# Capturing attacks on IoT devices with a multi-purpose IoT honeypot

*A thesis submitted in fulfilment of the requirements*

*for the degree of Master of Technology*

*by*

Krishnaprasad P


DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

May 2017

# Certificate

It is certified that the work contained in this thesis entitled "Capturing attacks on IoT devices with a multi-purpose IoT honeypot" by "Krishnaprasad P" has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

16.5.17

Dr. Sandeep Shukla

Professor

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

*May 2017*

# *Abstract*

---

Name of the student: **Krishnaprasad P** Roll No: **15111021**

Degree for which submitted: **M.Tech.** Department: **Computer Science and Engineering**

Thesis title: **Capturing attacks on IoT devices with a multi-purpose IoT honeypot**

Thesis supervisor: **Dr. Sandeep Shukla**

Month and year of thesis submission: **May 2017**

---

The past few years have seen a meteoric rise in the use of IoT (Internet of Things) devices. This has resulted in malicious attackers targeting IoT devices more and more. The reluctance of users to change the default credentials of such devices has made attacking the devices much more effective. A major example of such attacks being the mirai botnet attack on October 2016 that targeted DNS providers and rendered many major websites unavailable. To counter this rapid increase in IoT attacks, we propose a new IoT honeypot that can capture attacks coming through 4 common channels: Telnet, SSH, HTTP and CWMP. The attacks which are captured are then analyzed to find common patterns and gain intelligence.

# *Acknowledgements*

I would extend my sincere gratitude to Dr. Sandeep Shukla for guiding me in this project. I would also like to thank Rohit Sehgal and Nishit Majithia for their help and co-operation during various phases in the project. I am grateful to my friends for being there for me all the time. I am also thankful to my parents and brother for the love they have given me. Finally, I would like to thank the open-source community for providing me with lots and lots of help.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **IoT** | **I**nternet **o**f **T**hings |
| **DNS** | **D**omain **N**ame **S**ystem |
| **DDoS** | **D**istributed **D**enial **o**f **S**ervice |
| **DTK** | **D**eception **T**ool**k**it |
| **HTTP** | **H**yper**T**ext **T**ransfer **P**rotocol |
| **CWMP** | **C**PE **W**AN **M**anagement **P**rotocol |
| **SSH** | **S**ecure **SH**ell |
| **SCP** | **S**ecure **Co**p**y** |
| **SCP** | **S**ecure **F**ile **T**ransfer **P**rotocol |
| **TLS** | **T**ransport **L**ayer **S**ecurity |
| **PHP** | **P**HP: **H**ypertext **P**reprocessor |
| **HTML** | **H**yper**T**ext **M**arkup **L**anguage |
| **MHN** | **M**odern **H**oney **N**etwork |
| **RSH** | **R**emote **Sh**ell |
| **CPE** | **C**ustomer **P**remises **E**quipment |
| **ACS** | **A**uto **C**onfiguration **S**erver |
| **XML** | E**x**tended **M**arkup **L**anguage |
| **SOAP** | **S**imple **O**bject **A**ccess **P**rotocol |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **JSON** | **J**ava**S**cript **O**bject **N**otation |
| **TFTP** | **T**rivial **F**ile **T**ransfer **P**rotocol |
| **C & C** | **C**ommand **A**nd **C**ontrol |

*Dedicated to my parents, brother and friends*

# Chapter 1

# Introduction

Internet of Things(IoT) in its very basic sense just defines any physical device connected to the internet. The fact that the devices are connected to a network allows them to communicate with each other and share data. Any physical device connected to internet can be considered to be an IoT device. Popular examples include thermostats, refrigerators, televisions, health monitoring devices and home automation systems[21].

The past few years have seen a rapid increase in the use of IoT devices. More and more people are relying on IoTs in their daily lives. The increase in popularity of IoTs has also led to an increase in the number of attacks targeting such devices. The past year has seen multiple attacks targeting IoT devices. The majority of these are DDoS attacks which are mounted after infecting the devices which are then used for spreading the infection further. The infected devices are used to mount a DDoS attack on a target later.

The most prominent of such attacks came using mirai, a malware that attacked devices with weak login credentials and then used the infected devices to spread further. On October 21, 2016, a massive DDoS attack was mounted using mirai that targeted Dyn, a DNS provider.It affected a large number of internet users in USA. The same mirai was also used to mount attacks in Germany and UK[22].

To capture such attacks and more, researchers use honeypots.

## 1.1 Honeypots

A honeypot is a system that acts as a trap to lure and ensnare attackers. Generally it will be a system that will be made attractive to the attackers by making it appear vulnerable and which may contain information that would be of use to a malicious intruder but in reality, all the activities of the attacker will be logged and then used for analysis to gain intelligence and prevent further such attacks[13].

Honeypots can be classified according to various parameters.

### 1.1.1 Classification of honeypots

#### 1.1.1.1 Based on interaction

**Low interaction honeypots** usually present an attacker with an emulated set of services with limited functionalities. Such honeypots are mainly used to find data about the source of attacks rather than the methods used in these attacks. **High interaction honeypots** on the other hand tend to provide a fully-fledged system so that attackers can interact more with the system. This can help in gaining a lot of information about the attacker as he is free to execute commands and download files to the system. **Medium interaction honeypots** lie in between low-interaction and high-interaction honeypots in the level of interaction allowed with the user.

#### 1.1.1.2 Based on deployment

**Production honeypots** are primarily used by corporations and set up in the production network along with other servers. Such honeypots are generally low-interaction. **Research honeypots** are used by researchers for academic purposes. They are generally used to capture the activities of malicious attackers. They tend to be more interactive than production honeypots.

### 1.1.2 Honeynets

A honeynet is a network of honeypots. It is a highly controlled network used to contain and analyze attackers. A honeynet primarily has the following modules.

I. *Data Control*: Data control ensures that even if the honeynet is compromised, it cannot be used to attack other systems. For that, flow of data in and out of the honeynet is regulated.

II. *Data Capture*: This is the process of capturing the activity of an attacker who has accessed the honeynet and then storing this data for further analysis.

III. *Data Analysis*: Data analysis is the module where data collected is analyzed and then used to draw up conclusions. This can then be used to make changes to the honeynet if necessary[10].

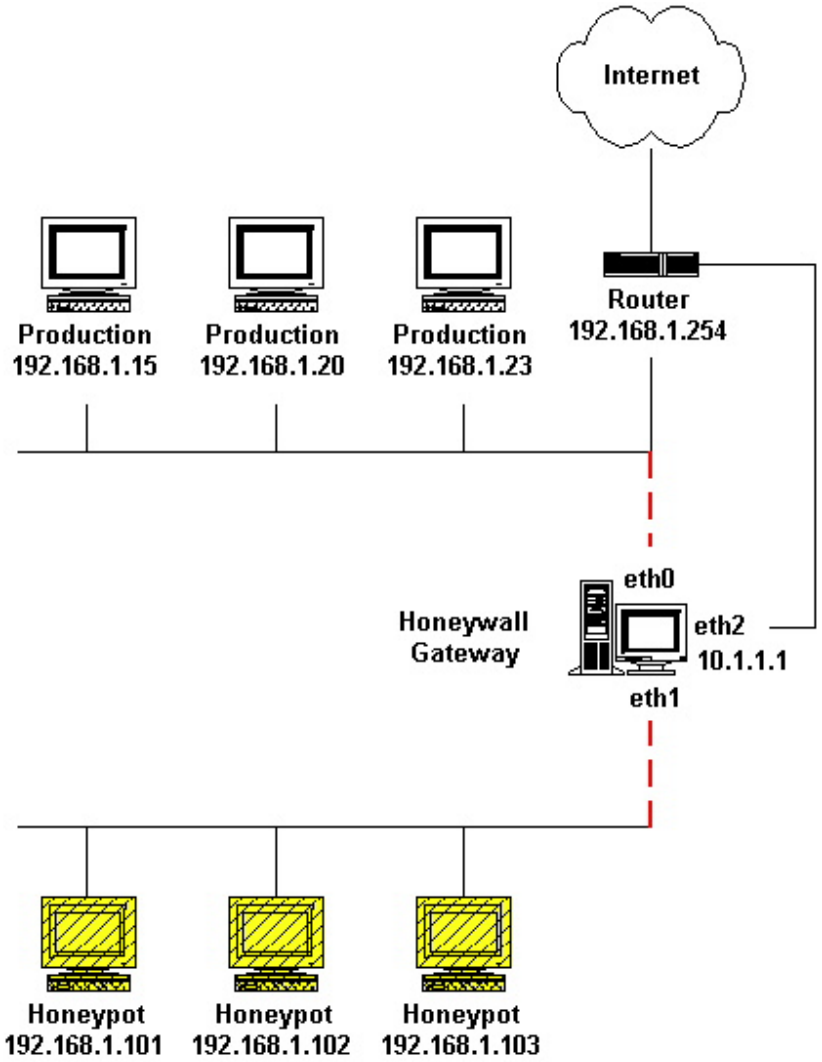
## 1.2 Objective

The aim is to analyze some of the existing attack patterns on IoT systems. Based on this, we design a honeypot for IoTs that can handle attacks via some of the major protocols used in IoTs. Once the honeypot is created, we deploy it and collect data. The collected data is then analyzed and patterns are then drawn from the data.

## 1.3 Organization

Chapter 2 consists of an overview about some of the more popular and common honeypots and their advantages and disadvantages. Chapter 3 consists of some data and analysis from the HonSSH honeypot deployed on a public machine. Chapter 4 discusses the design and implementation of our IoT honeypot. We also discuss the various frameworks and tools used in creating the honeypot. Chapter 5 includes the data obtained from this honeypot, its analysis and the intelligence that we obtained from it. Chapter 6 has the possible improvements that can be made to the honeypot.

FIGURE 1.1: Example of a honeynet



# Chapter 2

# Previous work

Although the concept of honeypots existed before 1990, it was not very popular. One of the earliest honeypots to be public was Deception Toolkit ( DTK ) in 1997. The DTK is capable of simulating a wide variety of unix vulnerablities on a system and is capable of masquerading as several different hosts as well and hence making it being capable of being a honeynet[15]. It was written in Perl and used TCP wrappers to process incoming service requests. Although it was simple, it was not high interaction and could be detected easily.

Then came the CyberCop Sting which was the first commercial honeypot to be released. Unlike DTK, it was a Windows honeypot.

CyberCop Sting simulates a subnetwork of routers and Solaris and Windows machines. The machines also simulate some services like Telnet. To an attacker, the honeypot looks like it is part of a network[9].

Things started picking up steam with the establishment of the Honeynet Project in 1999. The major objectives of the organization were to raise awareness of the existing threats on the Internet and to provide the necessary tools and methods to the general public so that they can also be aware and be actively part of the security community[16].

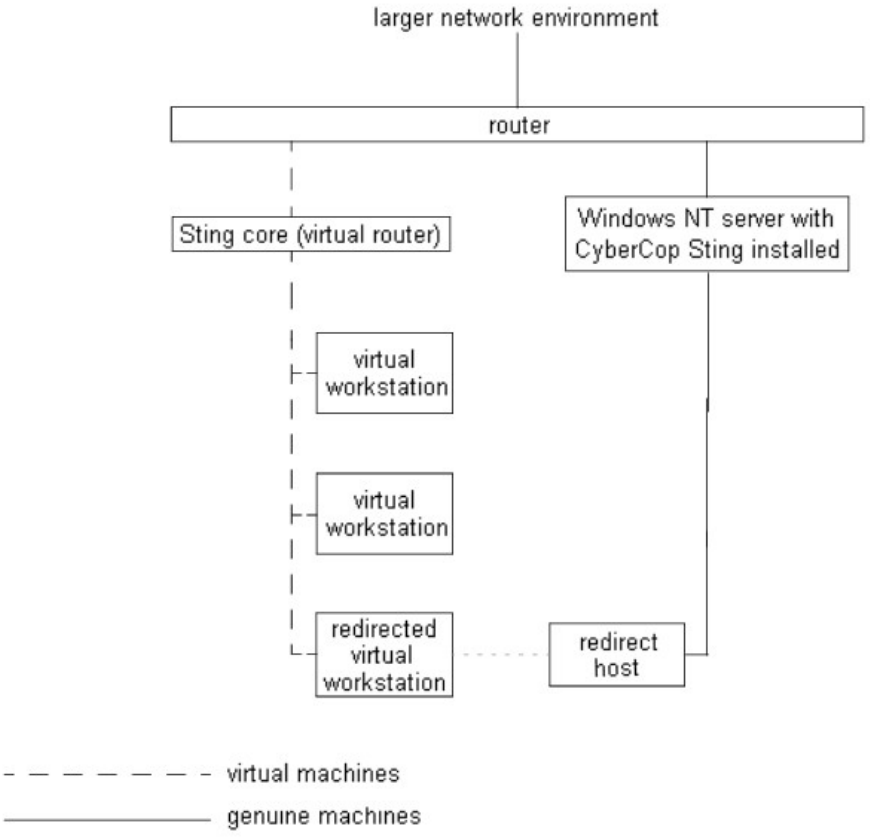
### 2.0.1 Popular honeypots

Currently there are many open source honeypots available. These can be freely downloaded and used.

#### 2.0.1.1 Kippo

Kippo is a medium interaction SSH honeypot designed to log brute-force attacks and shell interactions by an attacker. It is written in python using the twisted framework. Its model is very popular and has been used by others to create other honeypots. It simulates a shell and has a fake filesystem and the capability to add files to the filesystem. Although effective, it can be slow and hence can be detected[5].

FIGURE 2.1: CyberCop Sting



#### 2.0.1.2 Cowrie

Cowrie is a fork of kippo. It supports Telnet along with SSH. It supports more Linux commands and also emulates SFTP and SCP protocols[2].

#### 2.0.1.3 Dionaea

Dionaea is a python honeypot which uses libemu to detect shellcodes. It supports ipv6 and TLS. Dionaea is used to capture malwares by offering vulnerabilities to the attacker[1].

#### 2.0.1.4 Glastopf

Glastopf is a low-interaction web application honeypot capable of emulating several vulnerabilities which is used to gather data from attacks that target web applications. Glastopf works by sending the response expected by an attacker when he is trying to attack a web service. Vulnerabilities like remote file inclusion via a build-in PHP sandbox, local file inclusion providing files from a virtual file system and HTML injection via POST requests are emulated[11].

#### 2.0.1.5 Thug

Thug is a client side honeypot or rather a honeyclient. Instead of waiting to get attacked, a honeyclient acts as a client and actively seeks out malicious servers. Thug is written in python and emulates a web browser[4].

### 2.0.2 HonSSH

HonSSH is a high-interaction honeypot solution. Rather than emulating a server, HonSSH is more of a proxy. It lies in between the attacker and a honeypot and acts as an SSH proxy. It accepts connections from the attacker and then establishes connections with the honeypot. Any data from attacker is passed on to the honeypot

and vice versa. Meanwhile, all the data flowing through it is logged. HonSSH is inspired from kippo[12].

## 2.1 IoT Honeypots

### 2.1.1 Telnet-iot-honeypot

Telnet-iot-honeypot is a IoT honeypot to capture Telnet attacks. It is written in python and is mainly used to capture botnet malwares and binaries. Like cowrie, it simulates a Telnet session rather than offering a live terminal. The binaries are generally uploaded to VirusTotal for further analysis[14].

### 2.1.2 HoneyThing

HoneyThing was created as part of Honeynet GSoC project. It is a honeypot for Internet of TR-069 things. It simulates a router which has both a web interface and also supports CWMP protocol (CWMP is a protocol used to control IoT machines using an Auto Configuration Server). It emulates some popular RomPager vulnerabilities like Misfortune Cookie and Rom-0[29].

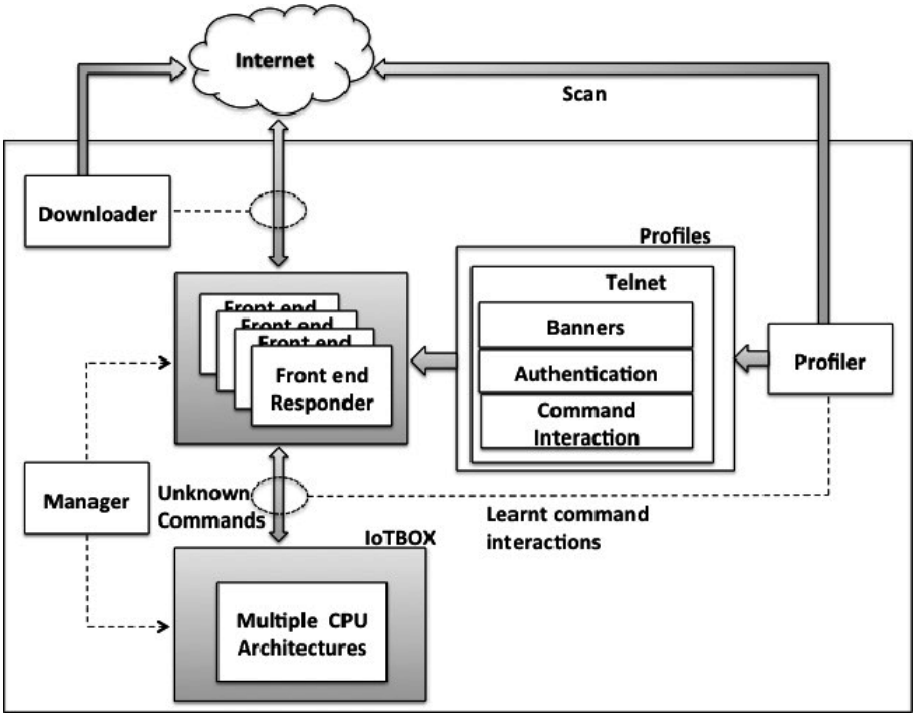
### 2.1.3 MTPot

MTPot is an open source honeypot by Cymmetria Research. It is a lightweight honeypot used to detect machines which are infected with the mirai malware and to also collect mirai malware samples if possible. Like other Telnet honeypots, it emulates a Telnet server and the settings are altered according to the version of mirai that is trying infect the honeypot[3].

### 2.1.4 IoTPOT

IoTPOT is a collaborative effort between researchers in German and Japanese universities. IoTPOT consists of an IoT honeypot and sandbox for Telnet attacks. It emulates Telnet services of various devices. It consists of two parts: a front end which is a low-interaction responder and a back end which is a high interaction virtual environment called IoTBOX. IoTBOX supports 8 CPU architectures including MIPS and ARM. The front end establishes a connection with the back end and transmits the commands from the attacker to the back end and sends the reply back from the back end to the attacker. It is unfortunately not open-source currently[28].

FIGURE 2.2: IoTPOT design



# Chapter 3

# Reconnaissance

Initially various honeypots were tried and tested to find attack patterns across attackers. Cowrie was tried in the early stages but we later realized that cowrie is really slow and very easy to detect. Then we came across Modern Honey Network (MHN) which is a honeypot management system with multiple honeypots under its control. We ran it for some time but it was being run inside the institute campus network. Unfortunately, no attacks were taking place as any attack from outside was blocked by the institute firewall and none of the machines inside the campus were compromised. So the MHN was ineffective.

Then we decided to try HonSSH which is basically a ssh proxy. We ran a virtual machine which was running Ubuntu in the back end. HonSSH used to listen on port 22 and any connection to port 22 was forwarded to port 22 of the Ubuntu virtual machine. The whole honeypot was set up on a machine running Ubuntu 16.04 and this machine was kept in the public network instead of the campus network. This way, we managed to attract a lot of attacks.

The honeypot was run for a period of 10 days, from 17th December to 27th December. There was a total of 181 different hosts which came and attacked the honeypot in this period of time.

| Country | Count |
|---|---|
| Vietnam | 49 |
| Argentina | 31 |
| France | 16 |
| Iran | 11 |
| India | 10 |

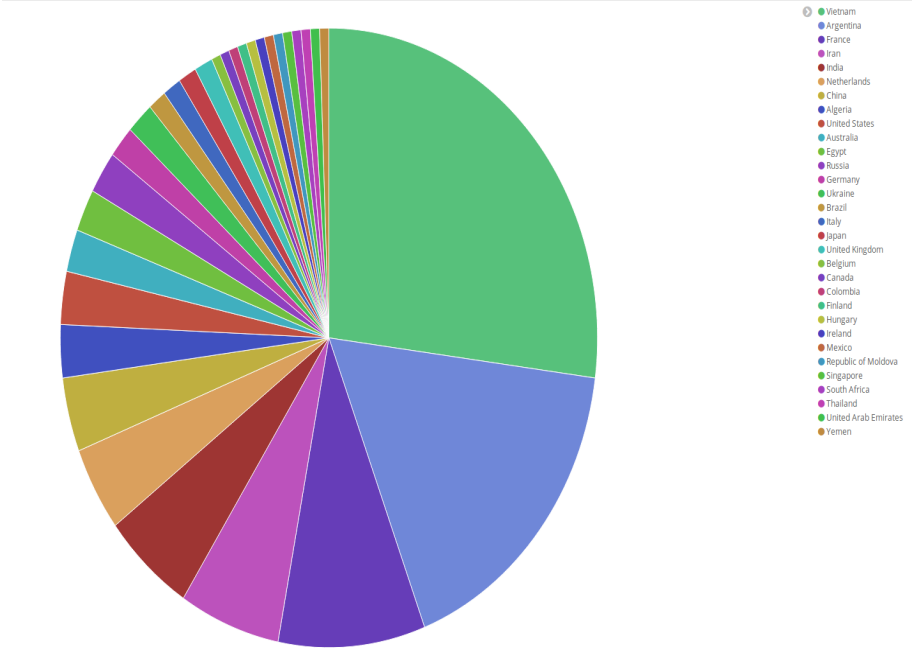
TABLE 3.1: Top 5 countries from which attacks originated

## 3.1 Attack using our honeypot

During this time, our SSH honeypot was used to attack other systems. This happened due to the fact that outgoing connections were enabled in our honeypot. So any attacker who gained accessed to the honeypot can then use it to connect to other machines and then mount attacks on them.

The original attack came from 140.164.14.40 which is located in Italy. The attack followed the following sequence.

FIGURE 3.1: Distribution of attacks by country



1. It tried to gain access to the system by trying various credentials. This step is common to most of the attacks.

2. Once access was gained to the system, a zip file was downloaded using the wget command.

3. The zip file was then extracted and the original file was deleted.

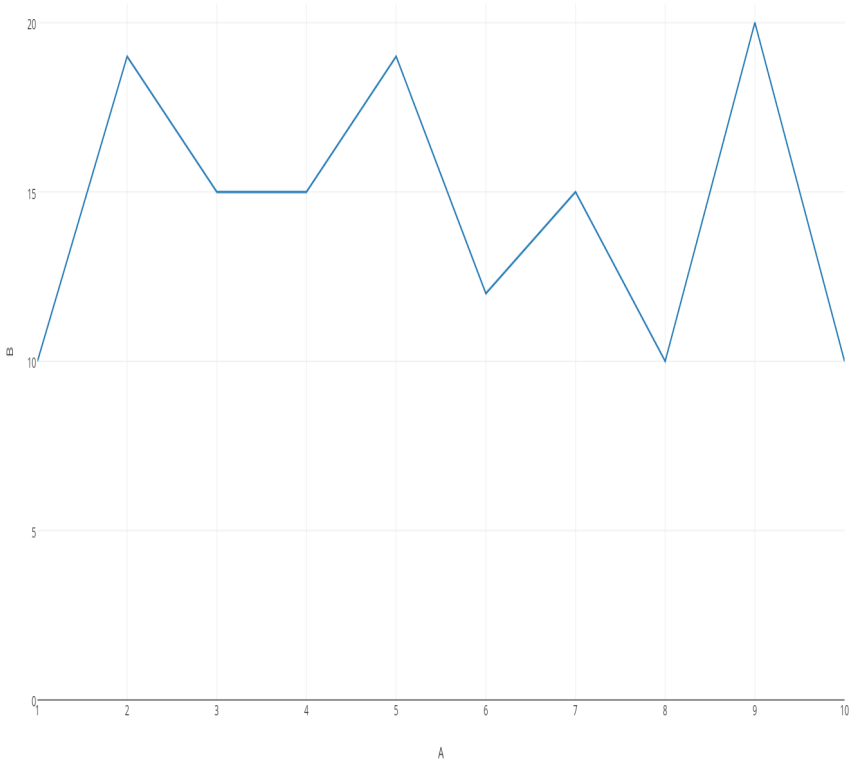
4. A file was created which stored the possible login credentials for a system.

5. IP address ranges 185.*.*.* , 188.*.*.* and 178.*.*.* were then scanned and the login credentials from the file was tried on these hosts.

6. If any of the credentials worked, then the IP address along with the credentials were saved to another file.

FIGURE 3.2: Attacks per day

X-axis represents the day and Y-axis represents the number of attacks



7. All the traces of the attack were removed in the end by deleting the files downloaded.

The following binaries were used in the attack.

- ssh2banner : This binary is used to fetch SSH banners. It takes an IP address range as an argument and stores the SSH banners from the hosts in this range to a file. This is done to check whether the servers are valid server or not.
- bssh2z : This binary is used to mount a brute-force attack on servers. the credentials used are taken from the credentials file created. Two output files are created by this binary: One which stores the IP addresses which have busybox shells and another which stores valid credentials for each host.

These binaries were not accessible to a common user. It required a password (a VIP_CODE) as an argument to excute. These VIP_CODES were fetched at run time from a server(151.80.25.88/vip.php which is located in France) and then used along with the binary.

## 3.2 Another notable attack

This attack was conducted in 2 phases.

1. In the first phase, the attacker used host 151.25.245.23 (Location: Italy) to attempt various credentials to log in to the system and then download a zip file in the honeypot using SFTP. Once the file was downloaded. the attacker disconnected from the system.
2. Later the attacker logged in using another host. The IP address was 62.141.107.90 (Location: Russia) and the attacker created a username-password file and then scanned an IP address range(212.*.*.*) and tried the combinations on those hosts. Any successful logins were then recorded to another file. Later, all the traces of the attack was deleted.

## 3.3 Conclusion

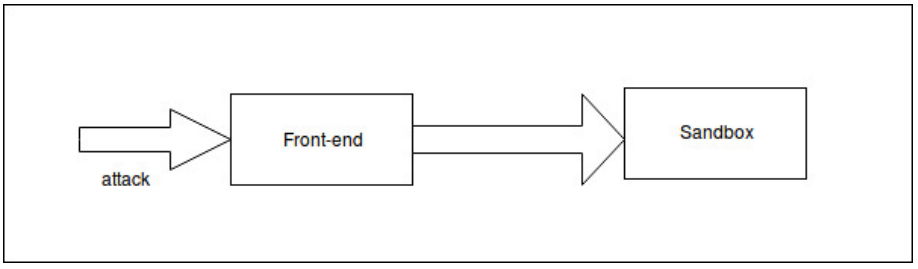
The fact that we managed to attract so many attacks in such a short period of time made us realize us how dangerous the internet is and how vulnerable our systems are to such attacks. Since most of the attackers were trying on weak credentials to gain access to the honeypot, it was decided to use weak credentials for our IoT honeypot.

# Chapter 4

# Honeypot Concept and Design

The main inspiration for the honeypot came from IoTPOT, a Telnet honeypot. Its basic design was simple but elegant. We had also used a similar design in our SSH honeypot during the reconnaissance phase.

FIGURE 4.1: IoTPOT basic design




## 4.1 Basic design

There are 2 main components for our honeypot. The first component is the front end of the honeypot which will be directly interacting with the attacker. In our case, we have 4 proxies running in the front end, each handling one protocol each. The protocols are

- SSH
- Telnet

- HTTP
- CWMP

The front end also collects and aggregates data from the attacks.

The back end can generally be a single machine(physical or virtual) or a network of machines that can handle the 4 protocols. Any request to any of the 4 protocol services will be forwarded to the corresponding back end module by the front end. The respective modules generally process or execute these queries. The response obtained is passed on to the front end. The front end then forwards the response back to the attacker. The attacker is unaware of the fact that the front end exists and is logging all the activities. He thinks that he is directly interacting with the back end module.

## 4.2 Front end

An IoT device generally has one more more of Telnet, SSH, HTTP and CWMP protocols enabled. This was the reason for making a honeypot capable of handling these four protocols.

### 4.2.1 Protocols supported

The following protocols are supported by the front end.

#### 4.2.1.1 Telnet

Telnet protocol is used for bidirectional text-based communication. For this, a virtual terminal connection is implemented in Telnet. The name “Telnet” stands for “teletype network”. It uses TCP and Telnet data consists of both the data sent by the users and also the control information for the current communication channel. Port number 23 is used for the server side connections. Telnet does not encrypt any data send during a session. So it is being used less and less these days and have

been replaced with more secure protocols like SSH. Although new and more secure versions of Telnet have been developed, they are not very popular. Despite the drawbacks, it is still used by a lot of IoT devices due to the fact its implementation is relatively simple[25].

#### 4.2.1.2 SSH

SSH stands for "Secure Shell". Unlike Telnet, it sends data via an encrypted channel. It was developed to be an alternative to insecure protocols like Telnet and RSH. It is generally used for remote login to a system. Public key cryptography is used for encryption here. Servers use port 22 to accept connections from a remote user. Although it is commonly used in servers, SSH is still not that common in IoTs although its popularity is growing in this respect[24].

#### 4.2.1.3 HTTP

The Hypertext Transfer Protocol (HTTP) is an application layer protocol which follows the request-response model. An HTTP client submits a request to the HTTP server during the communication. The server processes the request and then sends a response back to the HTTP client. The response could contain the status of the request and also the data requested by the client. All IoT devices may not have an HTTP interface but routers usually do have one which is used to manage the configuration[20].

#### 4.2.1.4 CWMP

CWMP (CPE WAN Management Protocol) is used for communication between the Customer Premises Equipment (CPE) and an Auto Configuration Server(ACS). It is also known TR-069 which is the technical specification for the protocol. It is an application layer protocol for remote management of devices. The protocol is based on SOAP and HTTP. The basic protocol used in CWMP is HTTP where XML messages which are part of SOAP protocol is sent using HTTP protocol[26].

Although CWMP is not used in normal computers, it is prevalent in IoT devices.

### 4.2.2 Machine Requirements

The front end has 4 scripts running. Each script acts as a proxy for one of the following protocols: SSH, telnet, HTTP and CWMP. All the scripts are written in python and uses the twisted framework. The scripts write the data collected from the attackers into log files. Data is also pushed to Elasticsearch which can then be visualized using Kibana.

FIGURE 4.2: CWMP protocol

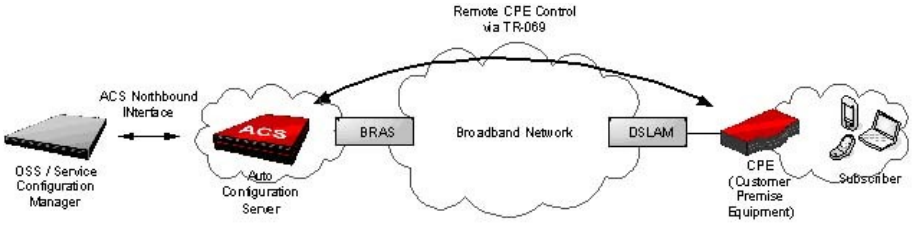
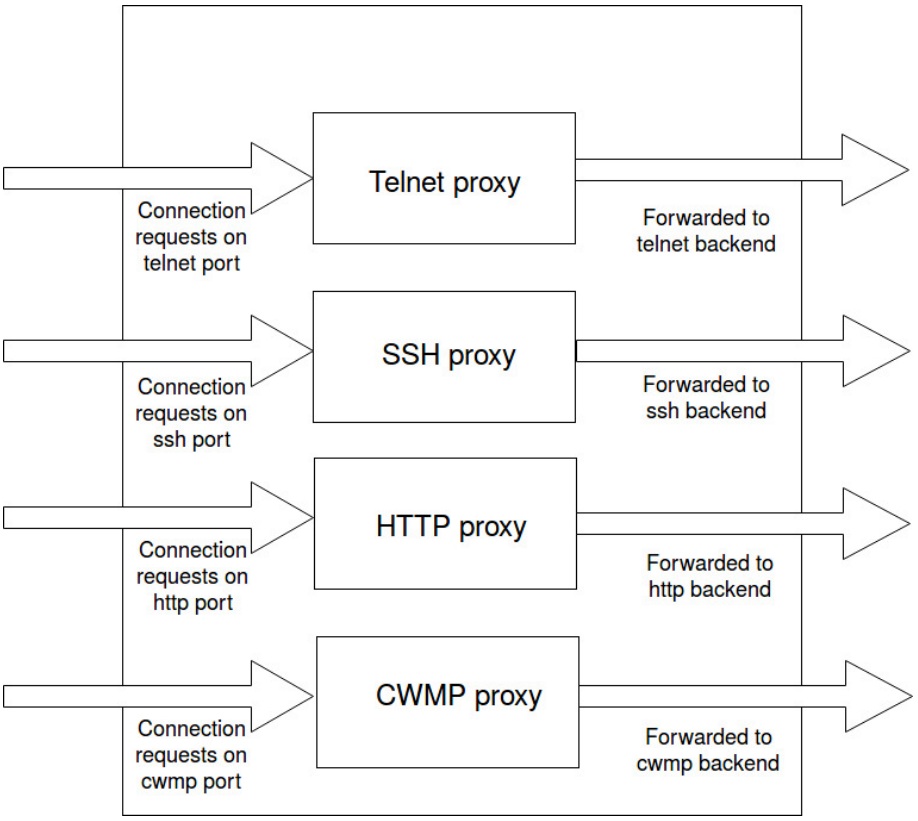



FIGURE 4.3: front end design



#### 4.2.2.1 Twisted

Twisted is an event-driven networking engine written in Python. Event-driven programming is a paradigm where the flow of the program is decided by the events that happen, like a connection request or a mouse click. This is implemented using an event loop that waits for events and then calls a relevant event handler to take care of the event. The event loop used in twisted is called the reactor loop. Twisted code is written in the form of callbacks which is called by the twisted framework[27]. Logical protocols and physical transport layers are completely separated here. Most of the major network protocols(both application and transport) are implemented in twisted. Some existing honeypots like kippo and cowrie already use the twisted framework in their code.

#### 4.2.2.2 Python Libraries used

In addition to the twisted libraries, the following python libraries were also used.

**ConfigParser** ConfigParser is used to parse the configuration files for the proxies.

**geoip2** This provides an API to use GeoLite2 databases which are IP geolocation databases.

**pyes** pyes is a python library used to provide APIs for using Elasticsearch.

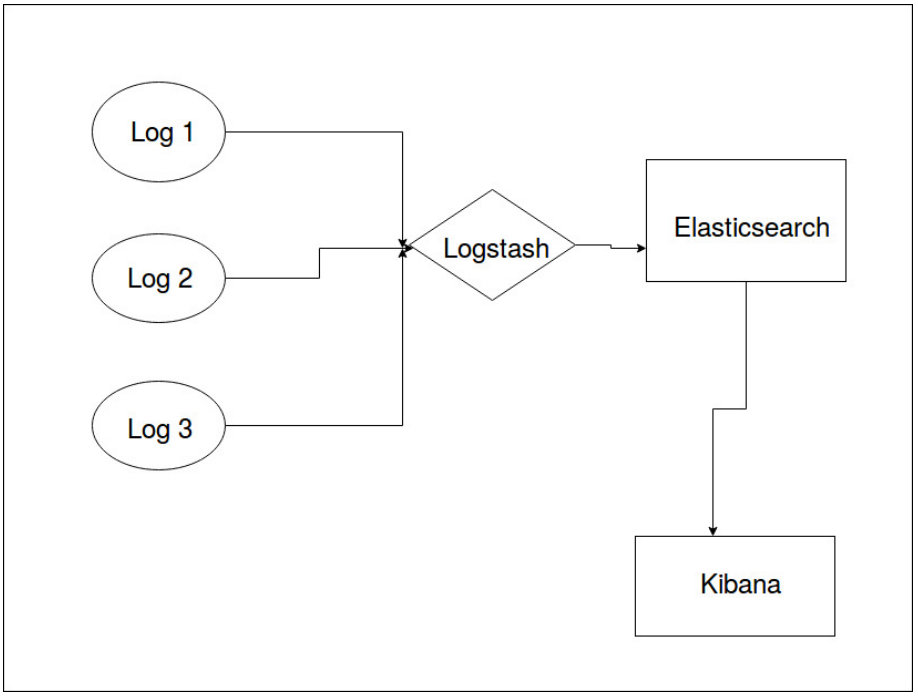
#### 4.2.2.3 ELK stack

ELK stack consists of Elasticsearch, Logstash and Kibana.

**Elasticsearch** Elasticsearch is a search engine based on Lucene. Lucene is a free and open source information retrieval software library. Elasticsearch is open-source and written in Java. It provides a really powerful text search capability. It uses

schema free JSON documents and also has an HTTP interface. Elasticsearch acts as the data storage module for the back end[19].

FIGURE 4.4: ELK stack




**Logstash** Logstash is a tool for log data intake, processing, and output. It is an open source data processing tool that takes data from a variety of sources at the same time, transforms it, and then sends it to a variety of possible outputs, most commonly Elasticsearch. Logstash is able to support a variety if logs, from web to system logs. All the logs are processed in a continuous, streaming fashion. Logstash, while sending the data from input to output filters, splits the data into fields that makes it more convenient to store data[7].

**Kibana** Kibana is an open source data visualization plugin for Elasticsearch. It provides visualization capabilities on the content stored on Elasticsearch. Histograms, line graphs, pie charts and many more visualization methods are supported. Geospatial data can also be inserted on maps. Timelines are supported which enable time series analysis of the data. There are also plugins to help do unsupervised machine learning on the data[8].

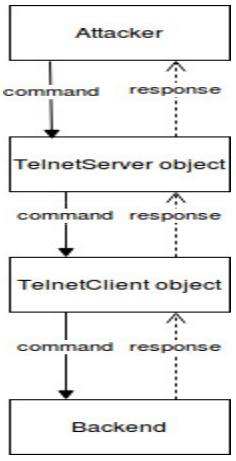
### 4.2.3 Code implementation

The implementation of all 4 proxies are discussed in detailed below. Any data in Elasticsearch is stored in an index called attacks.

#### 4.2.3.1 Telnet proxy

The code uses StatefulTelnetProtocol[17] to implement the protocol class for the proxy. Each time a new client connection comes, a new object is created in the TelnetSeverFactory which is an implementation of Factory class. The object is an instance of TelnetServer. Any object in the TelnetServer uses the TelnetServer-Protocol as its protocol instance. TelnetServerProtocol is an implementation of StatefulTelnetProtocol. Each client connection has a corresponding TelnetServer object. Here it acts as a server object. Once the client is connected, a new object from TelnetClient is created corresponding to each TelnetServer object. The protocol used for the TelnetClient is TelnetClientProtocol. The TelnetClient object then established a connection with the backend Telnet handler.

FIGURE 4.5: Telnet proxy




Any data from the client is handled by the TelnetServer object. It takes the data, writes it into the log files and then forwards it to the corresponding TelnetClient object. The TelnetClient object then takes this data and sends it to the backend object. The backend object handles this data and sends a response back to the TelnetClient object. This object then forwards this data back to the TelnetServer object which again forwards this to the client.

The proxy also has an option to write data directly into the Elasticsearch database. 3 types are created in the "attacks" index: "connections" which will store the time of the initial connection and the IP address of the attacker, "telnet_connections" which will store the IP address, the connected time and the disconnected time of the attacker and "telnet_auth_details" which will store the credentials tried by the attacker while logging into the system.

#### 4.2.3.2 HTTP proxy

HTTP proxy is already implemented in python using the ReverseProxyResource class[18]. It is basically a resource that renders results obtained from another server. The ReverseProxyResource unfortunately does not log the attacks in a manner that we need it to. So we inherited ReverseProxyResource into a class called MyReverse-ProxyResource. This contains the same methods as the ReverseProxyResource but extra logging capability is added where all the parameters in the HTTP resource object are written to the log files.

Like the Telnet proxy, there is an option to write data into Elasticsearch. 2 types are created in the "attacks" index: "connections" which will store the time of the initial connection from the attacker and his IP address and "http_requests" which store the IP address and the time of the request and also the necessary data about the request content.

#### 4.2.3.3 CWMP proxy

CWMP proxy is a modification of the HTTP proxy code. But extra capability is added to parse the CWMP commands. Hence there are 2 logs created for each attack: one which contains the raw HTTP data received by the proxy and one for the parsed CWMP output from the logs.

In Elasticsearch, 2 types are created in the "attacks" index: "connections" which will store the time of the initial connection from the attacker and his IP address and "cwmp_requests" which store the IP address and the time of the request and also the necessary data about the CWMP request content.

#### 4.2.3.4 SSH proxy

The SSH proxy being used is a fork of HonSSH. The experience of using HonSSH convinced us that it is a really well written and flexible proxy that can be used for various situations. It uses the twisted conch library which is the twisted library of SSH protocol.

The code is basically divided into 2 parts: pre-auth and post-auth. Pre-auth is the phase where the HonSSH accepts the connection from the attacker and then establishes a corresponding connection to the honeypot. Honssh server accepts the credentials from the attacker. Depending on the settings, it can either send the original credentials sent by the attacker or send a given set of credentials to honeypot so that it can get authenticated.

After the authentication is done, the post-auth phase comes into play. Any command sent by the attacker is decrypted by the HonSSH server, then encrypted using the key of the honeypot and sent to the honeypot. The response from the honeypot is decrypted by HonSSH and then encrypted again using the key of the attacker and then sent to the attacker.

Honssh has a separate module for output. This module supports various plugins that can write output into various formats upon the happening of an event. Currently there are plugins for textlog, slack, hpfeeds, email, mysql etc. We forked HonSSH and wrote a new plugin that would support Elasticsearch in the output format. This plugin pushes data into the elasticseach each time a connection is made, a connection is closed and a login happens successfully or unsuccessfully.

Like Telnet, 3 types are created in the "attacks" index: "connections" which will store the time of the initial connection from the attacker and his IP address, "ssh_connections" which will store the IP address, the connected time and the disconnected time of the attacker and "ssh_auth_details" which will store the credentials tried by the attacker while logging into the system. Data is pushed into "connections" when a new connection is made. Data is pushed into "ssh_connections" each time a connection is closed and data is pushed into "ssh_auth_details" each time a login attempt is made, be it successfully or unsuccessfully.

#### 4.2.3.5 Docker

Dockers are virtual containers that can be used to run applications. Unlike other virtual containers, docker containers are easier to set up and run and they are comparatively light-weight. A complete application along with the necessary dependencies for the application can be combined together in a docker container. Such a container can be run in any Linux system that supports docker. This is much more comfortable and easier than manually installing dependencies on a machine to run an application.

#### 4.2.3.6 ELK stack

The ELK stack is implemented using docker. 3 docker containers are created , one each for Logstash, Elasticsearch and Kibana.

First, the Elasticsearch docker is created. Version 5.2.2 of Elasticsearch is used here. The base image is alpine which is used for it's security, simplicity and resource efficiency. All the necessary dependencies are installed and then Elasticsearch is installed. A user 'elas'is created for running Elasticsearch. Port 9200 on which Elasticsearch runs by default is then exposed for access by other docker containers. A folder "esdata" is created in the host machine to store the Elasticsearch data so that it can be accessible to other docker machines also.

Logstash docker is installed next. For this, Logstash:5.2.2 docker image which is already available is used. Logstash requires custom configuration files for processing individual logs. The configuration files are in a directory inside the host machine and Logstash is configured to use those files while processing.

Kibana is configured at the end. Again , alpine is used as the base image. Once the Kibana is installed in the docker container, port 5601 is exposed so that it can be used to visualize the data.

## 4.3 Backend

The front end is flexible enough to support any kind of back end as long as it is accessible from the front end and can support the protocols SSH, Telnet, CWMP and HTTP. It would be advisable to connect an IoT or multiple IoTs as part of the back end but even a normal server can be connected. Since the front end is a bunch of proxies, the 2 components are independent of each other. For our experimental setup, we set up a bunch of docker machines that can simulates IoTs or/and their protocols.

### 4.3.1 OpenWRT

OpenWrt is a Linux distribution for embedded devices. OpenWrt provides a full fledged filesystem with package management also enabled. This makes it easier for a user to configure and customize OpenWrt by downloading any required packages using the package management system.

The main components for OpenWrt are Linux and BusyBox. All components have been optimized so that OpenWrt can be run on small embedded machines with lower memory capability and processing power[23].

OpenWrt can generally be configured using a command-line interface. Ash shell is used as the interface here. Web interface can also be used for configuring the system. LuCI is the common web interface supported by OpenWrt. Extra packages can be downloaded and installed using the opkg package management tool.

OpenWrt can run on various types of devices, like routers, tablets, phones and laptops. It is also possible to run OpenWrt on personal computers, which are most commonly based on the x86 architecture.

After trying various operating systems for embedded systems, we decided to use OpenWrt. But we were unable to obtain an OpenWrt image that is flexible enough for our use. The images available on the internet had a specific problem: OpenWrt by default had an issue that initially when no root password is set, only Telnet is enabled and SSH is disabled. When a root password is set, SSH service can be started but

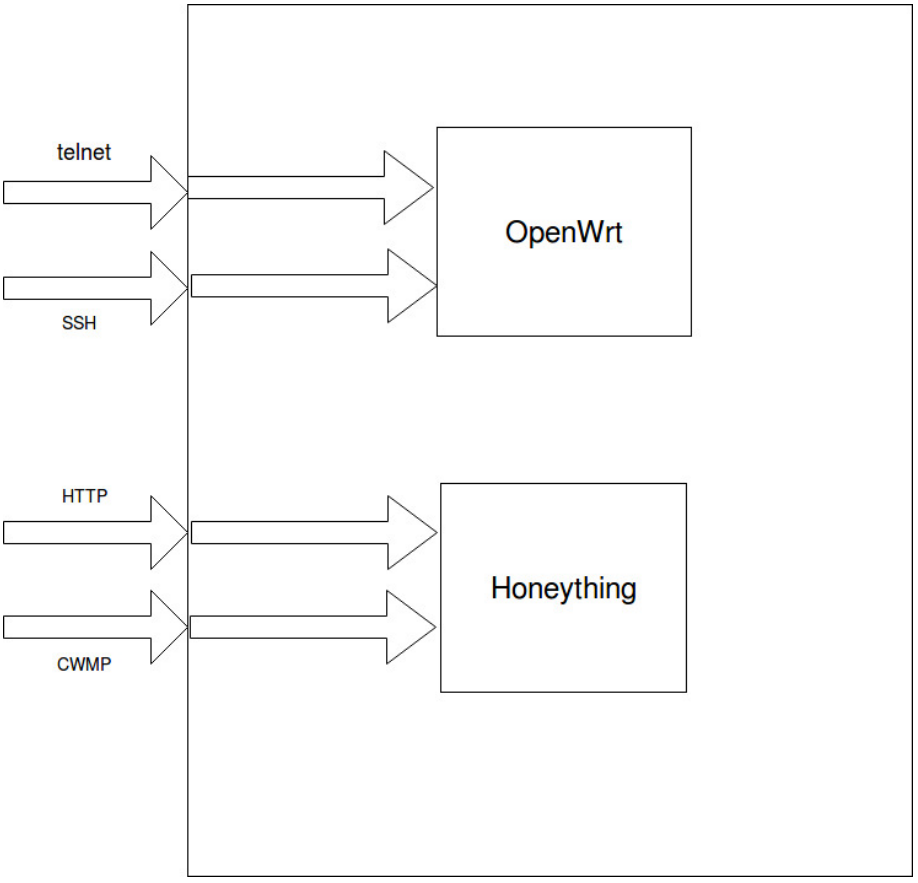
at this point Telnet would be disabled. To circumvent this issue, an OpenWrt image was build from scratch using the build tools available in the OpenWrt repository. With this new patch, both SSH and Telnet could be supported by the OpenWrt image simultaneously.

Once the image was built, it was imported to a docker machine and then pushed to docker repositories so that the docker image can be downloaded and run quickly. This docker image can be used as the back end for both Telnet and SSH services. Even a network of such docker machines can be created to simulate an IoT network.

### 4.3.2 EasyCWMP

The aim of EasyCwmp project is to create a CWMP client for OpenWrt that is lightweight and easy to use. The EasyCwmp design has 2 parts:

FIGURE 4.6: backend design: Configuration 1



- EasyCwmp core: it includes the TR069 CWMP engine and it handles the part where communication takes place beween the client and the ACS server. The code is written in C.
- EasyCwmp DataModel: DataModel of TR-069 is supported along with other DataModel standards such as TR-098 and TR-181.

The main aim of the design was to bring a separation between the CWMP method execution and the CWMP engine[6].

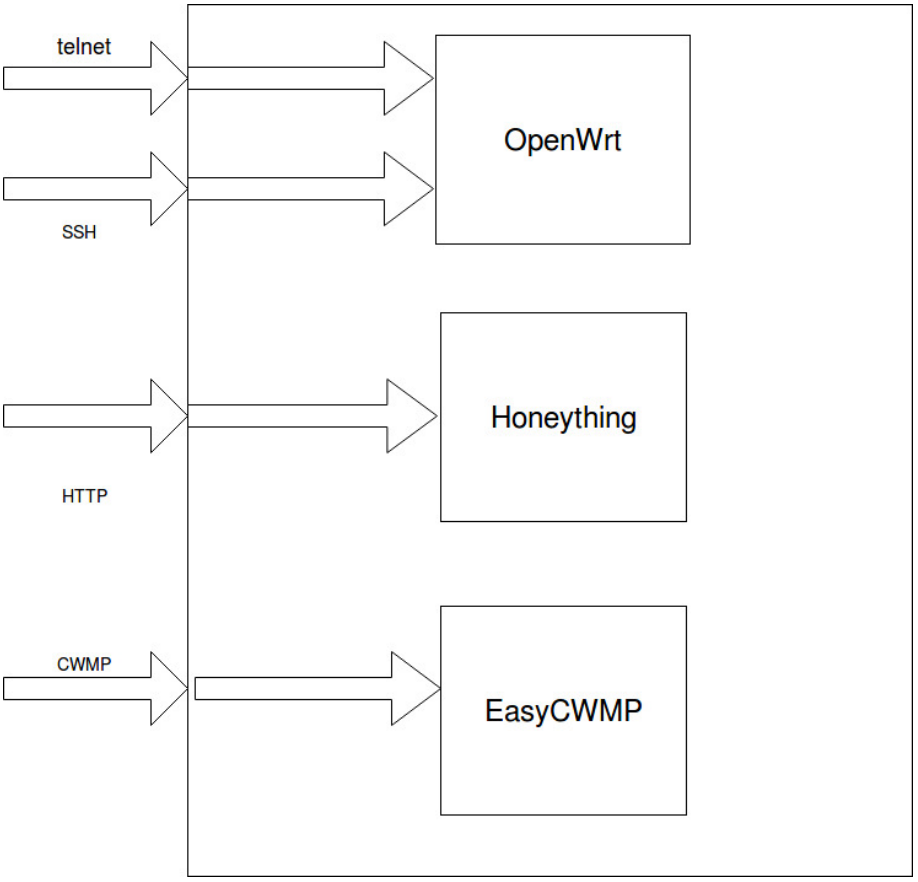
Although EasyCWMP was made specifically for OpenWRT, we were unable to configure it on our OpenWRT machine. But EasyCWMP can be installed on a normal computer system too. And since an attacker only interacts with EasyCWMP interface and not the shell, it would be hard for him to identify whether the system is an IoT or not. So we decided to configure EasyCWMP on an Ubuntu 16.04 machine and set a username and password. This image was also dockerized and uploaded on the docker hub.

### 4.3.3 HoneyThing

Honeything as discussed earlier is a honeypot for Internet of TR-069 things. It provides an HTTP interface for a TP-Link router which needs credentials for access. Similarly, it also acts as a CWMP client with a custom ACS server configured[29].

It was tough to configure HoneyThing on an OpenWrt machine. So like the EasyCWMP docker image, an Ubuntu 16.04 machine was used and both HTTP and CWMP service were enabled in the image. This was also dockerized for further use. Both the HoneyThing image and EasyCWMP image were used alternatively as the CWMP backend while the HoneyThing image was exclusively used in the case of HTTP backend.

FIGURE 4.7: backend design: Configuration 2



# Chapter 5

# Analysis and Conclusions

## 5.1 Analysis

The honeypots were deployed on different locations for a sustained period of time. In total, attacks came from total of 6774 hosts during this time period.

## 5.2 Telnet attacks

In Telnet, the attacks came from 4140 different IP addresses. A lot of them were mainly login attempts that tried to brute force through the credentials and got disconnected in between.

Most of the attacks on Telnet followed a general pattern.

#### 5.2.0.1 Pattern obtained

1. For login, they generally brute force through a fixed set of credentials which tends to be common for most of the attackers.

2. Once the attacker is logged in, he checks whether the system is an IoT and not a normal system or honeypot. This is usually done in two ways.

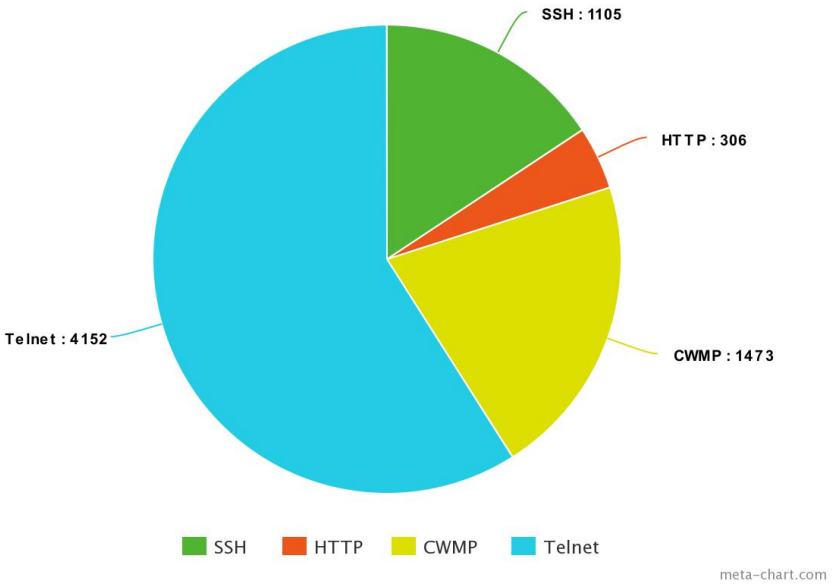
(a) The attacker executes "/bin/busybox <random-string>".A BusyBox system responds with "applet not found". If BusyBox is not enabled, then the system will respond with "bash: /bin/busybox: No such file or directory".

(b) executes something like "echo -e \148\141\171\146\147\163". A honeypot will most probably not process this command. Even a normal system will respond with \148\141\171\146\147\163. A BusyBox device will however respond with "hayfgs". Each number represents the ascii value of a character in its octal form.

The main reason for this is that BusyBox supports evaluation of slash-escaped characters in both echo and the shell built-in printf commands. The first method may not detect a normal server if BusyBox is also enabled in the server. But this method helps in recognizing IoTs because even if BusyBox is enabled, the primary shell will be processing the echo command and hence will not escape the characters.

3. It then tries out wget or TFTP to see if it works. If yes, use them to download the binaries. Wget and TFTP are the methods usually found on IoTs. Methods like SFTP are rarely found on less powerful devices.

FIGURE 5.1: Distribution of attacks per protocol

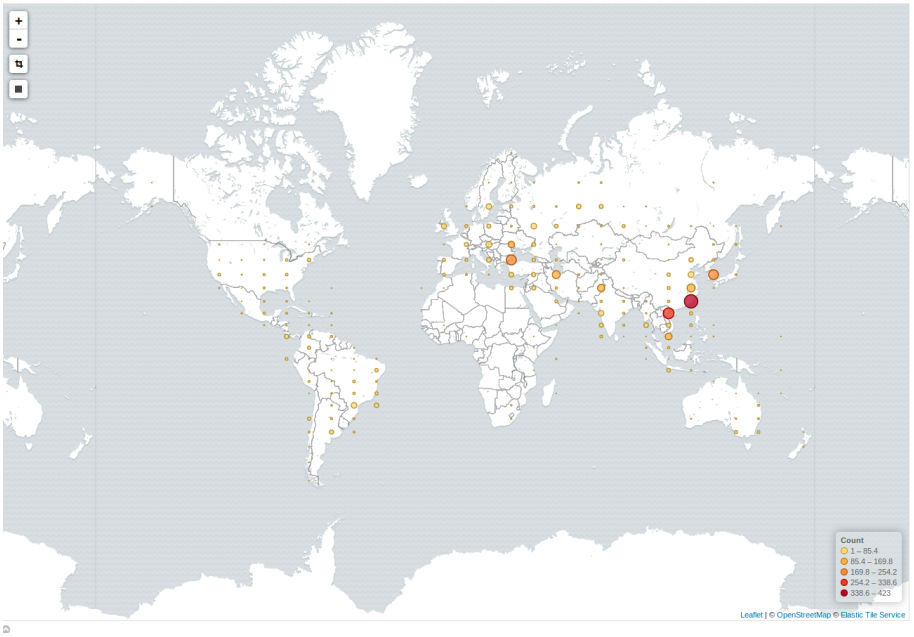


4. If not, echo the file content directly into the file.
5. Run the binary in the file.
6. Delete and remove all the traces of the attack.

### 5.2.1 Some major attacks

Some major attacks that came via the Telnet protocol is described here.

FIGURE 5.2: Geographical map based on the number of attacks




#### 5.2.1.1 Attack 1

This attack happened from 4 different hosts.

1. After gaining access to the server, the attacker executes executes "echo -e \147\141\171\146\147\164".
2. If the response is \147\141\171\146\147\164, then the connection is disconnected. Else, if the response is "gayfgt", then system is reported back to a command and control server.

3. The credentials of the system are most probably stored for further use later to mount DDoS attacks.

This is very similar to the elf_bashlite.a attack. Therer are various versions of this malware available. A source code similar to the attack was found here:

- client.c, server.c

#### 5.2.1.2 Attack 2

In this attack, after logging into the machine, the attacker finds all the partitions and tries to write the string "kami" to a file called ".nippon". This is to check whether the partition is writable or not. After that, the binary is saved to a file by directly echoing it. There were multiple attacks from the same ip in a span of 2 hours.

#### 5.2.1.3 Attack 3

This is similar to previous attack. This also echoes the binary into a file called "upnp". Although this file name is common to a lot of attacks, the contents are different in this attack. Although this was submitted to VirusTotal, no matches were found in the existing database.

#### 5.2.1.4 Attack 4

This attack happened from 220 different ips and hence was the most common attack in Telnet. The credentials used were similar to the other attacks but the credential list was slightly bigger than the previous attacks. After logging into the system, the attacker tries a variety of shell escape vulnerabilities to attempt to drop out of any potential restricted shells. This attack is the "hajime" attack. "Hajime" stands for beginning in Japanese. It is currently one of the fastest spreading malwares. But currently the malware has no attack module. It can only spread from one machine to another and hence currently is not very dangerous. While spreading, it is careful enough that it avoids some IP address ranges which include the US defence network.

### 5.2.2 Conclusions

The following conclusions were present from the attacks.

- Telnet being present in most of the IoT devices is easily the most attacked protocol. The fact that it is unencrypted also makes it easier to target.
- Attackers are wise to telnet emulators and honeypots. This is the main reason there are strings printed to check the validity of the BusyBox shell. Multiple methods are used to check the same.
- Mirai has a big influence on most of attacks on Telnet. Most of the attacks follow a specific method which most probably originated from original mirai code.
- Mirai is being overtaken by hajime. Hajime though recently new, is the cause of a large number of attacks. Although it currently just propagates now, if DDoS capability is added, it can be much more dangerous than mirai. It supposedly has a peer-to-peer network for command and control rather than a single C&C server.

## 5.3 HTTP attacks

The honeypot was deployed on 7 locations: Bangalore, Toronto, France, London, Singapore, Amsterdam and San Francisco. There were attacks from 306 different hosts on all the honeypots together. Saudi Arabia accounted for about 20 percent of the attacks.

### 5.3.1 Some major attacks

One attack that came was from host 67.68.235.183. The attacker tried to exploit a vulnerability in the apache struts Jakarta file upload multipart parser which allowed

| Country | Count |
|---|---|
| Saudi Arabia | 61 |
| United States | 50 |
| Netherlands | 23 |
| Germany | 21 |
| Ethiopia | 14 |

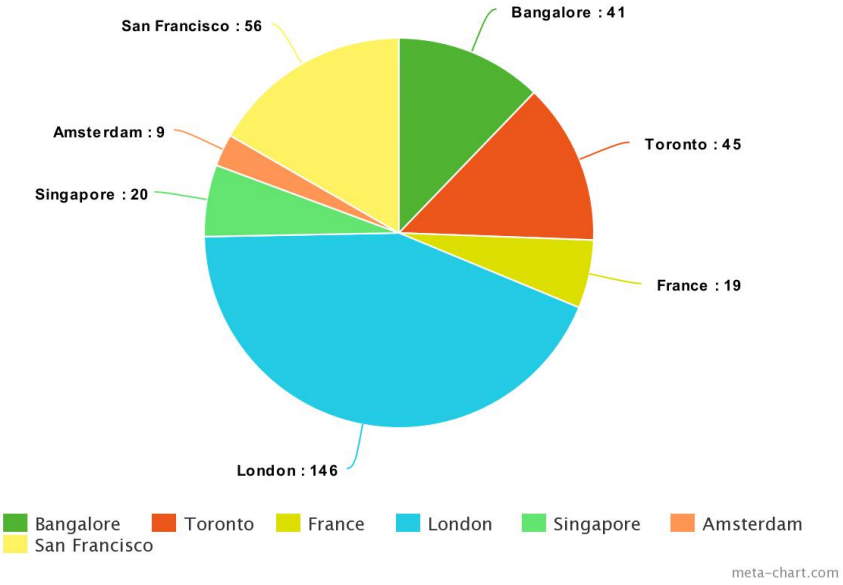
TABLE 5.1: Top 5 countries from which HTTP attacks originated

remote code execution in both Windows and Linux systems. The attack was executed by giving the commands to be executed in the content-type part of the HTTP request.

Another attack was the ZmEu attack. This is characterized by GET requests from user-agent "ZmEu". The attack came from the host 94.102.49.175. The attacker was trying to find vulnerabilities in applications like phpMyAdmin. Since none of the web applications the attacker was looking for was running, no further steps were taken.

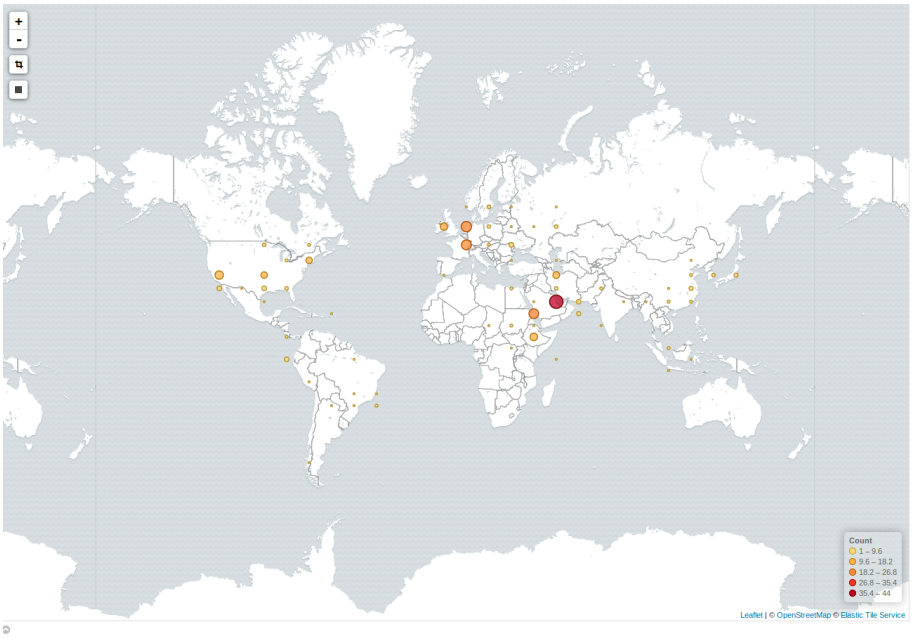
Only 3 attackers tried entering credentials in the HTTP login page. Only one of them entered the correct credentials ( username:"admin", password:"admin" ) but did not do anything of note. The other two attackers tried the same username but

FIGURE 5.3: Country-wise count of HTTP attacks



the password entered was "Ha2S+eOKqmzA6nrlmTeh7w==". Interestingly, the hash of this password is the same as the hash of the string "admin".

FIGURE 5.4: Geographical map of number of HTTP attacks




### 5.3.2 Conclusions

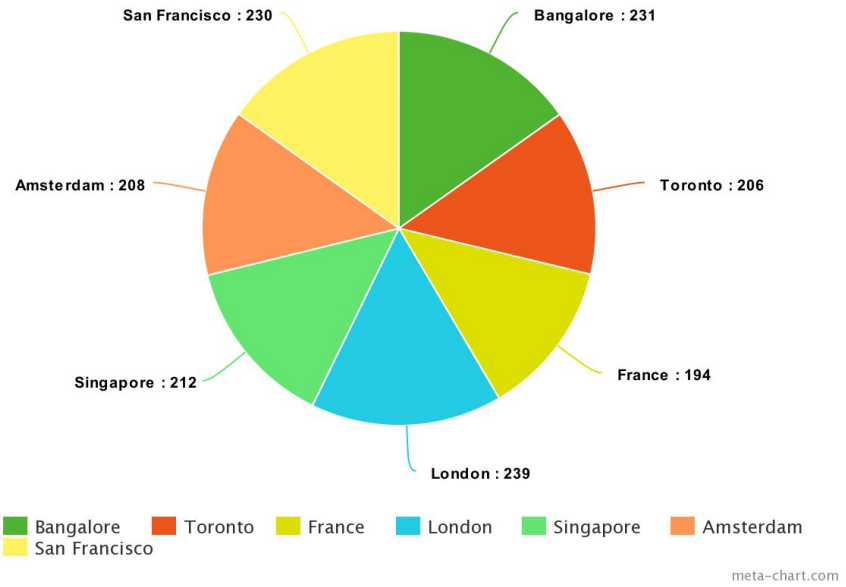
The following conclusions were present from the attacks.

- HTTP attacks are very rare compared to attacks on other protocols.
- Attackers are not attracted by pages which require login credentials. Unlike Telnet and SSH, attackers do not try brute forcing through the possible credentials.
- There are lot of web crawlers crawling the internet. This can be understood by the number of researchers trying to access the page.

## 5.4 CWMP attacks

The honeypot was deployed on 7 locations: Bangalore, Toronto, France, London, Singapore, Amsterdam and San Francisco. There were attacks from 1473 different hosts on all the honeypots together.

FIGURE 5.5: Country-wise count of CWMP attacks




| Country | Count |
|---|---|
| Iran | 291 |
| Australia | 167 |
| Republic of Korea | 123 |
| Russia | 106 |
| China | 98 |

TABLE 5.2: Top 5 countries from which CWMP attacks originated

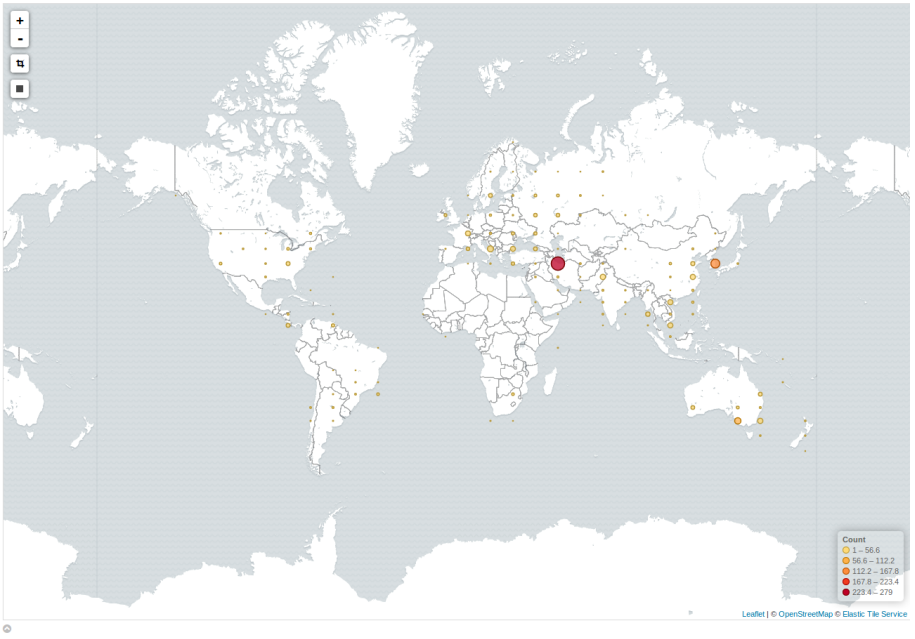
One attack(from IP address 115.69.6.196) tried to exploit a vulnerability in some systems that lets you execute BusyBox commands using CWMP. This command was used to try to download a binary and execute it.

### 5.4.1 Conclusions

- The fact that the number of CWMP attacks is so much more than HTTP attacks even though both are based on the same protocol shows that CWMP attacks are much more potent.
- Since CWMP port is generally open only on IoT devices, we can understand that IoTs are much more targeted than normal computers.

### 5.4.2 Some major attacks

FIGURE 5.6: Geographical map of number of CWMP attacks



# Chapter 6

# Future Work

The Telnet, HTTP and CWMP proxies are still not flexible enough like how HonSSH is. HonSSH is powerful enough to spoof passwords and support various kinds of input and output modules. So such capabilities can still be added to the code.

The back end can be changed to a more robust and flexible system or possibly a network. Currently the back end used is a single system for Telnet and SSH and another system of HTTP and CWMP. It would be better to give a network of machines at the back end so that once the attacker tries to access other machines while logged into the honeypot, he can be routed to a honeypot in the same network. Wireless capabilities can then be added to this network so that it can resemble a home or institute network.

# Appendix A

# Appendix A

Write your Appendix content here.

- CWMP proxy : https://github.com/krishnaprasad-p/proxies/blob/master/cwmp_proxy.py
- Telnet proxy : https://github.com/krishnaprasad-p/proxies/blob/master/telnet_proxy.py
- HTTP proxy : https://github.com/krishnaprasad-p/proxies/blob/master/http_proxy.py
- Proxy configuration : https://github.com/krishnaprasad-p/proxies/blob/master/proxy.conf
- SSH proxy : https://github.com/krishnaprasad-p/honssh
- HoneyThing docker file : https://github.com/krishnaprasad-p/honeything_docker
- HoneyThing docker image : https://hub.docker.com/r/krishnaprasad1990/honeything_image
- EasyCWMP docker file : https://github.com/krishnaprasad-p/easycwmp_docker
- EasyCWMP docker image : https://hub.docker.com/r/krishnaprasad1990/easycwmp_image
- OpenWrt docker image : https://hub.docker.com/r/krishnaprasad1990/openwrt_telnet_ssh

# Bibliography


[1] (2015). Welcome to dionaea's documentation! https://dionaea.readthedocs.io/en/latest/.

[2] (2016). Cowrie ssh/telnet honeypot. https://github.com/micheloosterhof/cowrie.

[3] Cymmetria (2016). Open source telnet honeypot. https://github.com/Cymmetria/MTPot.

[4] Dell'Aera, A. (2016). Thug. https://github.com/buffer/thug.

[5] desaster (2014). Kippo - ssh honeypot. https://github.com/desaster/kippo.

[6] easycwmp (2017). easycwmp. http://www.easycwmp.org/.

[7] elastic (2017a). Getting started with logstash. https://www.elastic.co/guide/en/logstash/current/getting-started-with-logstash.html.

[8] elastic (2017b). Kibana. https://www.elastic.co/products/kibana.

[9] Furche, J. and Elingehausen, R. (1999). Cybercop sting,getting started guide version 1.0.

[10] honeynet Project (2006). Know your enemy: Honeynets. http://old.honeynet.org/papers/honeynet.

[11] Lukas Rist, Sven Vetsch, M. K. M. M. (2010). Know your tools: Glastopf. http://honeynet.org/sites/default/files/files/KYT-Glastopf-Final_v1.pdf.

[12] Nicholson, T. (2016). Honssh. https://github.com/tnich/honssh/wiki.

[13] Peter, E. and Schiller, T. (2008). A practical guide to honeypots. http://www.cs.wustl.edu/~jain/cse571-09/ftp/honey.pdf.

[14] Phype (2016). Python telnet honeypot for catching botnet binaries. https://github.com/Phype/telnet-iot-honeypot.

[15] Piscitello, D. (2001). Honeypots: Sweet idea, sticky business. http://www.corecom.com/external/livesecurity/honeypots.html.

[16] Schneier, B. (1999). Honeypots and the honeynet project. https://www.schneier.com/crypto-gram/archives/2001/0615.html#1.

[17] twisted (2017a). t.c.t.statefultelnetprotocol(basic.linereceiver, telnetprotocol) : class documentation. http://twistedmatrix.com/documents/current/api/twisted.conch.telnet.StatefulTelnetProtocol.html.

[18] twisted (2017b). twisted.web.proxy.reverseproxyresource(resource) class documentation. https://twistedmatrix.com/documents/current/api/twisted.web.proxy.ReverseProxyResource.html.

[19] Wikipedia (2017a). Elasticsearch. https://en.wikipedia.org/wiki/Elasticsearch.

[20] Wikipedia (2017b). Hypertext transfer protocol. https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[21] Wikipedia (2017c). Internet of things. https://en.wikipedia.org/wiki/Internet_of_things.

[22] Wikipedia (2017d). Mirai (malware). https://en.wikipedia.org/wiki/Mirai_(malware).

[23] Wikipedia (2017e). Openwrt. https://en.wikipedia.org/wiki/OpenWrt.

[24] Wikipedia (2017f). Secure shell. https://en.wikipedia.org/wiki/Secure_Shell.

[25] Wikipedia (2017g). Telnet. https://en.wikipedia.org/wiki/Telnet.

[26] Wikipedia (2017h). Tr-069. https://en.wikipedia.org/wiki/TR-069.

[27] Wikipedia (2017i). Twisted(software). https://en.wikipedia.org/wiki/Twisted_(software).

[28] Yin Minn Pa Pa, Shogo Suzuki, K. Y. T. M. T. K. C. R. (2015). Iotpot: Analysing the rise of iot compromises. http://christian-rossow.de/publications/iotpot-woot2015.pdf.

[29] Ömer Erdem (2015). Honeything. https://github.com/omererdem/honeything.