command regularly encountered in the sessions is the “/bin/busybox echo” command.

Command entered	2017-05-14 00:39:29	162.243.205.70	CMD: /bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon; /bin/busybox cat /proc/.nippon; /bin/busybox rm /proc/.nippon
Command executed	2017-05-14 00:39:29	162.243.205.70	Command found: /bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon

Echo command run during session

From observing the honeypot interactions it was possible to see that the session would usually disconnect, after this command was executed by the attacker. The Telnet honeypot was still not capturing any malware at this point and therefore needed further amendments to be made. The

honeypot was apparently not able to deal with the “echo” command. In order to solve this issue the honeypot’s file system needed to be modified, and this was achieved as follows.

Inside the honeyfs directory the bin directory was created with the following command -

$ sudo mkdir bin

After creating this directory in the cowrie file system, we need to place a binary echo file in the directory in order to give the attacker the response it is expecting. We do this by copying the echo file from the Ubuntu server into the newly created bin directory, with the following command.

$ sudo cp /bin/echo /home/cowrie/cowrie/honeyfs/bin/

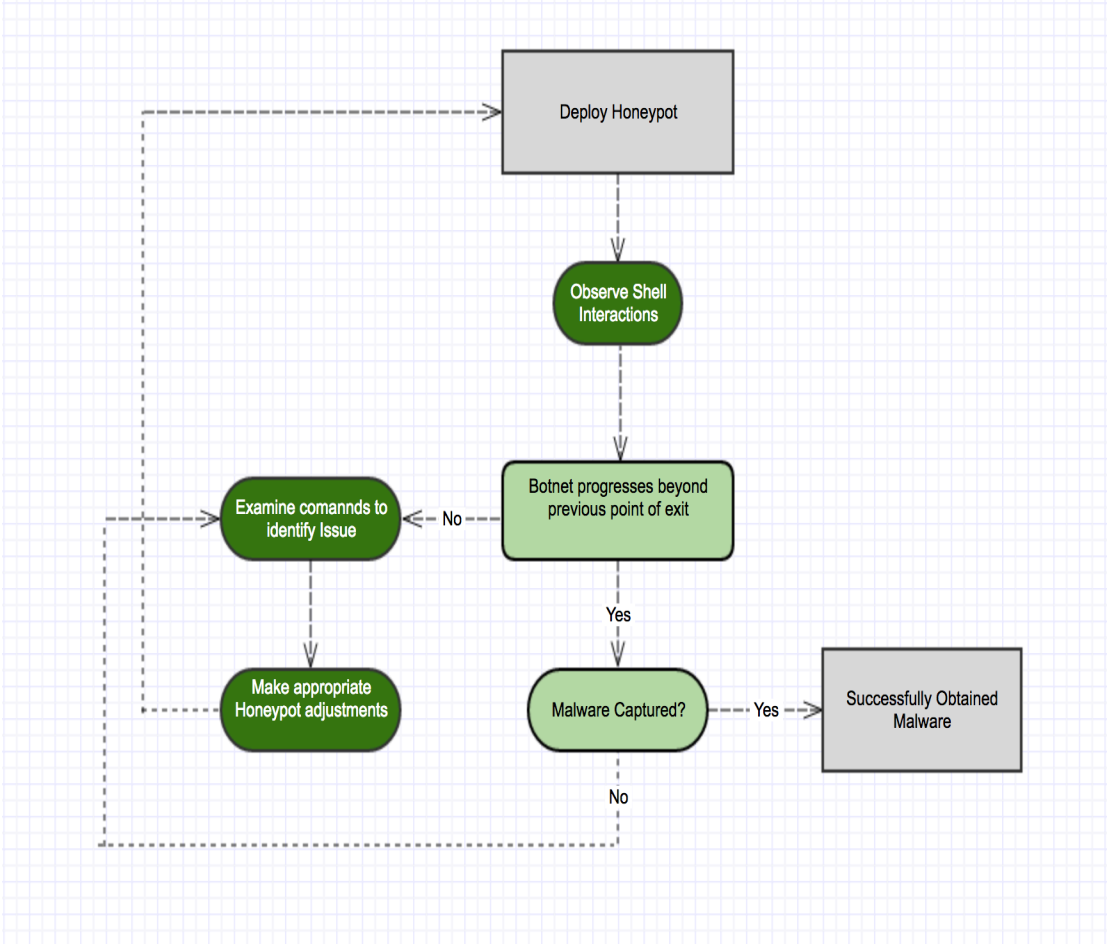
Now if we execute the following command from the cowrie directory as a test -

$ file honeyfs/bin/echo

we receive the following response –

honeyfs/bin/echo: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=9a7491319c83fdfccc9e23340320177e8d186e3d, stripped

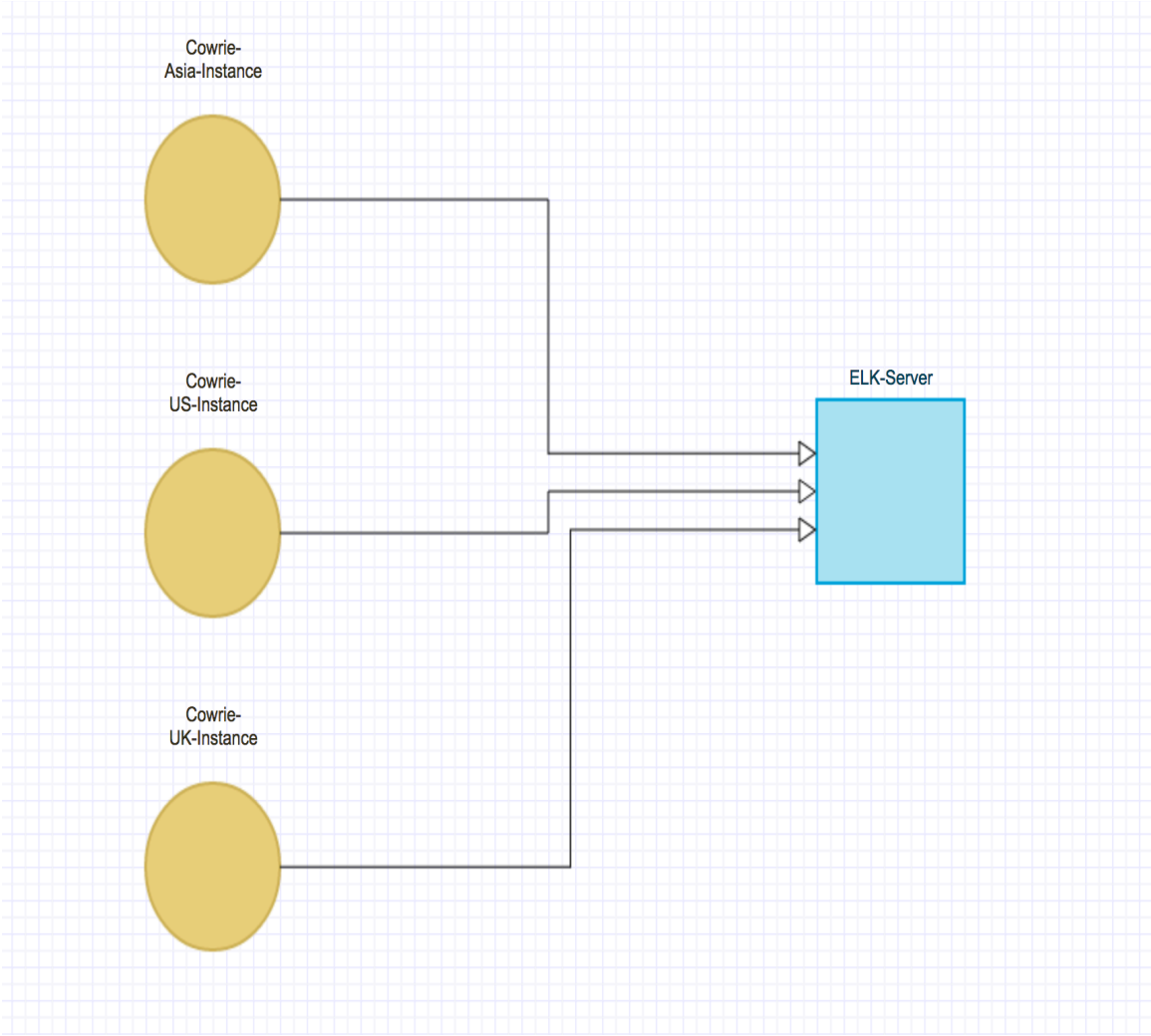
If the attacker attempts to execute the echo command after this amendment it will appear to the attacker that she is in fact inside a real system. Further observations and related amendments, such as limiting the number of characters in response to a specific command, needed to be made to the honeypot before it was capable of capturing malware binaries from the Mirai botnet. This process of tracking the attacker’s interactions and amending the honeypot accordingly is laid out in the diagram.



Capturing Malware

Once the honeypot deployers are aware of what it is they are hoping to achieve with their honeypot setup this process will need to be followed. In the case of this project the hope was to catch malware from IoT botnets, particularly from Mirai. So the sessions were tracked and the honeypot amended accordingly until the honeypot began to capture the malware. This applies to any other setup too. For example in the case of the SSH honeypot deployed during the project the command “cat /bin/ls” was regularly observed to be executed, so it was necessary to place a binary in the honeypot system to deal with the command. If the honeypot deployer is looking to track CPU specific malware the honeypot will need to be modified depending on the architecture. If the CPU architecture is ARM then the attacker may run the command such as “cat /bin/sh” at the beginning of the session to test the system is capable of running the malware. In order to deal with this situation the honeypot deployer will need to find a relevant binary and place it in the honeypot’s file system.

Another goal of the honeypot deployment in this project was to observe the interactions of attackers in different geographical locations. In order to accomplish this a number of the modified honeypots were deployed at various locations using AWS services.



Geographical Distribution of Honeypots

5.5 Conclusion

In this chapter we looked at how the honeypots were implemented during the project as a means of achieving the goals set out in the design section. How a honeypot is implemented is very much dependent on the goals of the researcher or oganisation deploying it. Changing the goals of the research will result in the need to change the honeypot's implementation. If one of the goals is to capture a certain type of malware, as was the case in this project, then the honeypot setup will need to reflect the needs of the malware in question. The following quote, from the creator of the Cowrie honeypot in response to a query on this subject, puts it in a rather succinct manner.

"*As you've found out, you will always have to modify your honeypot depending on what you want to catch. If the malware is ARM specific and checks by catting /bin/sh to check for ARM binaries, you could find a binary somewhere and put that in honeyfs/bin/sh. Honeypot detection and anti-detection is a game of cat and mouse. If I did all this in the official binaries, very quickly they would check for something else, more advanced.*" Michael Oosterhof

Chapter 6

Evaluation

In this chapter we will assess the current efficacy of honeypot solutions encountered in the duration of the project. We will analyse the outcome of honeypot deployments for the project while also examining data shared by the IoTPOT researchers.

6.1 Design Characteristics of Honeypots

In previous chapters of the report we looked at the design characteristics of honeypots. We discussed how the goal of the research dictates the necessary characteristics of the honeypot, while also discussing the characteristics needed to meet the requirements for the project.

I) Honeypot Interaction

The interaction level of the honeypot should reflect the goals of the project. In response to Mirai and other IoT botnets a number of low interaction honeypots, such as MTPot were created. The MTPot is well designed, easy to deploy and serves the purpose of logging access attempts from Mirai bots. Medium interaction honeypots such as Cowrie go beyond this simple low-level interaction. We have discussed the ability of such a honeypot to emulate an operating system in order to monitor an attacker's interactions with the system. A well-designed medium-level interaction honeypot may also have the ability to capture malware fur further analysis. A high-interaction honeypot allows a malicious user free reign over a fully-fledged operating system.

Such a honeypot can yield very interesting results, particularly in the analysis of botnets but also carries significant risk.

A medium interaction honeypot used in conjunction with other analysis tools such as a dedicated sandbox for malware analysis can make for a dynamic honeypot solution, potentially offering most of the features of a high-interaction honeypot while lowering the risk involved.

II) Operating System

The choice of operating system is also very much dependent on the type of attack the honeypot is supposed to entice. We discussed in the previous chapters the various options available, concentrating on Windows and Linux. We differentiated between these two suggesting when it would make sense to use a Windows operating system as opposed to Linux and vice versa. The choice of operating system is of utmost importance and should always reflect the type of attack the researcher intends to attract and monitor. In the case of this project it was essential that the honeypot was running a Linux operating system or an emulation of one.

III) System Vulnerability

The vulnerability or vulnerabilities present in a honeypot system will also contribute towards the type of attack the honeypot system will encounter. In the state of the art chapter we discussed various types of vulnerabilities, primarily software vulnerabilities. We also discussed making a system vulnerable to brute force access attempts. This is the vulnerability we needed for our system in order to entice attacks from IoT botnets. The honeypot deployers may also create a specific software exploit specifically in their system in order to attract and monitor another type of attack. For example, with the recent WCry attacks, creating a vulnerability in a system that attracted such an attack in order to monitor its interactions may hold great value. Again, just like the choice of operating system the types of vulnerability present in the honeypot system very much dictate the types of attack the system will entice.

IV) System Monitoring

Monitoring the activity of malicious users is the central reason for deploying a honeypot system. When deploying a high interaction honeypot the researcher must be wary of fingerprinting issues, that is, the method of monitoring must not make obvious to an attacker that they are in fact interacting with a honeypot system. There are various solutions for monitoring a system. If the goal of the honeypot is to monitor human attacks then the monitoring system will potentially need to be more sophisticated in order to avoid detection by the human attacker. There can be tradeoffs involved between quality of the monitoring system and the avoidance of detection.

Summary

As we can see from the previous subsections, a honeypot's design and implementation should be a reflection of the research and security goals laid out prior to its deployment.

6.1.1 The Current State of Honeypot Solutions

In the literature chapters we looked at the current state of the art honeypot solutions. During the literature research phase of the project it was difficult to find any full, actively maintained, open source high interaction honeypot solutions. In fact it would be truer to say that no such solution was found. The inherent security issues present in high interaction honeypot systems may be a contributing factor behind this. If a researcher develops such a system for their own purposes, do they necessarily feel secure in making it available to anyone, particularly users with limited or beginner's knowledge of the area.

In terms of medium-interaction honeypots we have already discussed some of the options in detail. There are a few viable options, with the Cowrie honeypot being a particularly valuable, well-maintained solution. The IoTPOT discussed in the chapter on the Internet of Things outlines a very interesting design for a dynamic honeypot solution. An open source solution based on a similar idea would be an interesting addition to the current collection of honeypot solutions. We will discuss a possible implementation of such a system in the conclusion section of this chapter.

6.2 Results of Deployment

The first notable observation, immediately after setting up the honeypots to listen on either port 23 or port 22, was the almost constant stream of traffic attempting to gain access to the system. With the first honeypot deployment the system allowed access to any attempts with the username root, regardless of the password used. Other honeypots deployed allowed access to any of the username / password combinations present in the original Mirai source code. In every instance the *root* username was always the most commonly used.

In the implementation chapter we discussed some of the issues that needed to be overcome in order to attract and maintain the interest of malicious users.

London SSH Pot Home Data Review Files Obtained

Username	Count
root	2720
admin	31
richard	6
user	2

Username Data – London SSH (Three weeks)

NYC Pot Home Data Files Obtained

Username	Count
root	302
admin	34
user	4
mother	2
support	2
guest	1

Password	Count
xc3511	55
(empty password)	45
7ujMko0vizxv	43
system	43
vizxv	34
admin	28

Username / Password Combinations – New York Telnet (One day)

In terms of the geographical locations of the source IP addresses, for every honeypot deployed during the project, the greatest source of attackers was China. The placement of the honeypot didn't have as big impact on the geographical location of the attacker as one might expect. If the honeypot was placed in the US or Europe, China was still producing the most attackers. There were some aspects that did seem to depend on geographical location. For example, a Dublin honeypot was deployed on DigitalOcean, and this was the only time there was an attacker with an Irish IP address. Another example was there seemed to be a slight increase in attackers with American IP addresses when the honeypot was placed in an American region. Honeypots were deployed in a number of regions over a three to four week period, and overall the geographical placement of the honeypot did not seem to have a significant effect on the geographical locations of the attackers. Russia and Vietnam were also common locations for attackers.

London SSH Pot Home Data Review Files Obtained

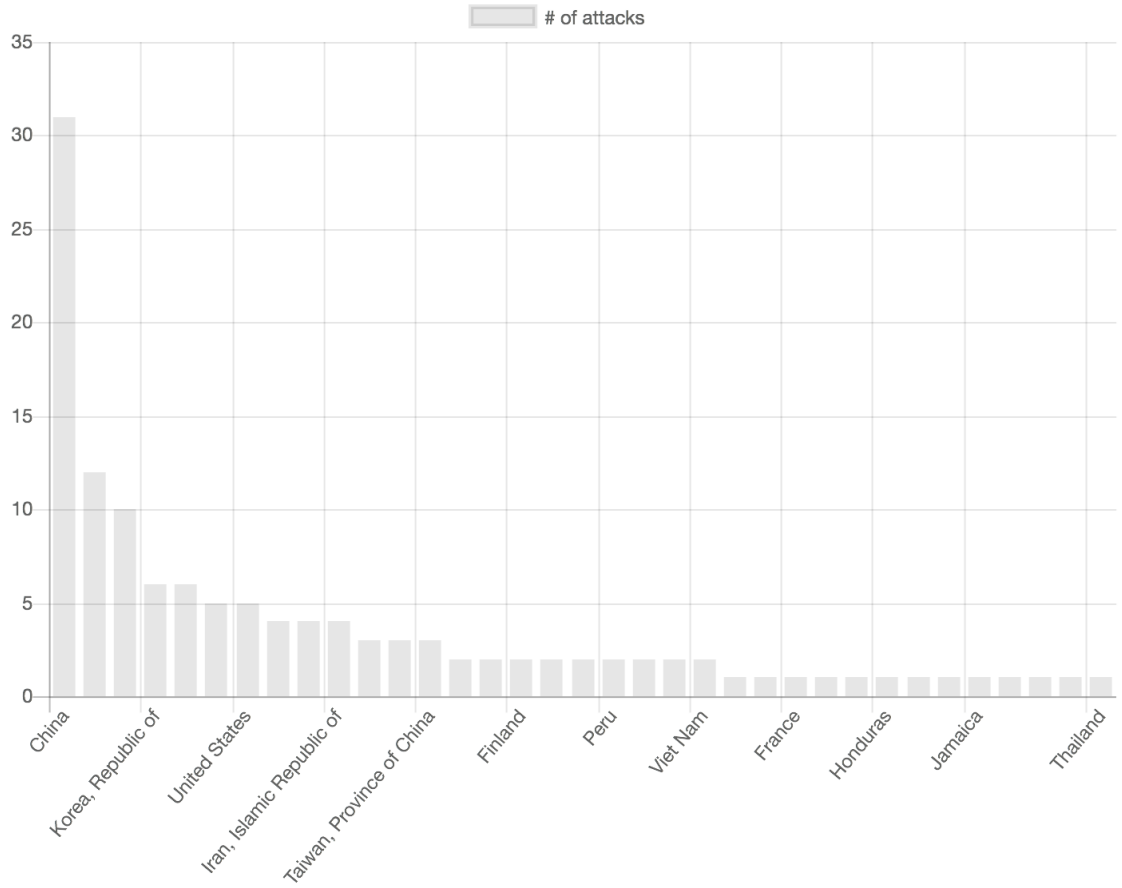
Country	IP count
China	151
Viet Nam	144
Germany	113
Brazil	76
Czechia	37
Russian Federation	29
United States	29
Argentina	24
France	22
India	14
Korea, Republic of	13
Netherlands	12
Ecuador	9
Romania	8
Singapore	7
United Kingdom	6
Japan	5
Italy	4

Geographical Locations of source IPs – London SSH (Three weeks)

Amsterdam Telnet Pot Home Data Files Obtained

Country	IP count
China	69
Russian Federation	16
Mexico	9
United States	9
Iran, Islamic Republic of	8
Brazil	6
Korea, Republic of	6
Ukraine	6
Thailand	5
Turkey	5
Argentina	4
Spain	4
Taiwan, Province of China	3
Viet Nam	3
Colombia	2
France	2
United Kingdom	2
Hong Kong	2
Indonesia	2
India	2

Geographical Locations of source IPs – Amsterdam Telnet (One day)



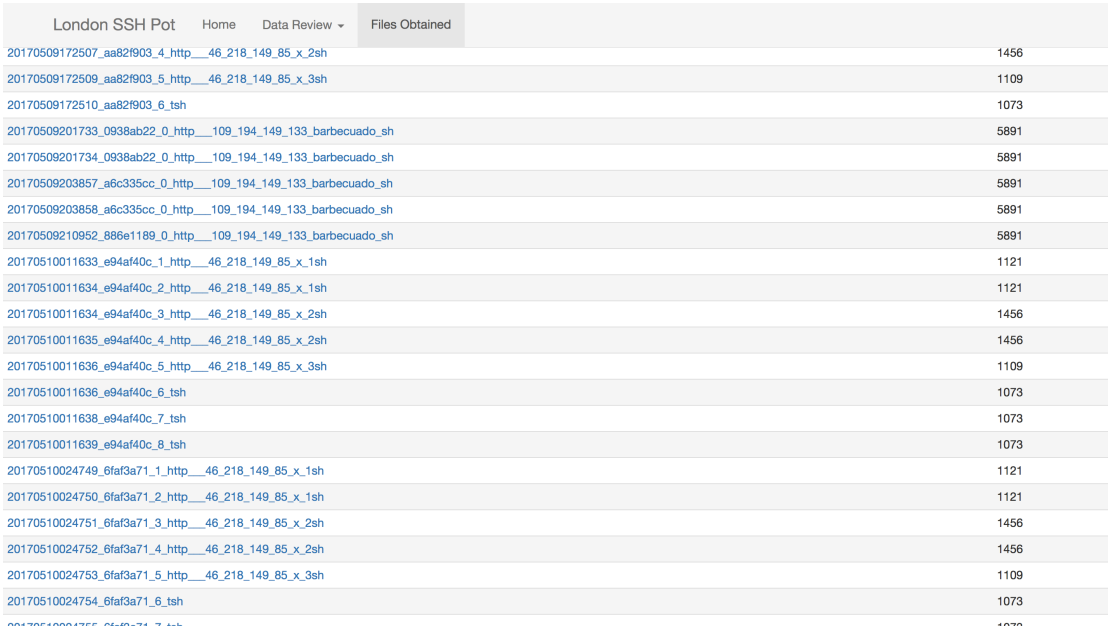
Geographical Locations of source IPs – Singapore Telnet

I) Malware Captured

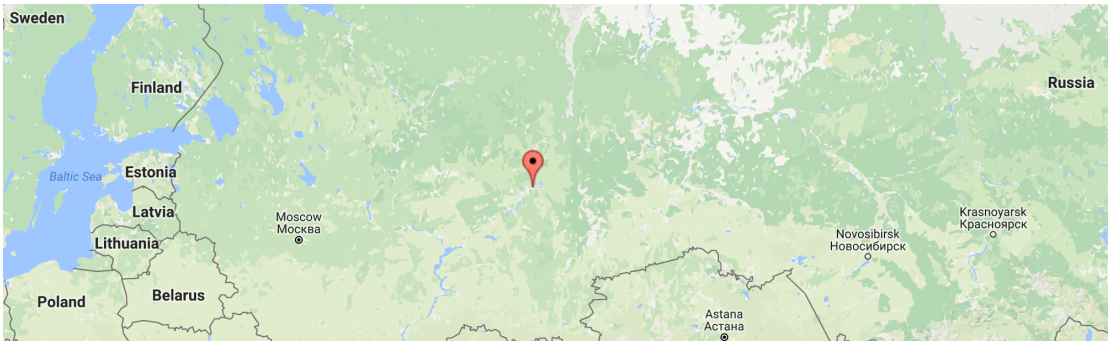
All the malware samples were saved as their SHA-256 checksum, and no duplicates were saved in the honeypots.

II) SSH Malware Sample

At the time of writing the SSH honeypot deployed in the London region had obtained over 500 files. The image below from the Python server obviously shows just a fraction of those files captured. From testing a sample of these files in the sandbox a number turned out to be benign while others were indeed malicious files.



Taking the file finishing with *"…barbecuadao_sh"* from the image, running a Sandbox analysis of the file revealed the following. The file type is Ascii text, the target CPU architecture is x86. The network analysis reveals a successful connection via HTTP to the IP address 109.194.149.133. This IP address has a geographical location somewhere in Russia, shown on the map.



IP Location

The malicious user runs the following commands from the device for a set number of hardcoded file names, one of these being FatGGei -

wget http://109.194.149.133/FatGGei;
curl -O http://109.194.149.133/FatGGei;
chmod +x FatGGei;
./FatGGei;
rm -rf FatGGei

So we can see the user transfers a file to the system, makes it an executable, runs the executable and then removes the executable from the device.

From the TCP stream we can see the following interaction among many others –

[172.16.1.14:51068 --> 109.194.149.133:80]
GET /FatGGei HTTP/1.1
Host: 109.194.149.133
User-Agent: Wget
Connection: close

[109.194.149.133:80 --> 172.16.1.14:51068]

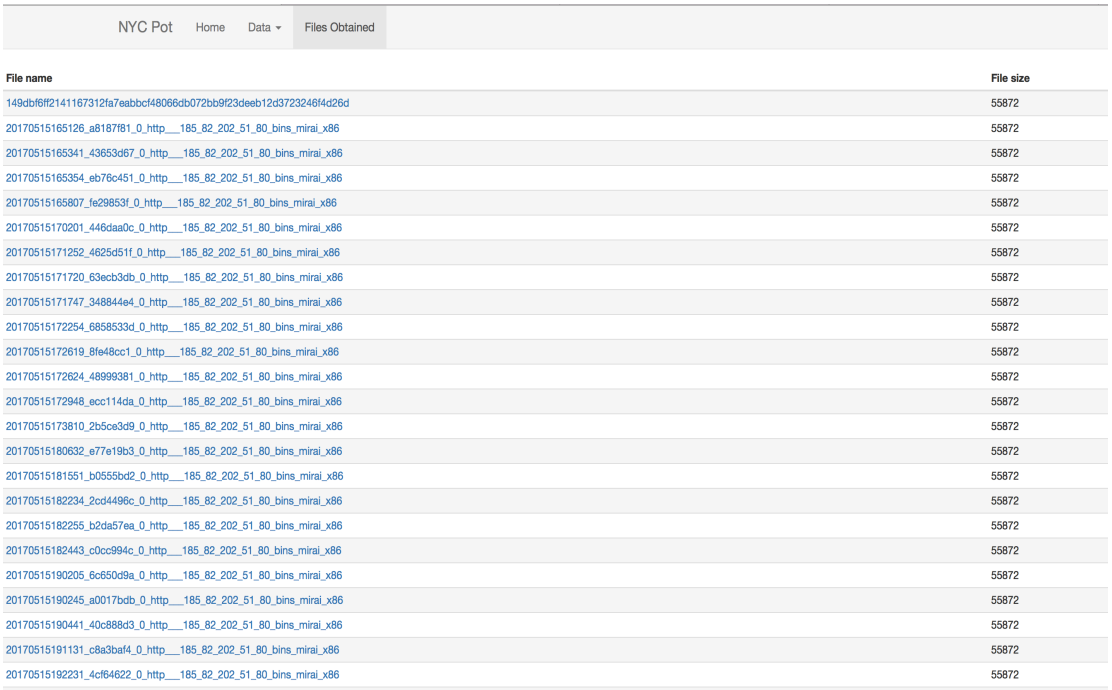
HTTP/1.1 404 Not Found
Date: Fri, 12 May 2017 23:10:33 GMT
Server: Apache/2.4.7 (Ubuntu)
Content-Length: 285
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
* The requested URL /FatGGei was not found on this server. *
<hr> <address>Apache/2.4.7 (Ubuntu) Server at 109.194.149.133 Port 80</address>
</body></html>

As we can see from this interaction above the FatGGei file was not found on the contacted server. This was the case for all the other attempted file transfers for this session. The bot in this case seems to have been unsuccessful in obtaining the malware files.

III) Telnet Session & Malware Sample

As stated previously, the Mirai and Hajime botnets tend to target devices with the Telnet service enabled. For this reason the samples collected from the Telnet honeypots were of the greatest interest in relation to the goals of this project. Very soon after the Telnet honeypots were running with the correct modifications, some of which were laid out in the implementation chapter, the honeypots began to capture malware samples, the images below showing a portion of captured samples by a Telnet honeypot set up with New York as its geographical location. As can be seen most of the files in this honeypot contain the Mirai title.



Captured Malware

Firstly we will look at the patterns observed in the attacker's behaviour, upon gaining access to the device. From this we will be able to ascertain whether it corresponds with what we expected from the Mirai review in the IoT chapter. We will step through a particular session, note the server name chosen for this honeypot, was svr04.

The session begins with the enable command –

```
root@svr04:~# enable
enable .
enable :
enable [
enable alias
enable bg
enable bind
enable break
enable builtin
```

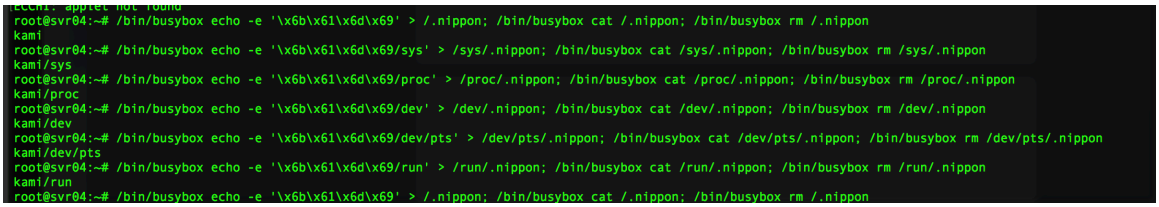
Then as expected the attacker runs the "/bin/busybox ECCHI" command, receiving the correct "ECCHI: applet not found" response -

```
root@svr04:~# shell
bash: shell: command not found
root@svr04:~# sh
root@svr04:~# /bin/busybox ECCHI
ECCHI: applet not found
root@svr04:~# /bin/busybox ps; /bin/busybox ECCHI
  PID TTY          TIME COMMAND
 5673 pts/0        0:00 -bash
 5679 pts/0        0:00 ps
ECCHI: applet not found
```

The user checks that proc is mounted -

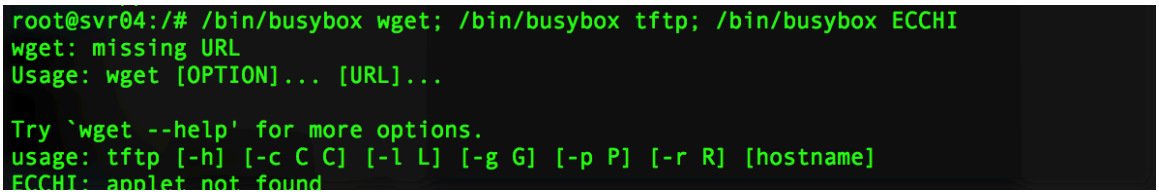
```
root@svr04:~# /bin/busybox cat /proc/mounts; /bin/busybox ECCHI
rootfs / rootfs rw 0 0
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
proc /proc proc rw,relatime 0 0
udev /dev devtmpfs rw,relatime,size=10240k,nr_inodes=997843,mode=755 0 0
devpts /dev/pts devpts rw,nosuid,noexec,relatime,gid=5,mode=620,ptmxmode=000 0 0
tmpfs /run tmpfs rw,nosuid,relatime,size=1613336k,mode=755 0 0
/dev/dm-0 / ext3 rw,relatime,errors=remount-ro,data=ordered 0 0
```

The user checks the filesystem, checking that the echo binary exists –



The user checks if the filesystem can be written to -

The user checks that the wget and tftp commands exist, receives usage response so now knows commands exist in system -

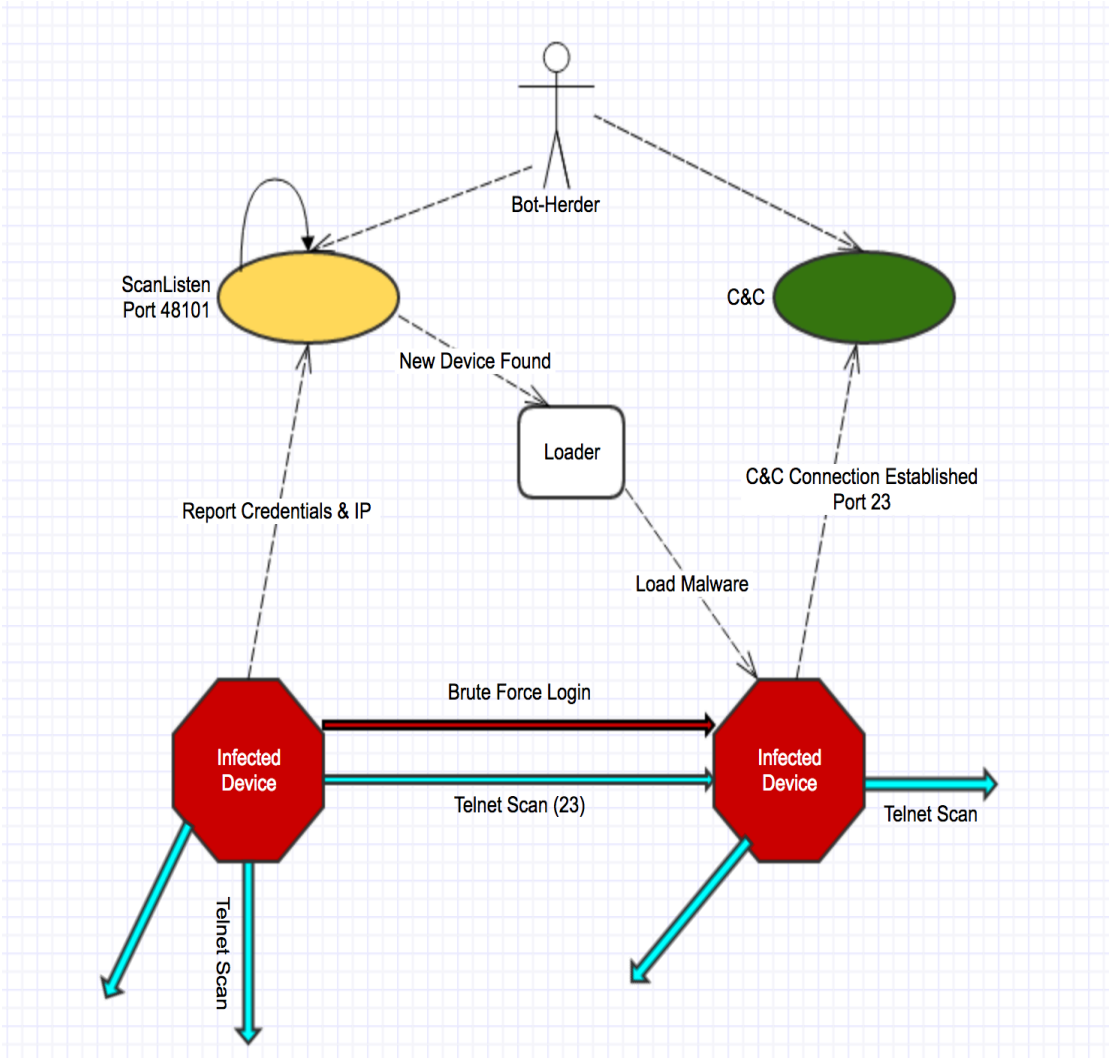


The user downloads the malware -

The user attempts to run the obtained malware and then remove it from the system -

The session ends here. This specific session is indicative of the sessions observed in the Telnet honeypots. The session proceeds very much as expected and results in the capture of Mirai malware.

In chapter three we discussed the operation of the Mirai botnet, the session above relates to the Loader phase. The diagram below should serve as a reminder to the reader of how Mirai operates after executing the malware. When executing the Mirai malware samples there wasn't any evidence of communication on port 48101. There were DNS queries to the sites www.gzxfr5axf7.com and www.gzxfr5axf6.com with responses of 69.64.147.242 and 38.68.24.94 respectively. There were also numerous Telnet connection attempts to a long list of IP addresses, similar to that described in the previous section.



Mirai Operation

IV) IoTPOT Malware

The developers of the IoTPOT honeypot shared hundreds of malware samples, captured with their IoTPOT honeypot, upon request. Sandbox analysis of a sample of these files showed a variety of CPU architectures, and a similar behaviour to the Mirai samples, more often than not attempting connections to a range of IPs via Telnet. The CPU architectures found included MIPSEL, ARM and x86 with x86 being encountered the most often. The network analysis often pointed to a botnet similar to Mirai, while in some cases the static analysis seemed to confirm this with the exact same credential strings and the same attack vector presents.

6.3 Summary

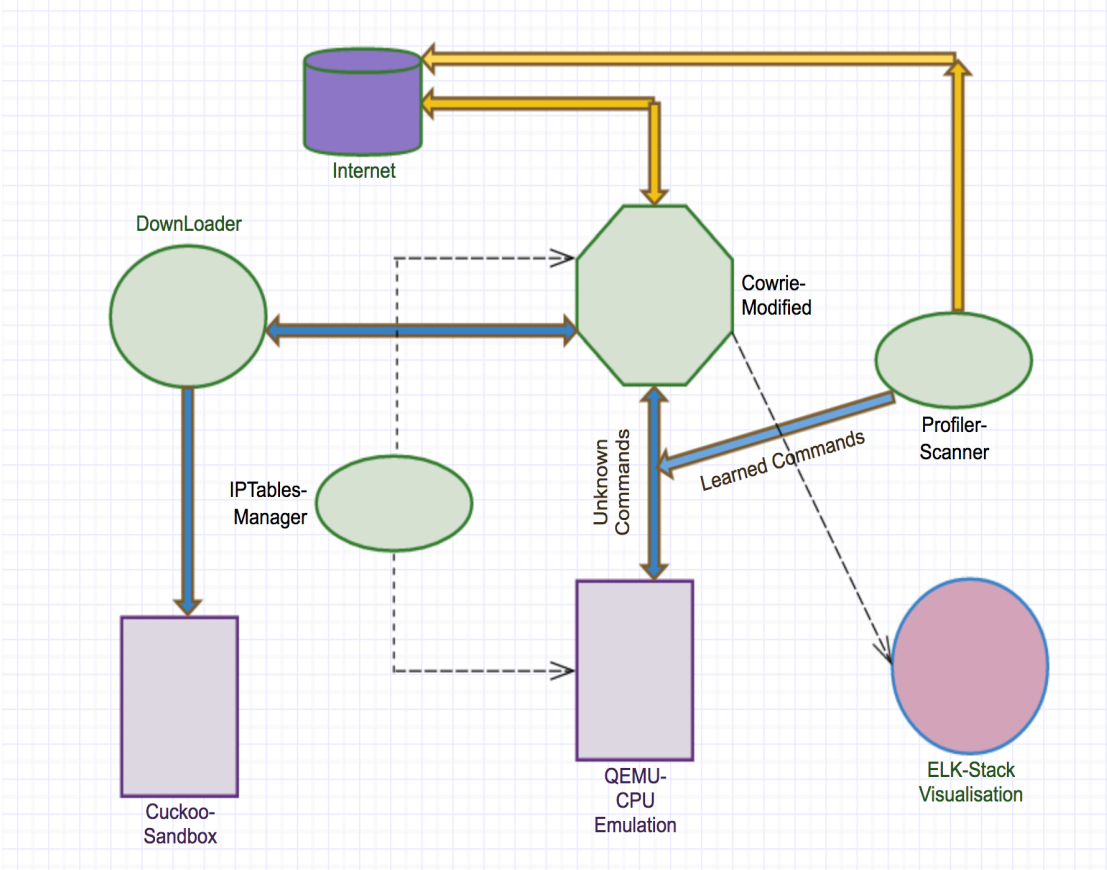
We can see from this chapter that the goal of enticing IoT botnets to interact with the system and then capturing the malware was achieved. The system was not a high interaction honeypot and therefore to analyse the malware collected the files were executed in a sandbox environment. This gave an insight into the network activity of an infected IoT device. Interestingly the malware binaries shared by the IoTPOT researchers appeared on the whole to operate in a similar manner.

Chapter 7

Conclusion

The project goals of enticing attacks from IoT botnets, monitoring the attackers during these attacks and obtaining the relative malware samples were all achieved. The captured samples were executed and analysed in a sandbox environment and it was possible to compare these samples with the shared IoTPOT samples. The report presented the state of the art in the area of honeypots and botnets. It then focused on the current threat of insecure IoT devices, the botnets that target them and the IoT honeypots designed specifically to observe and counteract this threat. With a deeper understanding of the field, reflecting on the experimentation carried out during the project combined with the many sources of literature reviewed, I would suggest that a dynamic open source honeypot tailored towards the IoT problem would be of great value. I believe the tools for the solution already exist. I briefly outline a potential solution below.

The diagram below suggests a possible design for such a honeypot system. The design is based on the non open source IoTPOT described in [36]. Crucially all the main components of the design below are open source with the source code available on Github. The main work involved in developing such a system would be the integration of all the components into a single piece of software that could be used out of the box. The IP Tables Manager and Profiler-Scanner components could be created simply enough with a scripting language such as Python and using Masscan for the banner grabbing. Looking to design and implement a system like the one described below could perhaps be an interesting project in and of itself, as there is currently no piece of open source honeypot software with the dynamic ability of the IoTPOT.



Dynamic IoT Honeypot Outline

Bibliography

[1] https://www.nytimes.com/2016/10/22/business/internet-problems-attack.html?_r=0

[2] https://www.honeynet.org/

[3] https://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots

[4] Proceedings of the 12th European Conference on Information Warfare and Security, Kuusisto

[5] Honeypots: Tracking Hackers – Lance Spitzner

[6] https://github.com/tnich/honssh

[7] AmpPot: Monitoring and Defending Against Amplification DDoS Attacks, Lukas Krämer, Christian Rossow et al.

[8] https://github.com/desaster/kippo

[9] https://github.com/micheloosterhof/cowrie

[10] https://github.com/DinoTools/dionaea

[11] https://github.com/foospidy/HoneyPy

[12] http://mushmush.org/

[13] Glastopf: A dynamic, low-interaction web application honeypot

[14] https://github.com/mushorg/conpot

[15] https://github.com/dtag-dev-sec/honeytrap

[16] http:// Sicherheitstacho.eu

[17] https://github.com/rep/hpfeeds

[18] https://www.theguardian.com/technology/2017/apr/28/facebook-google-conned-100m-phishing-scheme

[19] https://www.google.com/about/appsecurity/reward-program/]

[20] https://www.fireeye.com/current-threats/recent-zero-day-attacks.html

[21] A2: Analog Malicious Hardware - Kaiyuan Yang, Matthew Hicks, et al

[22] http://www.ana.net/content/show/id/38432

[23] Computer Networking: A Top-Down Approach, James Kurose

[24] GRE – RFC 1701

[25] https://www.us-cert.gov/ncas/alerts/TA13-088A

[26] https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html

[27] https://hello.neustar.biz/2016-soc-report-security-lp.html

[28] https://www.symantec.com/connect/blogs/international-takedown-wounds-gameover-zeus-cybercrime-network

[29] https://www.symantec.com/connect/blogs/international-takedown-wounds-gameover-zeus-cybercrime-network

[30] http://news.softpedia.com/news/dridex-banking-trojan-will-soon-target-crypto-currency-wallets-508041.shtml

[31] Using Malware Analysis to Evaluate Botnet Resilience, Christian Rossow

[32] An Analysis of the Zeus Peer-to-Peer Protocol, Dennis Andriess et al

[33] https://www.malwaretech.com/2017/04/the-kelihos-botnet.html

[34] https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

[35] https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html

[36] IoTPOT : Analysing the Rise of IoT Compromises, Yin Minn Pa Pa, Christian Rossow, et al.

[37] https://www.us-cert.gov/ncas/alerts/TA16-288A

[38] http://www.gartner.com/newsroom/id/3598917

[39] http://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html

[40] https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/

[41] http://blog.netlab.360.com/new-mirai-variant-with-dga/]

[42] Hajime: Analysis of a decentralized internet worm for IoT devices, Sam Edwards Ioannis Profetis

[43] https://securelist.com/blog/research/78160/hajime-the-mysterious-evolving-botnet/]

[44] https://arstechnica.com/security/2017/04/vigilante-botnet-infects-iot-devices-before-blackhats-can-hijack-them/]

[45] https://github.com/Cymmetria/MTPot

[46] https://support.microsoft.com/en-us/help/14223/windows-xp-end-of-support

[47] https://www.hedgehogsecurity.co.uk/2017/05/12/nhs-england-cyber-attack/]