

# **Adaptive Containerised Honeypots for Cyber-Incident Monitoring**

by

**Amber Higgins, B.A.I.**

A Dissertation submitted to the University of Dublin, Trinity College  
in fulfillment of the requirements for the degree of  
Integrated Masters in Computer Engineering

Supervised by Dr. Stefan Weber

May 2018

## **Declaration**

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

---

Amber Higgins

May 18, 2018

## Summary

This research focused on understanding evolving threats to the security of systems which are becoming increasingly sophisticated. Organisations in 2018 are in desperate need of *adaptive* threat detection capabilities to protect their systems, which are increasingly becoming more interconnected and consequently more exposed. The problem of interest is how effective threat intelligence mechanisms can be implemented to provide mitigations against such threats.

To identify the area of research and approaches that were ultimately taken, an in-depth exploration of the current state of the art in cybersecurity, intrusion detection and the hosting of modern systems infrastructures was undertaken. This research identified a wealth of approaches taken in the past to implementing solutions driven by active network defence. However, these solutions were largely found to focus on prolonging attack sessions rather than providing practical, feasible solutions for deploying active network defence in these environments.

The primary contribution of this research is the development of a containerised cyber-incident monitoring system. The novel component of this solution is a highly deployable pre-configured network of *honeypots*, devices that are capable of providing active threat detection in IT infrastructures. Attack event data captured by these devices makes it possible to provide enhanced threat intelligence through visualisation and alert generation, enabling rapid response and meaningful deductions to be made. Additionally, the

design of adaptive honeypots to improve threat detection was explored, with a focus on understanding the most enticing properties of victim systems.

This research has suffered from unanticipated setbacks which have limited the degree to which definitive conclusions can be drawn regarding enhanced adaptivity of honeypot design. However, the successful implementation of an end-to-end solution which provides concise descriptions of complex threat data, whilst also proposing a means of making active network defence feasible for organisations, is a valuable contribution to bringing active network defense into widespread use.

# Acknowledgments

This research has been a significant undertaking that I could not have accomplished without the support and guidance of so many different people.

First and foremost my thanks goes to Dr. Stefan Weber, who has been an incredible support through all the trials and tribulations of this project. His insights, patience and guidance have been second-to-none.

Special mentions go to Jason Flood of IBM Ireland, Michel Oosterhof of the Cowrie project, and Tony Winters of Optum Ireland who offered valuable insights and guidance on the way forward at various points during this research.

I am incredibly fortunate to have a wonderful support network: To Tushti, Isla and Éamon, thank you for listening to all of my ranting and raving and for being there to offer your advice, opinions and help with my issues (whether technical or motivational)!

Lastly and most importantly of all, to my loving family - Mom, Dad, Romy and Holly - and my incredible partner Breandán. You have invested so much in support of my dreams and aspirations, and I could never have accomplished any of this without you.

AMBER HIGGINS

*University of Dublin, Trinity College*

*May 2018*

All warfare is based on deception.

Hence, when we are able to attack, we must seem unable;

When using our forces, we must appear inactive;

When we are near, we must make the enemy believe we are far away;

When far away, we must make him believe we are near.

- Sun Tzu, *The Art of War*

# Abstract

The Internet is becoming an increasingly hostile environment, and though the deployment of security technologies is steadily improving over time, there is a huge and increasing gap between current technological threats and the measures in place to mitigate them.

This research has focused on providing enhanced security through incident-monitoring, devising a highly-deployable cyber-incident monitoring system to consolidate threat intelligence collected from a network of honeypots: An approach which promotes reactivity in the face of increased uncertainty about the nature of attacks, emphasising an active rather than passive approach to securing modern infrastructures which have seen an unprecedented growth in connectivity in critical services including health, transport and energy.

In particular, much attention in this research has focused on how such a system can feasibly provide active network defence for organisations in a way that is both practical and usable to operate and maintain. The use of containers and Platform-as-a-Service solutions in the deployment of security applications is an area where there is huge potential in this regard.

While research and industry projects have explored these uses of honeypots before including in the context of cyber-incident monitoring, this research distinguishes itself on the basis of providing a fully-networked system of honeypots packaged as a single deployable unit which can be hosted in Linux-based environments to provide active network defence in modern IT infrastructures.

Attention has also been given to the adaptation of honeypot design to more effectively entice attacks and hence provide improved threat detection, something for which limited conclusive research exists. The exponential increase in connectivity of systems which

were traditionally isolated motivates the targeting of such designs at Internet-of-Things botnets, automated attackers whose activities are a growing threat to critical service infrastructures.

# Contents

<b>Acknowledgments</b>	v
<b>Abstract</b>	vii
<b>List of Tables</b>	xvii
<b>List of Figures</b>	xviii
<b>Chapter 1 Introduction</b>	1
1.1 Problem Area . . . . .	2
1.2 Research Objectives . . . . .	3
1.3 Contributions . . . . .	3
1.4 Report Structure and Contents . . . . .	4
1.5 A Note on Sources . . . . .	5
<b>Chapter 2 State of the Art</b>	6
2.1 The Modern Cyber-Threat Landscape . . . . .	7
2.1.1 Cyber Attacks . . . . .	7
2.1.1.1 Overview . . . . .	8
2.1.1.2 Attackers . . . . .	9
2.1.1.3 Attack Vectors . . . . .	11
2.1.1.4 Anatomy of an Attack . . . . .	12
2.1.2 The Internet of Things . . . . .	14

2.1.2.1	Pervasive Technologies . . . . .	14
2.1.2.2	Security of IoT Devices . . . . .	15
2.1.2.3	Internet of Things Botnets . . . . .	18
2.1.3	Critical Service Infrastructures . . . . .	20
2.1.3.1	Vulnerable Critical Infrastructures . . . . .	21
2.1.3.2	Case Study: National Health Service Attacks 2017 . . . . .	21
2.2	Intrusion Detection . . . . .	23
2.2.1	Motivations for Intrusion Detection . . . . .	23
2.2.2	Firewalls . . . . .	24
2.2.2.1	Overview . . . . .	25
2.2.2.2	Design . . . . .	25
2.2.2.3	Challenges . . . . .	27
2.2.3	Honeypots . . . . .	28
2.2.3.1	Overview . . . . .	29
2.2.3.2	Design . . . . .	31
2.2.3.3	Comparison with Firewalls . . . . .	34
2.2.3.4	Honeynets . . . . .	35
2.2.3.5	Challenges . . . . .	36
2.2.4	Cyber-Incident Monitoring . . . . .	39
2.2.4.1	Usability in Security . . . . .	39
2.2.4.2	Visualisation in Security . . . . .	40
2.3	Modern System Infrastructures . . . . .	40
2.3.1	Infrastructure-as-a-Service . . . . .	41
2.3.1.1	Benefits . . . . .	41
2.3.1.2	Virtual Machines . . . . .	42
2.3.2	Platform-as-a-Service . . . . .	43
2.3.2.1	Containers . . . . .	43
2.3.2.2	Security through Containerisation . . . . .	45

2.4	Closely Related Work . . . . .	47
2.4.1	Adaptive Honeypots . . . . .	47
2.4.1.1	Kippo . . . . .	48
2.4.1.2	Cowrie . . . . .	49
2.4.1.3	Honeyd . . . . .	50
2.4.1.4	IoTPot . . . . .	50
2.4.1.5	IoTCandyJar . . . . .	52
2.4.1.6	Zigbee Honeypot . . . . .	54
2.4.2	Containerised Honeypots . . . . .	54
2.4.2.1	Distributed Virtual Honeynets . . . . .	55
2.4.2.2	Using Linux Containers for Deceptive Honeypots . . . . .	56
2.4.3	Honeypot-Driven Cyber-Incident Monitors . . . . .	56
2.4.3.1	TraCING . . . . .	57
2.4.3.2	TPot . . . . .	57
2.4.3.3	Modern Honey Network . . . . .	59
2.5	Summary . . . . .	60
<b>Chapter 3</b>	<b>Problem Formulation</b>	<b>61</b>
3.1	Identified Challenges . . . . .	61
3.1.1	Incident Monitoring in Critical Service Infrastructures . . . . .	62
3.1.2	The Need for Active Defence Mechanisms . . . . .	63
3.1.3	The Effectiveness of Honeypots in Enticing Attacks . . . . .	64
3.2	Proposed Work . . . . .	65
3.2.1	Objective 1: A Honeypot-Driven Cyber Incident Monitor for Critical Service Infrastructures . . . . .	65
3.2.1.1	Honeypots for Active Network Defence . . . . .	66
3.2.1.2	Containerised Honeypots . . . . .	66
3.2.1.3	Visualisation of Honeypot-Generated Data . . . . .	67

3.2.1.4	Notification System for Threat Events . . . . .	68
3.2.2	Objective 2: A Proposal for Adaptive Honeypots . . . . .	68
3.2.2.1	Measuring the Effectiveness of Honeypot Characteristics .	69
3.2.2.2	Targeted Honeypots for the Internet of Things . . . . .	69
<b>Chapter 4 Design</b>		<b>71</b>
4.1	Functional Considerations . . . . .	72
4.1.1	Deployment Platform . . . . .	72
4.1.1.1	Choice of Hosting Solution . . . . .	72
4.1.1.2	Use of Containers . . . . .	73
4.1.1.3	Host Operating System . . . . .	73
4.1.1.4	Choice of Honeypot . . . . .	74
4.1.1.5	Container Capabilities . . . . .	75
4.1.1.6	Honeynet Design . . . . .	75
4.1.1.7	Ease of Deployment . . . . .	77
4.1.2	Incident Monitoring . . . . .	77
4.1.2.1	Isolating the Monitoring System . . . . .	78
4.1.2.2	Secure Transfer of Honeypot Logs . . . . .	78
4.1.2.3	Visualisation of Honeypot Log Data . . . . .	79
4.1.2.4	Intrusion Alert/Notification Mechanism . . . . .	79
4.2	Challenges . . . . .	80
4.2.1	Fingerprinting Honeypot Environments . . . . .	80
4.2.2	Ethics of Honeypots . . . . .	81
4.2.3	Persisting Volatile Container Content . . . . .	82
4.2.4	Docker Security Considerations . . . . .	83
4.2.5	Enticing IoT Botnet Attacks . . . . .	85
4.2.6	Identifying Attacks from IoT Botnets . . . . .	86
4.2.7	Credible Evaluation of Honeypot Experiments . . . . .	87

4.3	Summary . . . . .	88
<b>Chapter 5</b>	<b>Implementation</b>	<b>90</b>
5.1	Understanding Containerised Honeypots . . . . .	91
5.1.1	Deploying an AWS EC2 Instance . . . . .	91
5.1.1.1	Provisioning an Instance . . . . .	91
5.1.1.2	Key Authentication . . . . .	92
5.1.1.3	Security Groups . . . . .	92
5.1.2	The Cowrie Honeypot: A First Look . . . . .	92
5.1.2.1	Installing Cowrie . . . . .	93
5.1.2.2	Configuring Cowrie . . . . .	94
5.1.2.3	Exposing the Honeypot to Attacks . . . . .	95
5.1.3	Building Docker Containers . . . . .	97
5.1.3.1	Docker Images . . . . .	97
5.1.3.2	Entrypoint Scripts . . . . .	98
5.1.3.3	Docker Volumes . . . . .	99
5.1.3.4	Docker Networking . . . . .	100
5.1.4	Conclusions . . . . .	101
5.2	Host Server Deployment . . . . .	102
5.2.1	The EC2 Honeypot Server Instance . . . . .	102
5.2.2	The EC2 Management Server Instance . . . . .	104
5.3	Building the Docker Honeynet . . . . .	104
5.3.1	The Cowrie Container . . . . .	105
5.3.1.1	Building an Enhanced Docker Image . . . . .	105
5.3.1.2	Additional Linux Utilities . . . . .	106
5.3.1.3	Sharing and Persisting Container Data . . . . .	108
5.3.1.4	Port Publication . . . . .	109
5.3.2	The Docker Honeynet . . . . .	109

5.3.2.1	Defining a Docker Bridge Network . . . . .	110
5.3.2.2	Connecting Containers to the Honeynet . . . . .	111
5.3.2.3	Network Configuration of the Honeywall Container . . . . .	112
5.3.3	Design Pivot: A New High-Interaction Honeypot . . . . .	113
5.3.3.1	Re-Evaluation of Initial Design . . . . .	114
5.3.3.2	Defining a New Container Image . . . . .	115
5.3.3.3	Configuring SSH and Telnet . . . . .	116
5.3.3.4	Device Banner Configuration . . . . .	117
5.3.3.5	Privileges and Authentication . . . . .	118
5.3.3.6	Logging Attack Events . . . . .	119
5.3.3.7	Package Utilities . . . . .	122
5.3.4	Finalising the Docker Honeynet . . . . .	122
5.3.4.1	The Docker <i>bridge</i> Network Configuration . . . . .	123
5.3.4.2	The Docker <i>dmz</i> Network Configuration . . . . .	125
5.4	Visualisation of Attack Data . . . . .	130
5.4.1	Management Server Instance . . . . .	131
5.4.1.1	Elasticsearch . . . . .	131
5.4.1.2	Kibana . . . . .	132
5.4.1.3	Nginx . . . . .	132
5.4.1.4	Logstash . . . . .	132
5.4.2	Honeypot Instance . . . . .	134
5.4.2.1	CRON . . . . .	134
5.4.2.2	Filebeat . . . . .	134
5.5	Alert/Notification System . . . . .	135
5.5.1	PSAD . . . . .	135
5.6	Summary . . . . .	136

<b>Chapter 6 Evaluation</b>	<b>138</b>
6.1 Design of Experiments . . . . .	138
6.1.1 Router Honeypot Container . . . . .	140
6.1.2 Cowrie Honeypot Containers . . . . .	142
6.2 Conducting Experiments . . . . .	143
6.2.1 Capturing Device Banners . . . . .	143
6.2.2 Experiment 1, Attempt 1 . . . . .	143
6.2.2.1 Configuration of the Router Honeypot . . . . .	144
6.2.2.2 Configuration of the Cowrie Honeypots . . . . .	144
6.2.2.3 Findings of the Experiment . . . . .	146
6.2.3 Experiment 1, Attempt 2 . . . . .	146
6.2.3.1 Reconfiguring the Router Credentials . . . . .	146
6.2.3.2 Findings of the Experiment . . . . .	147
6.2.4 Experiment 1, Attempt 3 . . . . .	148
6.2.4.1 Findings of the Experiment . . . . .	149
6.2.5 Concluding the Honeypot Experiments . . . . .	150
6.3 Discussion . . . . .	151
6.3.1 Honeypot-Driven Cyber Incident Monitoring . . . . .	151
6.3.1.1 Effectiveness of the Containerised Honeynet . . . . .	151
6.3.1.2 Architecture of the Monitoring System . . . . .	155
6.3.1.3 Impact of Visualisation on the Usability of Honeypots . . . . .	156
6.3.2 The Design of Adaptive Honeypots . . . . .	157
6.3.2.1 Unpredictability of Honeypot Experiments . . . . .	158
6.3.2.2 Ability to Capture Attack Data . . . . .	161
6.3.2.3 Hosting Honeypot Experiments on a Cloud Platform . . . . .	162
6.3.2.4 Fingerprinting Container Environments . . . . .	162
6.4 Summary . . . . .	163

<b>Chapter 7 Conclusions and Future Work</b>	<b>164</b>
7.1 Conclusions . . . . .	164
7.1.1 Cyber Incident Monitoring in Critical Infrastructures . . . . .	165
7.1.2 The Potential of Containers for Security Applications . . . . .	166
7.1.3 The Importance of Honeypots in Active Network Defence . . . . .	166
7.1.4 The Future of IoT Security . . . . .	166
7.2 Future Work . . . . .	167
7.2.1 Extension of the Cowrie Honeypot . . . . .	167
7.2.2 An Improved High-Interaction Containerised Honeypot . . . . .	168
7.2.3 Machine Learning Approaches to Adaptive Honeypot Design . . . . .	169
7.2.4 Alternative Threat Notification Systems . . . . .	170
7.2.5 Alternative System Architectures . . . . .	171
7.3 Closing Remarks . . . . .	172
<b>Appendix A Abbreviations</b>	<b>173</b>
<b>Bibliography</b>	<b>176</b>

# List of Tables

2.1	Comparison of Interaction Levels of Honeypots.	33
5.1	Routing Table of the EC2 Honeypot Instance	124
5.2	Routing Table of the Router Container	126
5.3	Routing Table of the Cowrie Container	128
6.1	Planned Iterations of the Honeypot Experiments	139

# List of Figures

2.1	Statista: Predicted Growth in Online IoT Devices by 2025 . . . . .	15
2.2	The Mirai Botnet: Most Common Default Passwords . . . . .	19
2.3	Illustration of WannaCry Infection in the NHS . . . . .	22
2.4	Use of Firewalls in a De-Militarised Zone . . . . .	25
2.5	Use of Honeypots in a De-Militarised Zone . . . . .	34
2.6	Lance Spitzner's Proposed Honeynet Architecture . . . . .	37
2.7	Components of a Virtualised Environment . . . . .	43
2.8	Components of a Container Environment . . . . .	44
2.9	The IoTPot Honeypot Architecture . . . . .	51
2.10	The IoT Candy Jar Honeypot Architecture . . . . .	53
2.11	Diagram of the TPot Architecture . . . . .	59
3.1	Illustration of a Black-Box System . . . . .	62
4.1	Overview of the Proposed System Design . . . . .	89
5.1	AWS EC2 Default Security Rules . . . . .	92
5.2	Cowrie Honeypot Default Passwords . . . . .	95
5.3	The Docker <i>build-and-run</i> Cycle . . . . .	99
5.4	A Conceptual Representation of Docker Volumes . . . . .	100
5.5	AWS EC2 Honeypot Server Instance Security Rules . . . . .	103
5.6	AWS EC2 Management Server Instance Security Rules . . . . .	105

5.7	Port Publication on the Honeywall Container . . . . .	110
5.8	The Router Container Honeypot Telnet Banner . . . . .	118
5.9	Docker Network Configuration of the EC2 Honeypot Instance . . . . .	123
5.10	Routing of Attack Traffic to the Router Container . . . . .	124
5.11	Console Output of <i>docker network inspect bridge</i> . . . . .	126
5.12	Attempted SSH Login to the Router Container . . . . .	127
5.13	Routing of Attack Traffic to a Cowrie Container . . . . .	128
5.14	Console Output of <i>docker network inspect dmz</i> . . . . .	129
5.15	A Denied SSH Login from the Router to a Cowrie Honeypot . . . . .	130
5.16	An Email Alert from PSAD . . . . .	136
6.1	DDoS-for-Hire: The Web Domain of a Human Attacker . . . . .	150
6.2	Cyber Incident Monitor on Kibana . . . . .	157
6.3	Visualisation: Attacks per Honeypot . . . . .	158
6.4	Visualisation: Most Popular Protocols per Honeypot by Month . . . . .	159
6.5	Fingerprinting of Docker Environments . . . . .	163

# Chapter 1

## Introduction

This research aims to demonstrate the potential of adaptive honeypot technology in emerging cybersecurity use-cases: In particular, in cyber-incident monitoring in critical service infrastructures which are increasingly becoming the targets of severe cyber attacks.

*Section 1.1* gives a high-level overview of the area in which this research is based, laying the foundations to highlight the importance of this research for effective cyber-incident monitoring systems.

*Section 1.2* outlines the objectives that were identified in this research, motivated by the challenges outlined in *Section 1.1*. These objectives include the development of a novel cyber incident monitoring system for effective intrusion detection, and a proposal for increased adaptivity of honeypots to enhance their threat detection capabilities.

*Section 1.3* gives a brief description of the contributions made by this research to the field of cyber security.

*Section 1.4* provides an overview of the structure and contents of the remainder of this document.

Finally, *Section 1.5* provides a brief note regarding the sources cited in this document.

## 1.1 Problem Area

There are more systems and devices connected to the internet now than ever before as society's reliance on interconnected services continues to grow. These huge numbers of devices are increasingly being operated by individuals with limited technical knowledge, and even less awareness of threats to their security. With the scale of poorly configured security measures in systems expanding, the modern internet is an ideal playground for hackers - endless devices to be compromised, exploited and used for nefarious purposes.

Critical service infrastructures are being heavily targeted in attacks, impacting upon thousands of people and having major consequences for the operation of organisations. There is a growing trend of these attacks being attributed to nation-states across the world: A recent example is from February 2018, where the Russian military were accused by multiple nations of being responsible for NotPetya, an attack that saw the entire Ukrainian economy grind to a halt in June 2017. [1] There is no doubt that the growing role of cyber attacks in global warfare and terrorism is cause for major concern, and the operators of critical infrastructures are facing increased uncertainty about the safety of their systems.

There is also a major shift in the hosting of these infrastructures, as organisations continue to migrate to outsourced infrastructure management solutions. The concept of decoupling services by deploying them as microservices has enhanced operational efficiency in organisations. How to translate workable security solutions into these deployments is still largely an unsolved problem however. With the rate of migration to these deployments growing steadily year-on-year, it is crucial that security catches up sooner rather than later.

## 1.2 Research Objectives

The research described in this document is motivated by the current state of interconnectivity, changing system infrastructures and global cyber warfare. In recognition of the gargantuan challenges being faced in an increasingly connected world, it is clear that there is a need to explore options to both deter and to more effectively defend against exploitation by malicious cyber-entities, whatever their motivations. Exploring technologies that *can* actively defend systems against the perpetrators of these activities is key to addressing these challenges.

This research explores an approach to providing practically feasible active defence for critical service infrastructures that are being targeted in ever-more sophisticated attacks, establishing that the use of containerisation and a centralised incident monitoring platform are ideal in achieving this objective. The novelty of this approach is primarily in the use of containers to host a honeynet, a fully-networked active defence system packaged as a single unit that can be automatically deployed within a matter of minutes.

Additionally, the design of adaptive honeypots<sup>1</sup> to improve threat detection in these systems is also explored, with a focus on understanding the most enticing properties of victim systems. Internet-of-Things botnets, a recent and evolving attack phenomenon, are targeted in the design of these honeypots as a topical and relevant use-case.

With the evolution of new threats to the security of data and systems every day as technology continues to advance, proposing a novel and effective system to address the stated objectives is the central aim of this research.

## 1.3 Contributions

Though this research has been limited in its scope due to the time constraints associated with an 8-month Masters programme, it produced a containerised honeynet-driven inci-

---

<sup>1</sup>Adaptivity in this context should be interpreted as the ability of a honeypot to assist the attacker in believing that they have achieved their goals.

dent monitor: An end-to-end system that can be efficiently employed as an active network defence measure to enable rapid response to security incidents. As an area which is still catching up with transformations in modern systems architectures, this proof-of-concept design demonstrates the feasibility of containerising security solutions to deploy at scale.

## 1.4 Report Structure and Contents

The remaining sections of this document are structured as follows.

*Chapter 2* is entitled *State of the Art*. It provides an in-depth look at the state of the art in cybersecurity in 2018, giving particular attention to increased connectivity and the implications of this for critical service infrastructures. Intrusion detection techniques and their role in cyber-incident monitoring is explored, comparing and contrasting passive and active defence intrusion detection mechanisms. Modern infrastructure hosting solutions in widespread use by organisations today are discussed, with an analysis of what these can contribute to modern security applications. Finally, a number of projects closely related to the work carried out in this project are evaluated.

*Chapter 3* is entitled *Problem Formulation*. It explores the formulation of the research objectives in the context of the major challenges identified in *Chapter 2*, addressing them in the establishment of the proposed work for this research.

*Chapter 4* is entitled *Design*. It describes the considerations and challenges involved in the initial design phases of the project, as well as the decisions that resulted from this process.

*Chapter 5* is entitled *Implementation*. It details the implementation phase of the project, describing the steps involved in developing the various components of the system. A number of pivotal design decisions as the project evolved are also discussed before arriving at the final solution.

*Chapter 6* is entitled *Evaluation*. It provides an evaluation of the implemented system as well as detailing the experiments that were carried out and the challenges faced in

doing so.

Finally, *Chapter 7* is entitled *Conclusions and Future Work*. It gives some overall conclusions regarding the final solution, before exploring a some potential avenues for the continuation and enhancement of the work done in this project. It concludes with some final remarks and recommendations based on the research conducted.

## 1.5 A Note on Sources

Research in the area of cyber security is interesting in that it relies heavily on current, up-to-date awareness of unfolding events. This often means that the sole reference resources from which information can be obtained about these events are not from academia, since academic research almost always lags behind the most current trends documented by the media, security companies and government bodies.

Many of the sources cited in this document are non-academic, and hence often not peer-reviewed by independent bodies. The greatest of care has been taken to select maximally reputable, objective and independent sources as reference material for the work. However sources cited which do not originate from peer-reviewed publications should be carefully considered by the reader based on the potential for bias of any parties involved in the writing.

All visual figures were generated as part of the writing of this document unless stated otherwise, in which case the original source is cited.

# Chapter 2

## State of the Art

This section is intended as an introduction to the state of the art in cybersecurity and cyber-threats as they exist in 2018, with an aim of giving the reader in-depth context on the area in which the research fits.

*Section 2.1* outlines the current state of cybersecurity in the world as it stands in 2018, providing an illustration of the range of attacks and attackers threatening modern systems. The rapid introduction of connectivity into systems as a result of the Internet-of-Things phenomenon is explored, with reference to the evolution of threats that exploit this increased connectivity.

*Section 2.2* considers the role of effective intrusion detection in securing systems and devices, emphasising the need to move away from reliance on passive threat detection mechanisms that are widely deployed in systems. The role of honeypot technology in providing active network defence is discussed in particular, with a detailed analysis of the important characteristics of their design in order to employ them effectively as well as the challenges that they face in widespread deployment. Lastly, the strengths of cyber incident monitoring platforms with regards to enhanced usability and communication capabilities are highlighted.

*Section 2.3* explores the current state of the art in deployment platforms that are widely in use in production systems, considering both Infrastructure-as-a-Service (IaaS)

and Platform-as-a-Service (PaaS). Both the opportunities and challenges to implementing security in evolving systems architectures are addressed here.

*Section 2.4* describes a number of closely related projects encountered during this research, highlighting their interesting attributes and contributions to the area of cybersecurity. These projects focus variously on the design of adaptive honeypots, the use of containerisation in facilitating their easy deployment, and the benefits that can be gained from visualisation of threat data.

Finally, *Section 2.5* summarises the knowledge gained from the state of the art as explored in this chapter, motivating the decision to tackle the challenges that have surfaced in this chapter.

## 2.1 The Modern Cyber-Threat Landscape

The cyber-threat landscape is continuing to shift and transform at pace in 2018. Cyber incidents are not localised but instead are happening on a global scale, their reach extending across all physical borders and legal jurisdictions and affecting a diverse set of victims. The growth in diversity and severity of attacks has resulted in the formation of new security taskforces<sup>1</sup> by nations across the world in order to identify how to protect their society from growing cyber threats.

### 2.1.1 Cyber Attacks

Cyber attacks continue to grow in prevalence and diversity with the continuous conception and deployment of new technologies. The success of these attacks is often less to do with

---

<sup>1</sup>For instance, in February 2018 US Attorney General Jeff Sessions announced the formation of a new Cyber-Digital Task Force as part of the US Department of Justice. In the Memorandum published by the department, they state that "many of the most pressing cyber threats that our nation faces transcend easy categorization. These threats include: efforts to interfere with, or disable, our critical infrastructure; efforts to interfere with our elections; use of the Internet to spread violent ideologies and to recruit followers; theft of corporate, governmental, and private information on a mass scale; use of technology to avoid or frustrate law enforcement, or to mask criminal activity; and the mass exploitation of computers, along with the weaponizing of everyday consumer devices ... to launch attacks on American citizens and businesses." [2]

the skill of those behind the attacks, and more with the lack of awareness and care taken by their victim. Many successful cyber attacks have not been particularly sophisticated, often resulting from preventable, known vulnerabilities that had not been addressed.

#### **2.1.1.1 Overview**

Attacks can be broadly classified as being either intentional or unintentional.

##### **1. Intentional**

Cyber attacks can be classified as intentional if the victim has been targeted by the attacker. The most common motivations for intentional cyber attacks are as follows:

- **Cybercrime**

Cybercrime is most often motivated by financial gain, and manifests in a huge variety attacks including social engineering, ransomware<sup>2</sup> and identity theft.

- **Activism and Terrorism**

Cyber-activism, or "hacktivism" as it is also known, involves launching cyber attacks in order to raise awareness of a cause: Cyber-terrorism often fits the same description. These attacks generally focus on spreading propaganda, website defacement, Distributed-Denial-of-Service (DDoS), and data leaks.

- **Nation State Attacks and Cyberwar**

Cyber war refers to attacks by nation states as part of their political agenda. There is increased activity across the world from attackers acting on behalf of nation states with the intention of achieving military and political impact. As stated in a whitepaper published by Cisco Corporation in 2017 [4], "states have built their own cyber attack teams. These teams are no less important to

---

<sup>2</sup>Ransomware is a category of malicious software (malware) which, when present on a system, will render some part of the system inoperable until a ransom is paid to release it. In the case of the WannaCry ransomware, all files on the infected system were encrypted and the attackers demanded a payment of bitcoins to a particular bitcoin wallet in order to release it. [3]

their state strategies than their army or navy... Whether you know it or not, you are in a cyber war”.

## 2. Unintentional

Cyber attacks can be classified as unintentional if the victim was not specifically targeted, but happened to be a victim of an attack as a matter of circumstance. The most common motivation for unintentional cyber attack is social engineering through schemes such as email phishing scams.

The phases involved in the progression of a cyber attack depend on a variety of factors including the motivation for the attack, the type of attacker and the nature of the attack.

### 2.1.1.2 Attackers

Attackers in the context of this work are those who *intentionally* attack IT systems and infrastructures. Various terms including "bad actors" and "black hats" are widely used in the security industry to refer to these individuals.

In the world of today, it is very easy for anyone who feels inclined to attack IT systems to learn how to do so. There are endless resources, toolkits and communities available across the Internet to enable an individual to conduct successful exploits on any scale.

Attackers, as individuals performing malicious activity, will generally strive to conceal their real identity. There exists a variety of distinct attacker classifications, outlined below.

- **Sophisticated Attackers**

These are attackers with high-level technical skills, capable of discovering new categories of vulnerabilities and writing powerful attack toolkits. In general they are the most difficult to defend against because of their substantial knowledge and skills, and are widely recruited by nation states to conduct espionage or sabotage activities.

- **Script-Kiddies**

So-called “script-kiddies” launch attacks using malware and toolkits created by others, and are not very sophisticated. In general they are motivated by peer-group competition and reputation rather than financial gain or political causes.

- **Hacktivists**

These attackers carry out *hacktivism attacks*. They are motivated by a cause that they aim to promote and publicise, and can include terrorists and government-funded organisations as well as vigilante groups. They often seek to damage another party. A well-known active hacktivist group that often make headlines are the *Anonymous* hacktivist group<sup>3</sup>.

- **Insiders**

Insiders are one of the greatest threats to the security of organisations, since they are trusted individuals inside the organisation’s network. Similarly to cyber attacks, there are two categories of insider attack: Intentional, and unintentional.

1. *Intentional*

These individuals often have sophisticated technical knowledge and a motivation to damage the organisation, such as being paid by a competitor or nursing a personal vendetta, allowing them to launch an attack on the systems from the inside. Rigorous background checks during the hiring process as well as incentivising loyalty within the organisation can help to mitigate this threat.

2. *Unintentional*

Unintentional attacks mostly occur due to social engineering schemes such as phishing email scams. Promoting the security vigilance of members of an organisation through education and training is crucial in preventing these attacks.

---

<sup>3</sup>A recent example of a successful attack by members of the Anonymous group was to take down the website of Spain’s Constitutional Court in support of the “Free Catalonia” movement in October 2017. [5]

- **Bots**

Bots are attackers which conduct attacks by executing malware. A group of devices controlled by such malware is called a *botnet*. The malware used to infect these devices is propagated from device to device, seeking to infect as many devices as possible. When compromised, the computational resources of the devices are harnessed, allowing the so-called *botmaster* - the individual controlling the botnet - to coordinate the devices to perform a specific function as a collective group.

#### 2.1.1.3 Attack Vectors

There are countless different attack vectors that exist, a number of categories of which are outlined below.

- **Social Engineering**

These attacks include deception tactics like email phishing, which involves obtaining the trust of privileged insiders by posing as a trustworthy entity in a communication. An emerging social engineering tactic is the use of social media botnets to influence the opinions and actions of unsuspecting users: An excellent example of this is the so-called *Brexit Botnet*, where large numbers of Twitter accounts were observed posting and re-tweeting tweets promoting the 'Vote Leave' campaign, before being removed shortly after the referendum results were released. [6]

- **Vulnerability Exploits**

This involves intentionally exercising a vulnerability in software or firmware. Every system has vulnerabilities: It is up to those who develop them to develop mitigations against them once they are discovered, and those who use them to make sure that they keep up-to-date with patches as they are released. Typical examples of severe exploit vulnerabilities include the OpenSSL Heartbleed vulnerability<sup>4</sup>.

---

<sup>4</sup>This vulnerability allowed attackers to bypass encryption protections by sending carefully crafted requests in order to obtain a chunk of private memory in response. [7]

- **Authentication Attacks**

Authentication attacks generally make use of either stealing credentials or brute-forcing authentication<sup>5</sup>.

- **Network Attacks**

These attacks can be either passive or active. A passive network attack could involve eavesdropping on network traffic, whereas an active network attack could be a DDoS attack using high volumes of network packets to cause service outages.

#### **2.1.1.4 Anatomy of an Attack**

There are 5 primary steps involved in the steps typically involved in the formulation of an attack on a system: Reconnaissance, enumeration, penetration, exfiltration and sanitisation. [4]

##### **1. Reconnaissance**

Reconnaissance involves investigating the target, generally by gathering information that is available in the public domain and not interacting directly with the systems associated with the target. This kind of information gathering can include investigating elements of the systems such as the technologies in use, enabling the attacker to hone in on the targets of choice in the system.

##### **2. Enumeration**

In this phase, the attacker will utilise information gathered in the *reconnaissance* phase to begin probing the target systems using various techniques and toolkits<sup>6</sup>, i.e. direct interaction with the target systems in order to *fingerprint*<sup>7</sup> them. As well as

---

<sup>5</sup>Brute-force authentication works on the basis of weak authentication mechanisms being implemented to allow access to the target system. This includes design flaws such as weak username-password credentials being used, or allowing an unlimited number of login attempts to the system. Bots often use this strategy in attacks by trying as many combinations of username and password as possible.

<sup>6</sup>One such toolkit is *Nmap*, a powerful port scanner and device fingerprinting tool.

<sup>7</sup>Fingerprinting in relation to security refers to the identification of a system based on tell-tale characteristics: For example, based on a response obtained for a request sent over a particular protocol, it may be possible to identify what version of that protocol is supported by the recipient.

this, measures are often taken to trick those with knowledge of the system internals into giving away valuable information through tactics such as spear-phishing<sup>8</sup>.

### 3. Penetration

This stage involves the exploitation of any vulnerabilities identified by the attacker in the *reconnaissance* and *enumeration* phases, with the intention of gaining access to the target system. There is a wide variety of ways in which a vulnerability can be exploited: For example, an interface to a database management system may not have been fuzzed<sup>9</sup> sufficiently, allowing a SQL injection<sup>10</sup> attack to occur.

Once the attacker gains access, they will attempt to maintain a foothold in the system through measures such as privilege escalation and exploitation of further connected systems, with the intention of maintaining "Command and Control" (C+C). In an automated attack, this allows the compromised device to communicate with external C+C systems to receive commands and subsequently act on them.

### 4. Exfiltration

After compromising the target system, an attacker will try to maintain access to the system such that they will not lose control of it in the future. They will often install backdoor mechanisms<sup>11</sup> or extract data from the system to ensure this, which they will do this as stealthily as possible in order to avoid being detected.

### 5. Sanitisation

As explained previously *Section 2.1.1.2*, attackers will almost always strive to remain anonymous: It is in their best interests to remain undetected after compromising

---

<sup>8</sup>Spear-phishing is similar to regular email phishing, except that malicious emails are sent from a known or trusted sender. This deception tactic makes spear-phishing very effective in inducing targets to reveal confidential information.

<sup>9</sup>Fuzzing is a process used for input testing of systems, where many combinations of invalid, unexpected or random data are provided as inputs and the system then observed for failures.

<sup>10</sup>SQL injection is an attack where a web interface accept malicious SQL inputs that lead to the attacker having control of the underlying database.

<sup>11</sup>A backdoor mechanism is a means of accessing the system in question which bypasses any security mechanisms in place.

a system so that their exploitation of the system is not discovered and removed. Before disconnecting from the compromised system, the attacker will attempt to remove all evidence of their presence on the system to ensure this.

### **2.1.2 The Internet of Things**

The Internet-of-Things (IoT) is a concept with many dimensions as it encompasses an enormous range of technologies, standards and services. In general, an IoT system is understood to be a collection of smart devices working on a collaborative basis to fulfil a common goal. [8]

The IoT has taken the world by storm in the past several years, bridging the gap between the physical world and the Internet. Connectivity has been introduced rapidly into critical systems and processes: Industries and sectors including agriculture, energy, health and transport have all seen revolutionary transformations as part of what has been dubbed 'Industry 4.0'.

#### **2.1.2.1 Pervasive Technologies**

According to statistics published by Statista<sup>12</sup> in 2018, [9] by 2025 it is predicted that there will be over 75 billion IoT devices online globally: Several times greater than the human population of the world<sup>13</sup>. Figure 2.1 shows their predicted exponential growth trend over the period from 2015. These devices include everything from environmental sensors to wearable technologies and process automation components, and are playing an increasingly important role in the lives of all members of society. For a single IoT system, sensors could be used to collect information about a particular environment in a Wireless Sensor Network (WSN) configuration whilst communicating the information to a remote application, where it is aggregated and pushed to a set of distributed smartphones.

---

<sup>12</sup>Statista cite the original source of this data as being from IHS Markit, an analytics firm headquartered in London.

<sup>13</sup>The latest population statistics published by the Department of Economic and Social Affairs Population Division of the United Nations in 2017 [10] estimate the population of the world at approximately 7.6 billion individuals.

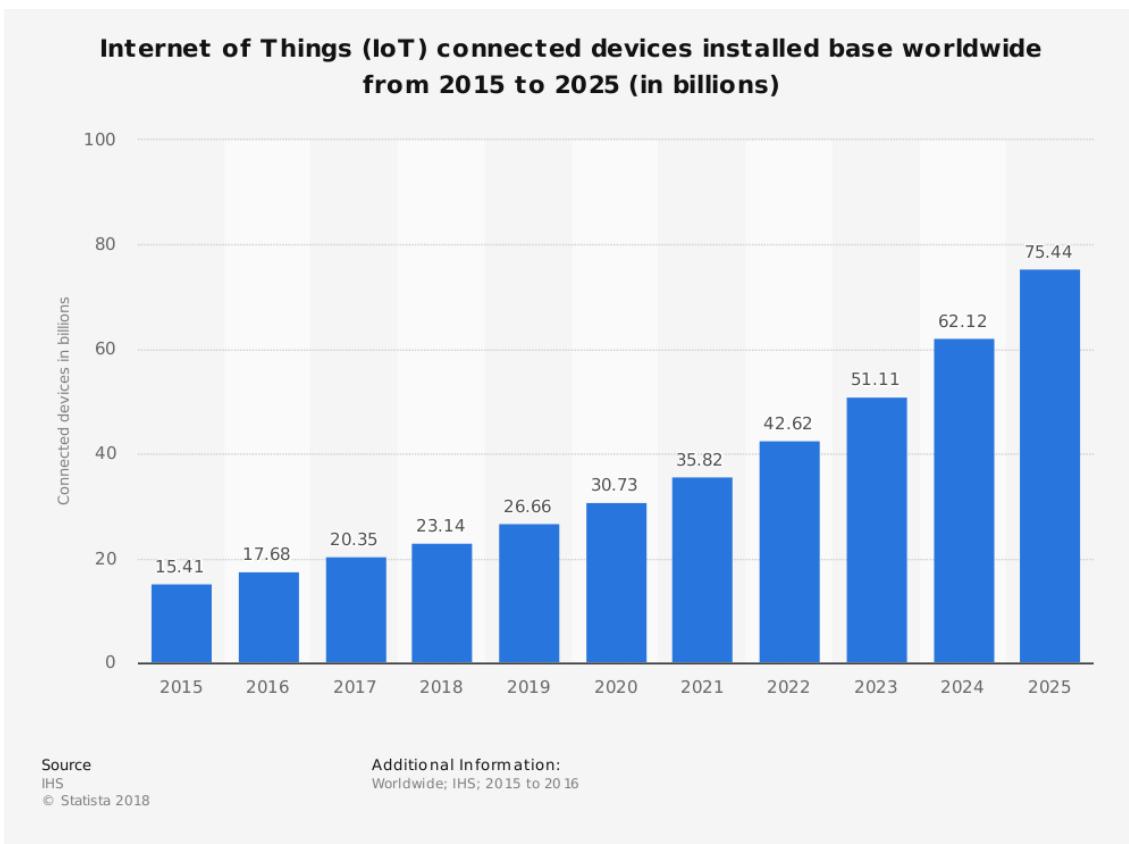


Figure 2.1: Statista: Predicted Growth in Online IoT Devices by 2025

This graph is provided based on research by Statista GmbH, showing their predicted growth trend for the number of internet-connected IoT devices by 2025. Historical data is included back to 2015.

### 2.1.2.2 Security of IoT Devices

Whilst connectivity between devices and processes has enhanced and optimised industrial processes, enabling the conception of countless new applications, the pace of this transformation has resulted in little to no attention being given to the security of these systems. Apart from concerns about the extent of pervasive monitoring conducted by IoT systems<sup>14</sup>, new applications are being found for IoT devices at such pace that instead of being built into these applications from the beginning, security is largely being left behind to catch up on later. Most of the time, it can be taken that there is no such thing as

<sup>14</sup>The extent to which connected devices have become part of everyday life concerns many individuals, particularly since the primary purpose of IoT devices is to continuously gather, analyse and transport data to other systems. In an eye-opening paper published in 2017, [11] Lackorzynski *et al.* emphasise the dangers of internet-connected devices such as the *Hello Barbie* childrens toy, which is "a doll that could talk back to the user... in this case very probably a minor".

“security by default”.

The risks - both to security and privacy - associated with the pervasiveness of the Internet in everyday life today are something that the general population are largely oblivious to. Insecam, which is a website broadcasting online feeds from thousands of IP cameras with default login credentials without the owners’ knowledge, illustrates the gravity of this issue. [12] Devices and systems which never had communication capabilities before are being brought online, exposing them to significant potential tampering and misuse in ways that could affect the welfare of entire populations. Crucial systems on which society relies on a daily basis such as power grids, medical devices and traffic control systems are increasingly falling into this category, becoming exposed to attacks in a way that they never were in the past.

Some of the greatest issues with the security of these devices include the following:

### **1. Weak Default Credentials**

The main contributor to the success of most attacks on IoT devices to date is the fact that these devices are shipped with weak default credentials that are identical across all instances of a given model. Additionally, these credentials are usually the only security measure used to mitigate against threats to these devices: There is very little effort required for an attacker to successfully compromise such a device.

### **2. Limitations in Applying Updates**

The environments in which many IoT devices are deployed make it very difficult to patch vulnerabilities and apply updates to these devices. In general, large numbers of heterogeneous devices are deployed in a geographically dispersed, distributed setup and communicate over a variety of different networking protocols. Applying updates in such a deployment is inherently challenging.

### **3. Limited Capacity for Encryption**

Due to the limited computational capacity and bandwidth of many low-powered

IoT devices, the deployment of encryption on these devices is near to impossible. The compute-intensive operations required by most encryption protocols to both encrypt and decrypt data-in-transit and data-at-rest make it very difficult to implement encryption practically. To illustrate this, the Internet Engineering Task Force (IETF) stated in their most recent specification for Low-Powered Wide-Area Networks (LPWANs) that implementing key authentication mechanisms becomes "challenging to handle in LPWANs with bounded bandwidth." [13] As discussed by Dowling *et al.*, if encryption is not implemented correctly, this can open the door for attacks on confidentiality, origin authentication and data integrity if the device in question is compromised. [14]

#### 4. Insecure Communication Protocols

Many IoT devices use insecure communication protocols because of their requirement for low power consumption. The *telnet* network protocol, which runs on port 23/TCP, has been largely retired as it is highly insecure, transmitting all traffic in clear-text including login credentials<sup>15</sup>. It is however a lightweight protocol to deploy on a low-powered device, meaning that it has been widely included in IoT devices for authentication. This further weakens the security of these devices: A passive attacker eavesdropping on network traffic can capture IoT device data for later analysis or compromise the device itself.

For attackers, IoT devices are very enticing: They are always online, using weak security mechanisms and often with access to powerful shells. The emergence of freely-available online scanning tools such as Shodan<sup>16</sup> mean that identifying these devices becomes trivial. As affirmed by leading cybersecurity expert Bruce Schneier in his article *The Internet of Things is Wildly Insecure - And Often Unpatchable* in 2014, "if we don't

---

<sup>15</sup>The full telnet protocol specification was finalised by the IETF 35 years ago in 1983, and explains the motivations behind the protocol and its intended uses. [15]

<sup>16</sup>Shodan is a search engine for internet-connected devices. If a search is performed for "default password", devices using default passwords are identified all over the world and returned to the user as a result. [16]

solve this soon, we're in for a security disaster as hackers figure out that it's easier to hack routers than computers". [17]

### 2.1.2.3 Internet of Things Botnets

Given the exponential growth in the number of internet-connected devices over the past several years, and given that attacks will always evolve as new technologies emerge, new attack vectors and threats have been realised in relation to IoT devices since Bruce Schneier's insightful and telling article in 2014. [17] Though many IoT devices are low-power devices commonly deployed in remote areas with low network bandwidth, large numbers of these devices can have a combined impact that packs a significant punch: Enter IoT botnets.

IoT botnets are likely the most active and topical breed of botnet at present. They are botnets composed entirely of compromised IoT devices. Most of these devices have been found to run the Secure SHell<sup>17</sup> (SSH) and telnet authentication protocols with simple default username-password combinations, making it straight-forward to launch brute-force authentication attacks to gain root access to the device.

The highest profile attack to-date by an IoT botnet took the world by surprise in late 2016. The Mirai botnet, which amassed an army of over 300,000 IoT devices by capitalising on their poorly-implemented authentication mechanisms, conducted one of the biggest ever DDoS attacks by volume. The attacks began by targeting *Krebs on Security*, [19] the website of well-known investigative security journalist Brian Krebs, eventually peaking at almost 1Tbps with an attack on French cloud hosting provider OVH. Research conducted by Antonakakis *et al.* [20] on the success of Mirai found that for the 3 most common architectures found to be running the telnet protocol<sup>18</sup>, "security cameras, DVRs, and consumer routers represent(ed) the majority" of these devices.

The scale and impact of this botnet was a revolutionary development. The fact that

---

<sup>17</sup>The SSH protocol runs on port 22/TCP, and was designed by the IETF for secure remote login using encryption. [18]

<sup>18</sup>MIPS 32-bit, ARM 32-bit, and x86 32-bit were the three most common CPU architectures identified in the collected device samples, accounting for circa 74% of these. [20]

Password	Device Type	Password	Device Type	Password	Device Type
123456	ACTi IP Camera	klv1234	HiSilicon IP Camera	1111	Xerox Printer
anko	ANKO Products DVR	jvbzd	HiSilicon IP Camera	Zte521	ZTE Router
pass	Axis IP Camera	admin	IPX-DDK Network Camera	1234	Unknown
888888	Dahua DVR	system	IQinVision Cameras	12345	Unknown
666666	Dahua DVR	meinsm	Mobotix Network Camera	admin1234	Unknown
vizxv	Dahua IP Camera	54321	Packet8 VOIP Phone	default	Unknown
7ujMko0vizxv	Dahua IP Camera	00000000	Panasonic Printer	fucker	Unknown
7ujMko0admin	Dahua IP Camera	realtek	RealTek Routers	guest	Unknown
666666	Dahua IP Camera	111111	Samsung IP Camera	password	Unknown
dreambox	Dreambox TV Receiver	xmhdipec	Shenzhen Anran Camera	root	Unknown
juantech	Guangzhou Juan Optical	smeadmin	SMC Routers	service	Unknown
xc3511	H.264 Chinese DVR	ikwb	Toshiba Network Camera	support	Unknown
OxhlwSG8	HiSilicon IP Camera	ubnt	Ubiquiti AirOS Router	tech	Unknown
cat1029	HiSilicon IP Camera	supervisor	VideoIQ	user	Unknown
hi3518	HiSilicon IP Camera	<none>	Vivotek IP Camera	zlxz.	Unknown
klv123	HiSilicon IP Camera				

Table 5: **Default Passwords**—The 09/30/2016 Mirai source release included 46 unique passwords, some of which were traceable to a device vendor and device type. Mirai primarily targeted IP cameras, DVRs, and consumer routers.

Figure 2.2: The Mirai Botnet: Most Common Default Passwords

A table provided in the *Understanding the Mirai Botnet* paper, [20] showing the passwords that were most commonly found across all of the devices analysed in their research.

it was capable of threatening some of the worlds best-defended targets using enormous numbers of devices with low computational capabilities speaks volumes about the scale of issues that are currently being faced in IoT device security. As stated by Bruce Schneier in his 2017 article *Botnet of Things*, "botnets will get larger and more powerful simply because the number of vulnerable devices will go up by orders of magnitude over the next few years ... overall, the trends favor the attacker". [21]

A number of other IoT botnets have surfaced since the Mirai attack, many of them variants of the original Mirai source code which was published online by a user called *Anna Senpai* on the popular *HackForums* site. [22] One noteworthy botnet is BrickerBot, an IoT botnet which surfaced shortly after the initial Mirai attacks on *KrebsOnSecurity*. [23] The author of this botnet, who calls themselves the *janitor*, claims to be a *white hat*<sup>19</sup> hacktivist intent on performing what they termed "Internet Chemotherapy": The complete destruction of IoT devices with poorly-implemented security measures as

<sup>19</sup>This is a term commonly given to individuals who try to identify security vulnerabilities through hacking. They are different to malicious attackers in that they respect any relevant laws, and do not hack in order to damage the target system. Clearly, the *janitor* doesn't fit this description.

a punishment to their owners. In a farewell email made available to security site *BleepingComputer* on 10th December 2017, the *janitor* warns that the world should "WAKE UP TO THE FACT THAT THE INTERNET IS ONLY ONE OR TWO SERIOUS IOT EXPLOITS AWAY FROM BEING SEVERELY DISRUPTED", and that organisations and individuals must take action to prevent this. [24]

### 2.1.3 Critical Service Infrastructures

At a time where heightened political tensions exist between many of the world's most powerful nation states<sup>20</sup>, cyber attacks are playing a prominent role in warfare and terrorism. Cyber attacks on critical service infrastructures have the potential to cause responses with severe impact. Illustrating the severity of threats from competing nation states, Unal *et al.* [26] discuss the growing reliance of nuclear weaponry on digital technology and communication systems, stating that "the likelihood of attempted cyber attacks on nuclear weapons systems is ... increasing from advanced persistent threats from states and non-state groups".

In a report published in 2017 containing recommendations to the U.S. administration on national cyber-security, [27] M.I.T. researcher Joel Brenner states that one of the eight major challenges facing the United States government currently is to "enable critical infrastructure operators to quickly identify and respond to cyber risk arising from cross-sector linkages as well as from their own network". This statement encapsulates precisely the issues facing critical service providers worldwide today, and is something that is being reflected in incidents across the world with an increasing number of attacks on high-profile targets.

---

<sup>20</sup>The United States are currently locked in a cyber-combat against "HIDDEN COBRA", a codename for what they claim is North Korea's DDoS botnet infrastructure, as part of ongoing disputes between the two nations. A US national alert [25] issued in June 2017 by the US-CERT claimed that HIDDEN COBRA had "likely targeted the aerospace, telecommunications, and finance industries" in the United States.

### **2.1.3.1 Vulnerable Critical Infrastructures**

It is clearer now than ever that key organisations and critical services require a renewed focus on defence strategies for their IT infrastructures. One such category of critical infrastructures is Supervisory Control and Data Acquisition (SCADA) systems, which are employed in industrial control environments to perform functions such as process control and monitoring. Typically, these systems deal with lots of components including sensors and mechanical parts like motors, enhancing the efficiency of operations in these environments.

Securing these systems is vitally important: They are the underlying control system of almost all nation-critical infrastructures such as transport, power and water. It is clear that the consequences of compromising these systems have the potential to be serious and far-reaching.

### **2.1.3.2 Case Study: National Health Service Attacks 2017**

In May 2017, one of the more memorable attacks in recent years on a nation-critical service occurred in the United Kingdom. The global WannaCry *ransomware* attacks, which successfully compromised more than 200,000 systems in over 100 countries, knocked systems in the UK National Health Service (NHS) at 37 sites offline for over a week with more than 6,912 appointments cancelled in that time. Though the NHS was almost certainly not a specific target of the ransomware authors<sup>21</sup>, the attacks had a huge impact on health services across the UK. [29]

Examining the compromise of the NHS infrastructure by the WannaCry ransomware presents a strong basis for addressing many issues seen across critical service infrastructures. Being such a high-impact incident, it emphasises the severe consequences that a cyber attack can have on a critical service. An audit report published by the UK National Audit Office after the incident [29] highlights a number of substantial shortcomings in the

---

<sup>21</sup>According to a paper published in the aftermath of the attacks, many other sectors across the world including transport and energy were also affected. [28]

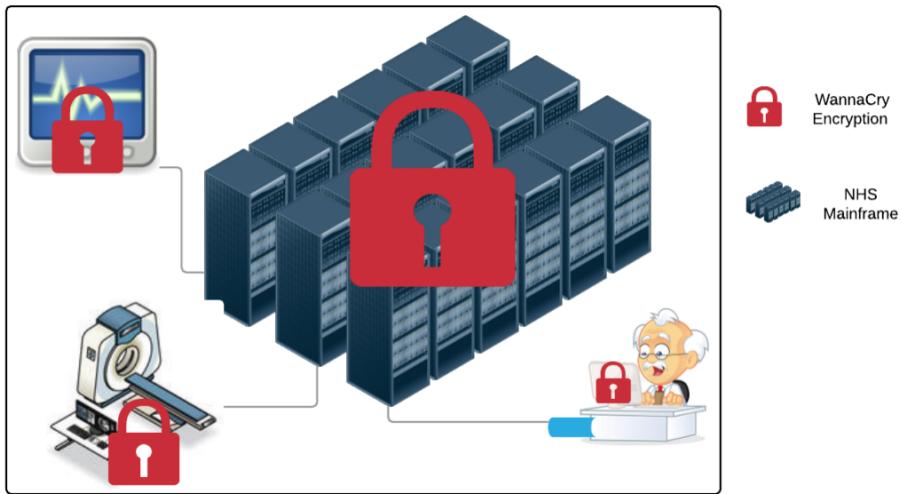


Figure 2.3: Illustration of WannaCry Infection in the NHS

An illustration of the infection of systems at NHS hospital sites by the WannaCry ransomware. The infection was not limited to user devices and servers, affecting systems including MRI scanners and blood test analysis devices. [30]

NHS security defence strategies up to 12th May 2017 when the attacks occurred:

1. There was no incident monitoring in place to alert security first-responders to any unusual behaviour in the NHS systems: Evident from the fact that it took over half a day to reach those who could take remedial action.
2. There was no centralisation of system data, meaning that security experts had to physically attend the affected sites in order to collect crucial data about the attacks. This further prolonged the period for which these systems remained inoperable.
3. Prior to the incident, recommendations regarding updates for the NHS IT systems were issued but not followed at the affected sites. Further to this, there was no system in place to determine whether or not the actions had ever been taken. Outdated and unpatched operating systems were a major contributor to the success of

the compromise, which was entirely based on a Windows exploit<sup>22</sup>.

Ultimately, the lack of monitoring within the NHS systems meant that when the attacks occurred, the systems at the affected sites remained offline for an extended period of time, causing chaos across the entire healthcare system in the UK. It is clear that in order to protect both their own systems and those of others that depend on them, organisations must place emphasis on security training for their members, the security and maintenance of their technologies, and the rigorousness of governance of their systems.

## 2.2 Intrusion Detection

An Intrusion Detection System (IDS) is a security application that is employed to detect attempts from an attacker to gain unauthorised access to a system or system resource. They are usually deployed in a region known as the De-Militarised Zone (DMZ), a sub-network separating an internal network from untrusted external networks, providing an added layer of isolation between internal and external systems<sup>23</sup>. Most traditional IDSs work off the basis of comparing activities to a defined security policy, and either permitting or denying the action on this basis.

In this section, the motivations for intrusion detection, along with two primary types of intrusion detection system, are discussed and compared.

### 2.2.1 Motivations for Intrusion Detection

IDSs play an important role in securing both enterprise and personal systems. Among the many advantages are the following:

---

<sup>22</sup>WannaCry and NotPetya, two ransomware strains which made their first appearance in 2017, both exploited this vulnerability. Officially named 'EternalBlue MS17-010' by Microsoft, a patch was released for it after major attacks by both of these ransomware strains. According to an article by Brian Krebs, [3] The EternalBlue vulnerability exploits the Microsoft Server Message Block (SMB) network file sharing protocol. SMB allows applications on a computer to read and write to files and to request services that are on the same network.

<sup>23</sup>Usually, content-serving systems are placed in this region of the network so that they can be accessed from external networks.

- By detecting an intrusion in the early stages, the intruder can be removed from the system before any damage is done.
- Attackers in general strive to remain anonymous and undetected. If a potential intruder has some knowledge of an effective IDS being in place in a system, they may be deterred from proceeding with an attack.
- Valuable information may be collected about the nature of the intrusions and how they took place, enabling administrators to continuously re-evaluate their system design and security strategies.

How each of these advantages are provided, and to what extent, is dependent on the type of IDS being employed. It should be recognised that threats can come from nodes both internal and external to the network in question, and so a maximally effective IDS will address both.

IDSs can be broadly categorised as either being *passive* or *active*.

- Passive IDSs are not proactive about preventing or interfering with attacker activities.
- Active IDSs are proactive about engaging with the attacker in order to counter an attack.

In this section, two particular IDSs will be discussed in detail: firewalls and honeypots.

### **2.2.2 Firewalls**

Firewalls are the most commonly deployed type of IDS, and are used in virtually all enterprise networks as part of an organisation's network security. In general, they are classified as a *passive network defence mechanism*. According to a literature survey by Voronkov *et al.*, "a firewall is a system consisting of software and/or hardware that is designed to prevent unauthorized access to/from a network or device". [31]

### 2.2.2.1 Overview

The basic principle of operation of a firewall is to act as a barrier on the periphery of a network, preventing intruders and malicious forces from gaining access to the system being protected. In practical terms, they filter network packets based on a defined security policy which is a set of configured rules, and either accept or reject a given packet according to this policy.

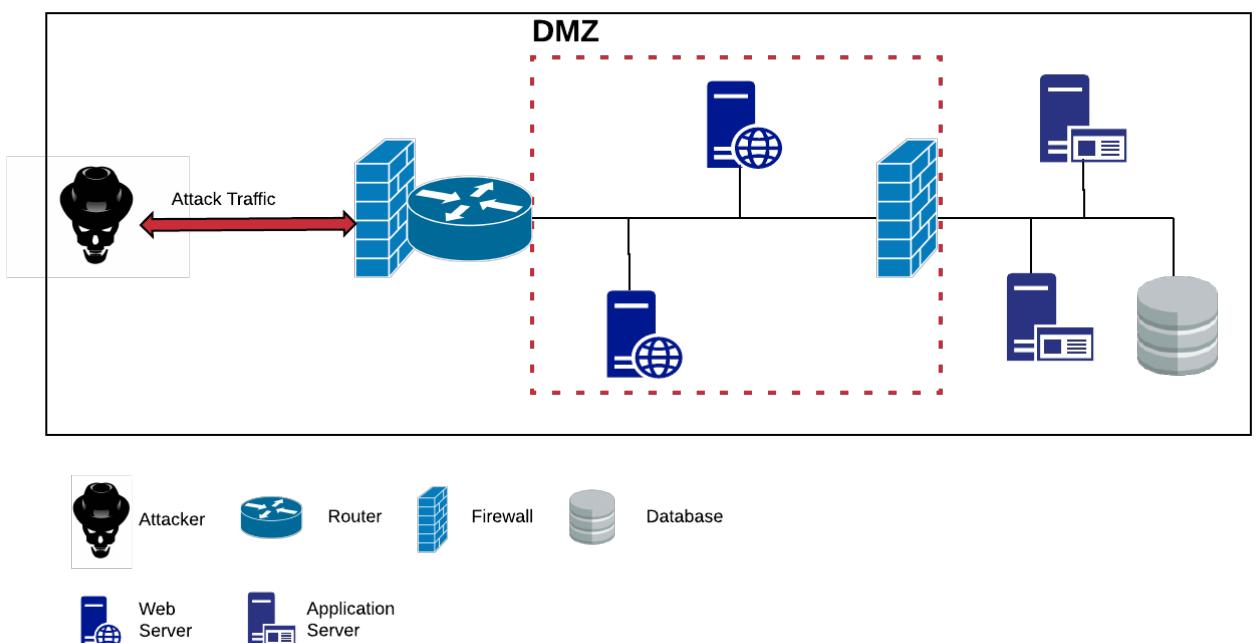


Figure 2.4: Use of Firewalls in a De-Militarised Zone

An illustration of a typical corporation network where firewalls have been employed at the internal and external entrypoints to the De-Militarised Zone (DMZ).

### 2.2.2.2 Design

A number of commonly employed approaches to implementing firewall policies are identified by Omar Santos in *End-to-End Network Security defence-in-Depth*. [32] Some of these are described as follows:

- Pattern Matching

The firewall tool searches for a fixed sequence of bytes within a packet, often aligned at a specific position corresponding to a field within the packet. It then makes a filtering decision by consulting its defined security policy.

One of the primary limitations of pattern matching is that it generally exhibits a high rate of false positives, making it relatively inaccurate as a technique of classifying packets that should be denied access.

- **Protocol Analysis**

Protocol analysis is accomplished by decoding protocol-specific communications, i.e. by examining the packets corresponding to particular communication protocols. The firewall will identify elements of the protocol and examine them for an infringement, for instance by examining explicit fields within the packets. An example for an SMTP packet might be examination of fields such as *HELO*, *MAIL*, *RCPT*, *DATA*, *SET*, *NOOP*, and *QUIT*.

- **Heuristic-Based Analysis**

Also known as *misuse detection*, this technique tries to identify intruders based on a set of known undesirable behaviours and patterns that can be determined according to rules. Systems built upon this concept are severely limited in intrusion detection capabilities since they cannot detect an intrusion for which no rule is defined.

- **Anomaly-Based Analysis**

Anomaly-based analysis techniques try to define the behaviour of a normal user over time, and compare the behaviour of all users to this in order to identify suspicious behaviour that may relate to an intrusion.

- **Deep-Packet Inspection**

Deep-Packet Inspection (DPI) involves closely examining information embedded in network packets in order to make more informed decisions about how to handle

the packet. This often allows for more effective mitigation against attacks such as DDoS, where DPI allows for the more accurate identification of packets that have undesirable contents or are in some way not legitimate.

#### 2.2.2.3 Challenges

There are a number of issues with using firewalls alone for intrusion detection. Though at a point in time these systems reflected the configuration of enterprise networks accurately, in the world of the modern web many of the premises of using firewalls simply do not hold any more. As enumerated by Steven M. Bellovin in *Thinking Security: Stopping Next Year's Hackers*, [33] the design and use of firewalls is based on the following premises:

1. A firewall operates at a *topological chokepoint* in the network it is protecting, such that it partitions two sections of the network.
2. All nodes being protected inside the firewall have the same security policy.
3. All nodes inside the firewall are trusted.

If all of these conditions hold, then a firewall will work well in the system in question. However, taking any modern IT infrastructure in an organisation today, most - if any - of these assumptions do not hold true.

- Enterprise networks increasingly deal with nomadic users, who are facilitated in constantly connecting to and disconnecting from these networks without any security auditing being performed.
- The devices that connect to enterprise networks today are incredibly diverse, and implement an even more diverse and inconsistent range of security policies.

On top of this, it is increasingly becoming difficult for firewalls to use packet inspection techniques to filter network traffic because of incompatibility with other security mechanisms. An increased uptake in TLS encryption and the push for packet header encryption

is causing enormous issues for the operation of firewalls, which cannot decrypt packets in order to make the required filtering decisions that it depends on.

There is no question that firewalls are no longer sufficient on their own in providing security for modern networks. They can be reasonably effective against known, less sophisticated attacks and are less effective against more sophisticated targeted attacks which are more likely to use new exploits. As Fred Schneider explains in his paper *Blueprint for a Science of Cybersecurity*, "a secure system must defend against all possible attacks - including those unknown to the defender. But defenders, having limited resources, typically develop defences only for attacks they know about." [34]

Overall, firewalls are most effective when used in combination with other security mechanisms, but their inability to classify and mitigate against new threats is a major limitation to their use. [33]

### 2.2.3 Honeypots

Well-known security expert and Honeynet Project founder Lance Spitzner defines honeypots as "a security resource who's value lies in being probed, attacked or compromised". Honeypots leverage the concept of deception in order to combat attackers. As explained in a paper by Cliff *et al.*, "the essential parts in cyber deception include crafted information by the defender (that will be used to mislead) and wrong actions taken by the adversary as a result of the deception." [35]

Put simply, honeypots are devices which masquerade as legitimate and systems in order to detect, track and analyse patterns of user behaviour when the system is illegally accessed. They are categorised as an *active network defence mechanism*, and are deployed solely with the intention that they will be attacked. A key characteristic of a successful honeypot is attractiveness to an attacker. 'Attractive' in this context means that the honeypot should appear to be exactly the device that the attacker is looking for: A device that has the potential to be easily exploited, whilst offering maximum value to the

attacker.

### 2.2.3.1 Overview

Honeypots are broadly seen as acting as both decoys and sensors in the network in which they are deployed.

- **Honeypots as Decoys**

A honeypot can be used to divert an attacker's attention away from the valuable components of the network in which it is deployed. This is particularly useful in a production environment, when there is a need to protect important systems on the network.

In order for this strategy to be effective, the honeypot should appear to be exactly what the attacker is looking for. By then being attacked, the honeypot can immediately notify system administrators of an intruder being present in the network, allowing them to take the appropriate course of action to secure it.

- **Honeypots as Sensors**

Honeypots can also be viewed as sensors, since they can collect valuable data about the attacks that they receive. This is particularly useful for detecting weaknesses and vulnerabilities in system design, since the captured attack data can be analysed to understand the attacker's behaviour and strategies as well as their motivations.

Thus, the value of honeypots is both in their ability to capture information as well as to defend the system within which they are deployed.

Amongst the many benefits of using honeypot technologies are the following:

1. **Configurability**

In general, honeypots are highly configurable and customisable and can be made to mimic real systems. This configurability allows those deploying them to adapt their defence strategy to tackle evolving attack behaviours.

## **2. Inside the Network**

Honeypots are deployed inside the infrastructure of the system they are protecting, rather than on the fringes as with firewalls. This allows for security much closer to the real systems that are being targeted by attackers.

## **3. Logging Capabilities**

Honeypots give a unique opportunity to capture valuable information about the nature of attacks being launched against a system, providing a means of observing attackers 'in the wild' without being detected. This gives system administrators a better chance of staying up-to-date with the evolving security requirements of their systems.

## **4. Few False Positives**

The premise of using a honeypot is that nobody should communicate with it: There is no legitimate reason to interact with a honeypot, since it is deployed with the sole purpose of attracting attacks. This means that typically, a honeypot will trigger very few false positives, since every interaction with a honeypot is automatically distrusted. This has added benefits for system administrators in that there is a lesser requirement to comb through very large log files in order to identify a significant event, which is common practice with firewalls and many other IDSs.

It is important to consider however that individuals within an organisation may mistakenly interact with the honeypot out of curiosity, something which can often be deduced from looking at the logging captured. In general, the presence of honeypots in a system should not be well-known within the organisation for this reason.

## **5. Offensive and Defensive**

As explained, honeypots are an active defence mechanism. They allow for learning about adversarial intent, capability, and techniques as well as thwarting attempts

to compromise real systems. By using honeypots, cyber attacks can be mitigated against by making access to valuable systems both expensive and ineffective.

### 2.2.3.2 Design

The design of honeypots to suit the context of their deployment is key to their ability to provide effective intrusion detection in a system. The primary design categorisations of honeypots were outlined in a paper by Mokube *et al.* [36] on the basis of (i) their level of interactivity with an attacker, and (ii) the context of their deployment.

#### 2.2.3.2.1 Interactivity

Interactivity levels of honeypots are an important consideration, which define (i) the ability of the attacker to interact with the honeypot, and consequently (ii) the volume and type of information that can be gathered by that honeypot. Levels of interaction range from simply allowing a connection to be made, to being able to download and install malware binaries. The cost versus learning benefits of honeypot interactivity levels increase proportionally, meaning that a highly interactive honeypot will likely be expensive to host and maintain.

Any effective honeypot must be capable of interacting at some level with an attacker, while also quietly monitoring their actions. There are three practical classifications of honeypot interactivity level: Low, high and medium.

- **Low**

Low-interaction honeypots are most commonly used to alert someone to the fact that an attack has occurred, but do not provide any means of interacting with the attacker or capturing the attack data. They are used in cases where a lower-risk solution is preferred: In general, a low-interaction honeypot will simply be an emulation of a real service, and so does not offer any opportunity for system compromise by an attacker. This can be a major limitation to their use, since there

is very little knowledge to be gained from preventing further interaction with an attacker.

- **High**

High-interaction honeypots are at the other end of the spectrum compared to low interaction honeypots. They are fully-fledged systems, and give the opportunity for attackers to use real applications during their interactions. A great deal can be learned about the nature of attacks from using high-interaction honeypots, since it is the closest thing to observing attacks "in the wild". The trade-off is the high-risk associated with the exposure of a real device to a malicious attacker, which in the worst case could see the entire system taken over by an attacker and used to launch further attacks on other devices. Some of the ethical considerations around this issue are discussed in *Section 4.2.2*.

- **Medium**

As described, the appropriate level of interactivity of a honeypot depends on the use context: Low-interaction devices are commonly used for detecting connection attempts, whereas high-interaction devices are live systems that can capture detailed information about the behaviour of attackers "in the wild". Medium interaction honeypots offer a good middle-ground, using emulated components of a real system to allow a level of interactivity with the attacker whilst not exposing any real systems that could be compromised.

A summary of the characteristics of each of these categories of honeypot are given in table 2.1.

**2.2.3.2.2 Deployment Scenarios** There are primarily two deployment scenarios for honeypots: Production and research. This distinction mainly distinguishes between the function that the honeypot needs to provide for the system in which it is to be deployed.

<b>Interaction Level</b>	<b>Information Quality</b>	<b>Risk</b>	<b>Maintenance Cost</b>
High	Excellent	High	Difficult
Low	Poor	Low	Simple
Medium	Good	Medium	Relatively Simple

Table 2.1: A summary of the characteristics of each different honeypot interactivity level, similar to that provided in a survey of honeypot technologies by Nawrocki *et al.*. [37] In general, the higher the interaction level the higher the cost, risk and value of information captured.

- **Production Honeypots**

Production honeypots are generally deployed in large, enterprise networks with the intention that they will act as a part of the active network defence in the organisation's infrastructure. An example of such a deployment is shown in figure 2.5. Their primary purpose is to act as a decoy, luring the attacker away from valuable machines on the network with a seemingly more valuable and vulnerable target. This enables the honeypot to alert system administrators early on to the fact that there has been an intrusion, giving them the opportunity to isolate the valuable devices in the network from the infected honeypot. As previously highlighted, it also enables system administrators to identify vulnerable points in their infrastructure.

The idea of attracting attackers into the system, encouraging them to interact with it and give away their attack strategies without causing any harm to the systems with real production value is the major attraction of using honeypots in a production environment.

- **Research Honeypots**

As the name suggests, these honeypots are generally deployed for the purpose of research rather than as a security measure. The emphasis with these honeypots is not so much on the ability of the honeypot to act as a decoy, and more on the ability of the honeypot to collect valuable data for analysis. By allowing attackers to interact with and infect the honeypot, research can be carried out relating to the

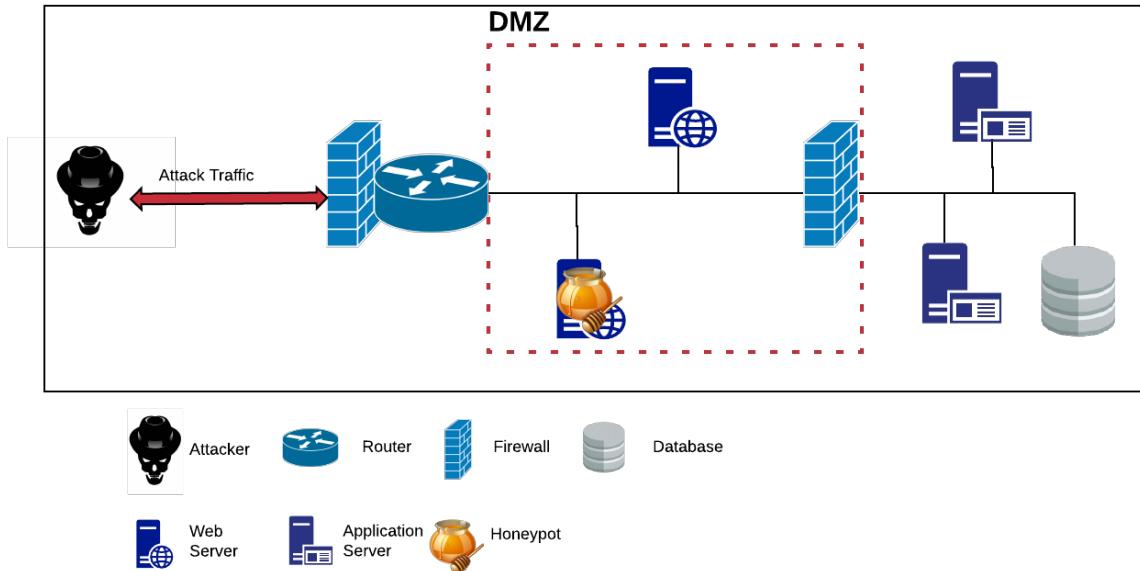


Figure 2.5: Use of Honeypots in a De-Militarised Zone

An illustration of a typical corporation network where honeypots have been employed in conjunction with internal and external firewalls, inside the DMZ.

behaviour and strategies of the attackers.

### 2.2.3.3 Comparison with Firewalls

When compared to traditional, passive intrusion-detection mechanisms such as firewalls, honeypots exhibit some crucial differences.

- Firewalls define all attackers passively and simply alert to the fact that a security incident, such as an unauthorised connection attempt, has occurred. A system administrator can then, for instance, blacklist<sup>24</sup> the source IP address of the connection to deny it access to the system.

However, if a new attack comes along for which there is no rule defined in the firewall's security policy, the firewall is not able to deal with this attack. This illustrates how firewalls can only provide *known-threat mitigation*.

---

<sup>24</sup>Blacklisting an IP address means that it is added to a list of addresses that aren't considered trustworthy.

- In contrast to firewalls, honeypots proactively entice attackers to give away their attack strategies and intentions by leaving their tracks behind on the device. This enables the same system administrator to do far more to enhance their system security policies and identify potential flaws in their infrastructure, allowing them to *improve* their system's security.

As well as this, the honeypot can attract and deal with previously *unknown threats*, giving them another edge over firewalls since they can provide *unknown-threat mitigation* as well as *known-threat mitigation*.

By using honeypots, system administrators are able to learn what attackers are targeting in their systems, enabling more effective defences to be implemented as vulnerabilities and flaws are identified. As suggested by Bellovin, "the world has changed ... the decision to rely on (firewalls) should be reexamined and perhaps abandoned". Despite this, firewalls are not obsolete and will continue to play an important role in IT and network security with their ability to detect and classify potential security risks at the edge of the network they are protecting. They are however best used in conjunction with active network defence mechanisms such as honeypots. [33]

#### **2.2.3.4 Honeynets**

Honeynets are networks of interconnected honeypots, which coordinate their efforts to provide active network defence. Given that they consist of more than one honeypot, they have additional abilities to capture valuable attack information regarding threats compared to standalone honeypots: For instance, it is possible to study the propagation of attacks from one honeypot to the next, and to simulate a variety of different systems that may attract different categories of attack.

Honeynets are significantly more complex than standalone honeypots given that they consist of a network of honeypot devices designed to be attacked. In general, a honeynet will contain *high interaction* honeypots so that an attack is able to propagate from one

real system to another real system. As with individual honeypots, any connections to a system in the honeynet are immediately distrusted and assumed to be malicious.

An element of a honeynet which is described by Lance Spitzner in *Honeypots: Catching the Insider Threat* [38] is something that he refers to as the *honeywall gateway*, a bridging device between external systems and the honeynet. All traffic between the external system and the honeynet must pass through this gateway. The honeywall will typically have two interfaces: One to connect it to the external web, and the other to connect to the honeynet. The honeywall thus acts to separate the honeynet from the external web. Since the honeynet is only reachable through the honeywall, which can monitor and control network traffic to and from the honeypots in the network, extensive extra monitoring can be performed on attackers of the system. Depending on the configuration, this component has the opportunity to capture data including network traffic and keylogging data.

In summary, using a honeynet allows for deployment of a highly-controlled active defence network where all activities of attackers are made visible. The insights that can be gained from observing the behaviour of attackers in an unrestricted networked environment are highly valuable to those who wish to continuously improve their system defences. However, the risk associated with hosting a system of this type is a major trade-off since there are many more points of failure in a network of intentionally vulnerable devices.

#### 2.2.3.5 Challenges

There are a number of challenges to the widespread deployment of honeypots in networks, outlined below.

##### 1. Detection

One of the major considerations when designing or using a honeypot-driven solution is the risk of the honeypot being detected as a non-genuine system. This is particularly relevant to low and medium-interaction honeypots which emulate a real

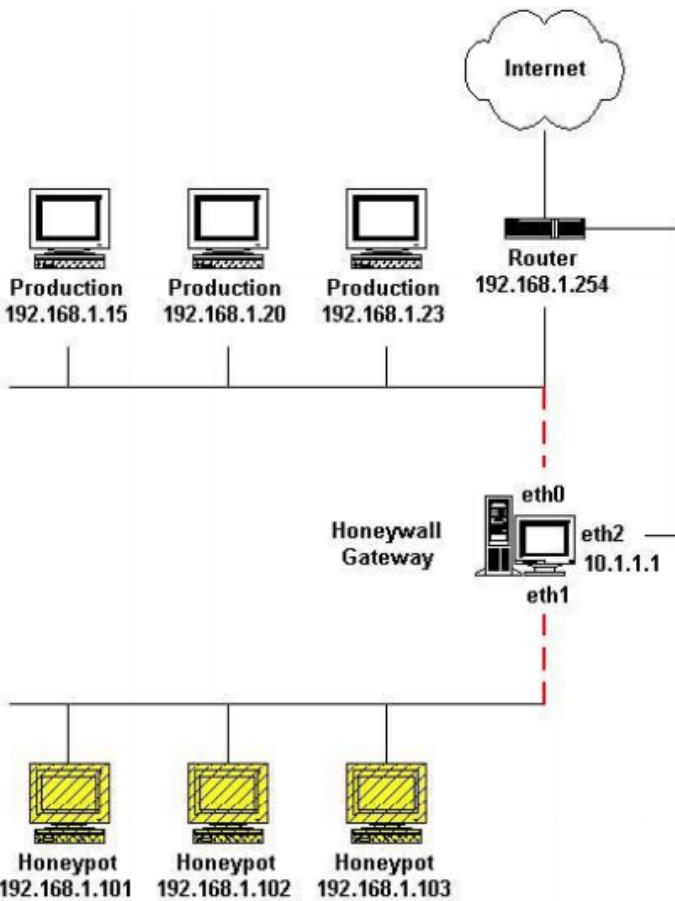


Figure 2.6: Lance Spitzner’s Proposed Honeynet Architecture

This figure shows a diagram taken from Lance Spitzners paper proposing a honeynet architecture. Original source: *Honeypots: Catching the Insider Threat.* [38]

system: It is never possible to exactly mimic the behaviour of the corresponding real system. This means that carefully crafted actions by an attacker may allow them to detect that they are not interacting with a bona-fide system by triggering different responses: A form of fingerprinting. It is an ongoing challenge for honeypot designers to combat fingerprinting checks by attackers.

In general, honeynets are less likely to be fingerprinted successfully than an individual honeypot since they provide a network of interconnected high-interaction honeypots providing real services. This kind of system is highly enticing to attackers because of the advanced system capabilities that it provides, and thus a less likely suspect as a non-genuine system.

## 2. Adoption

There are many challenges to overcome in order to make the use of honeypots more widespread. To many, the idea of inviting a malicious actor into a production system seems like a potentially destructive action, and so many system administrators do not consider honeypots as part of their security infrastructure. Canary, a honeypot solution developed by Thinkst Applied Research, highlight on their product landing page the challenge behind adopting honeypots in large networks: "With all the network problems we have, nobody needs one more machine to administer and worry about". [39] Although it could be argued that this is a sales pitch, this statement highlights the demand and lack of supply of feasible solutions for large organisations regarding deployment and maintenance of honeypot-based systems.

## 3. Ethical Concerns

The use of any surveillance technology, which is the category into which honeypots fall, has ethical questions associated with it. Honeypots are widely accepted as an ethical approach to intrusion detection and understanding cyber attacks. Since there is no legitimate reason for interacting with a honeypot, any interactions are likely to be of a malicious nature - in which case, the honeypot simply serves to counteract further attacks of this type in the future. Mokube *et al.* discuss these ethical issues in detail in their paper. [36] However, a discussion in a whitepaper published by the System Administration, Networking, and Security Institute (SANS), [40] it can be concluded that their use does not involve:

- Entrapment, since there is no inducement to attack the honeypot - rather, it is something that all devices are vulnerable to because attackers will always seek to attack devices within their reach, regardless of whether it may be a honeypot. As aptly phrased by security researcher Lance Spitzner [41], "attackers find and break into honeypots on their own initiative".

- Invasion of privacy, since surveillance mechanisms are broadly seen to be acceptable where they serve simply to protect their own environment. A commonly encountered example in physical environments is the use of CCTV cameras.

Overall, the ethical concerns around the use of honeypots are widely accepted to be resolved by the nature of the deployment of honeypots: Ultimately, their purpose is to defend against unethical actions by attackers. With reference to the research being undertaken in this project, ethical considerations are discussed later in relation to the design of the research environment in *Section 4.2.2*.

## **2.2.4 Cyber-Incident Monitoring**

Cyber-incident monitoring forms an important part of providing effective intrusion detection. Cyber incident monitors are platforms that support system administrators in identifying and acting upon threat intelligence collected through intrusion detection mechanisms such as honeypots and firewalls, primarily through providing enhanced usability and more effective communication. It is concerned with the effective management of intrusion detection tools, the recording of threat events, and the provision and presentation of threat information to first-responders.

### **2.2.4.1 Usability in Security**

It is widely accepted that the ability of a user to easily employ security systems makes a crucial contribution to their effectiveness: As argued by Sasse *et al.*, "security mechanisms are often too time consuming for people to bother with, or so complex that even those willing to use them make mistakes." [42] The use of security mechanisms adds an overhead to the use of systems almost without exception. Apart from the consequent disincentive for individuals to use these mechanisms, it becomes very hard to use them effectively when their use is not straight-forward.

In a literature review addressing the usability of firewall applications in particular, Voronkov *et al.* found that usability studies could greatly enhance the effectiveness of security mechanisms, since configuring them "is a process that is complicated and prone to error... misconfiguration of firewalls leads to a broad number of vulnerabilities in the network; environments with distributed and multiple firewalls just make matters worse." [31]

#### 2.2.4.2 Visualisation in Security

Visualisation techniques have become crucially important in delivering usable security solutions. Security problems are inherently complex: In the case of tools such as firewalls which work off the basis of a list of defined rules, it becomes very difficult to understand and maintain the overall security policy as the list grows.

The impact of using visualisation in security mechanisms is that threat intelligence is communicated more clearly and effectively to those consuming the data. Ease of visibility of the state of a system is a crucial element of active network defence in identifying and investigating potentially malicious activities and detecting unknown threats before they can do damage. As explained by Vasilomanolakis *et al.* in a paper detailing the development of a honeypot-based incident monitoring system, such a system "unif(ies) the various alert generators into a single system that assists the user to make informed decisions". [43]

## 2.3 Modern System Infrastructures

There is a significant shift in the hosting solutions that are being used by large organisations that rely heavily on their IT infrastructure, particularly for those providing services on such a platform. Outsourcing of resource management is now the preferred approach for many large organisations, given that specialised hosting providers are increasingly offering end-to-end solutions for hardware, network and security infrastructure manage-

ment.

The architectural design of these distributed services is also evolving. The concept of decoupling services by deploying them as microservices has enhanced operational efficiency in organisations, moving away from failure-prone monolithic designs of the past.

### **2.3.1 Infrastructure-as-a-Service**

In the past, organisations hosted their systems on servers and physical infrastructure that they owned and managed themselves. This approach was resource-intensive for organisations, motivating the development of IaaS. IaaS is a service model where physical infrastructure is provided to organisations on an outsourced basis including hardware, storage, servers, data centre space and network components, and the use of cloud platforms which allow an organisation to host their services on another organisation's hardware offers a practical solution to this.

#### **2.3.1.1 Benefits**

The benefits for organisations of outsourcing their infrastructure management to third-party providers include the following:

- There is a guarantee of maximum uptime since cloud platforms are a paid service.
- There is no need for upfront capital investment, since there are none of the installation or maintenance costs associated with hardware upgrades or expansion.
- There are a wide range of geographic locations available for deployment since hosting services are located all over the world, allowing organisations to host their services closer to users that are geographically dispersed.

In essence, using a cloud platform to host enterprise-level systems is highly economical, simplifying the management of resources and deployment with the result that they consume less of the focus of system administrators. With an ever-increasing number of

companies moving their operations to the cloud, a cloud-deployment appears to be the most feasible deployment solution for cyber-incident monitoring systems in the long-term.

### 2.3.1.2 Virtual Machines

Cloud platforms have typically leveraged virtualisation of the underlying bare-metal hardware through the use of virtual machines (VMs), which run software on top of physical servers to emulate a particular hardware system, and are the enabling component of IaaS. There are a number of components to any virtualised system:

- A VM host, which is a server that supports virtualisation.
- A *hypervisor*, or a VM monitor, which can be any of software, firmware, or hardware that is used to create and run VMs. The hypervisor sits between the physical host and the VM.
- The VMs themselves, which are isolated software environments that imitate dedicated bare-metal systems<sup>25</sup>.

VMs are relatively heavyweight and expensive to maintain, since each separate VM maintains its own operating system (OS) image<sup>26</sup>: An overhead in memory and storage footprint. This lack of flexibility also limits the portability of applications that run on virtual machines between different hosting solutions: It is difficult to replicate the same environment configuration on a second VM.

However, VMs allow for multi-tenancy on a single piece of physical hardware, as well as fine-grained resource allocation.

---

<sup>25</sup>The term *bare-metal* refers to the hosting and management of applications directly on the underlying hardware, without an intermediary software layer.

<sup>26</sup>An OS image is system image is a copy of the entire state of an OS stored in a persistent, non-volatile form. It is a package that contains all of the information required to run the OS on the machine, and is usually relatively large as a consequence.

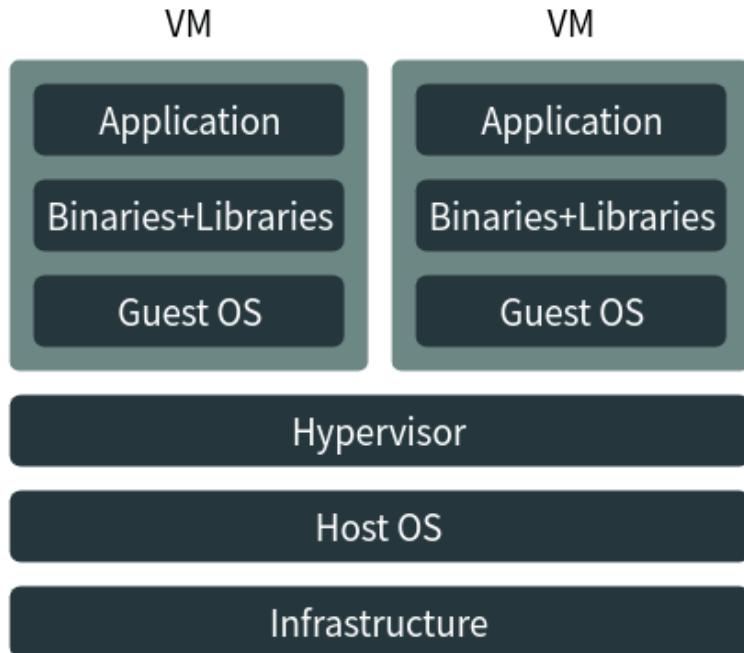


Figure 2.7: Components of a Virtualised Environment

A conceptual diagram showing the components of a virtualised environment. Original source: Boolean World. [44]

### 2.3.2 Platform-as-a-Service

The idea of PaaS is that users are able to focus on running and developing applications instead of managing the infrastructure that supports them. PaaS systems mean that a virtual infrastructure is provided and users can deploy any application on it without needing to worry how resources are allocated to it.

#### 2.3.2.1 Containers

A container is an isolated execution environment defined by a single package that includes everything needed to run it: Code, runtime, system tools, system libraries, settings. They are type of OS-level virtualisation, isolating an application from the host infrastructure on which it is being run: They are almost completely host-agnostic<sup>27</sup>. An application and all of its dependencies can be packaged up in a *container image* that can be published in

---

<sup>27</sup>Host-agnostic in this context refers to the fact that containers do not have an affinity to any particular host system.

a *container registry* so that unlimited numbers of users can work in identical application environments.

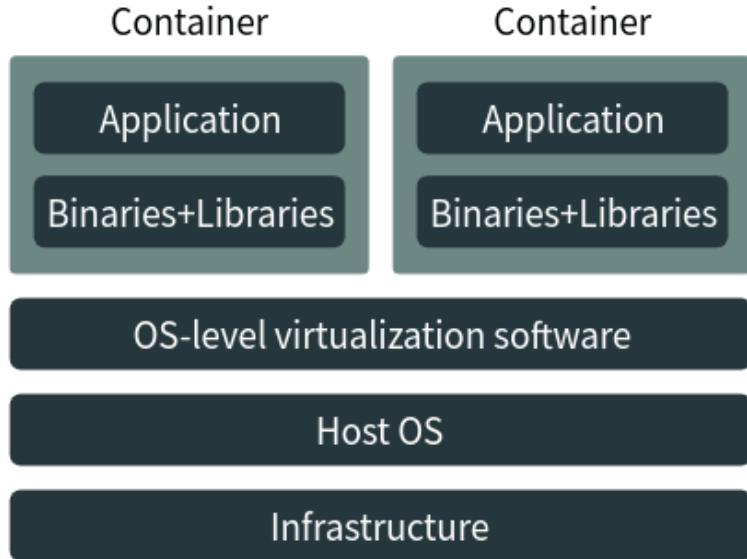


Figure 2.8: Components of a Container Environment

A conceptual diagram showing the components of a container. Original source: Boolean World. [44]

For setting up a standard application environment and deploying a large number of instances on-demand, containers are simple to configure and much faster to launch than VMs. Among the many advantages of using containers compared to VMs are the following:

1. Hosting a container does not include anywhere near the same storage overhead as a VM: Containers do not need to maintain their own copy of the OS image, since they share the kernel of the host OS.
2. All dependencies required by an application are packaged into a single, replicable image. This means that the portability of application environments that is so difficult to achieve with VMs, is native in containers.
3. Containers come with their own separate network stack and storage without the overhead of building and running a VM.

Many organisations are migrating their infrastructure deployments from VMs to containers. A recent example of this is Netflix, who started migrating their infrastructure hosting to the AWS cloud in 2008. At the time of writing in 2018, Netflix are migrating the hosting of all of their applications from VMs to containers. In a paper discussing this shift in Netflix’s infrastructure approach, [45] Leung *et al.* state that ”one of the benefits of using containers … is that it abstracts much of the machine-centric management that applications were doing in VMs”. This is one of the notable benefits of using containers over VMs: Those deploying the application no longer need to concern themselves with the environment specifics, and get the opportunity to focus on building and running applications.

### 2.3.2.2 Security through Containerisation

An area where much work remains to be done is in bringing security applications into the progressively more popular container ecosystem. There is tremendous potential for the containerisation of security applications, and a number of identified benefits of this approach are described below.

- **Disposability**

A major selling point of containers is that they are virtually disposable. In the event that a container becomes infected with malware, it is straightforward to remove it entirely and re-deploy it with minimal disruption to the system.

- **Isolation**

One of the key features of containers which has contributed to their widespread adoption is the ability to restrict and isolate application environments.

- Containers leverage a Linux functionality known as ”namespaces”, placing applications that run on them in restricted environments, isolated from each

other. Individual containers are run in their own independent namespace, and are given their own independent view of a variety of operating system resources.

- Containers can also be restricted such that any "root user privileges" do not have to correspond to the root user privileges on the host machine: The definition of "root user privileges" is completely separate from those of the host.

The net impact of this isolation on the security of containers is that the applications running inside one container will not interact, unless explicitly configured to do so, with applications inside another container or on the host machine. This is a substantial increase in security compared to running multiple non-containerised applications on the same system, and means that even if one container is compromised by malware, the others can continue to operate. This loosely-coupled approach to operating services has the capacity to deliver greater fault tolerance and scalability in systems in which they are deployed, reducing the exposure of the overall system to compromise.

- **Standard Base OS Images**

Containerised solutions benefit greatly from being able to build customised containers off standard, widely-used OS images. These standard images benefit from ongoing security auditing and patching, making them a secure foundation for building customised containers.

- **Customisability**

Containers are highly customisable, and enable fine-grained control for system administrators. It is possible to restrict everything from the privileges to the runtime services, enabling restriction of what resources can be accessed by any given container.

In an infrastructure ecosystem where services are increasingly being containerised, the containerisation of security applications is a must in order for them to remain compatible

with these new architectures. However, the introduction of new technologies into systems will always bring additional security risks, something that must be carefully considered by organisations migrating their operations to container-based architectures. [46]

## 2.4 Closely Related Work

The closely related projects described in this section are explored from the point of view of their most interesting characteristics rather than the achievement of their respective research objectives.

### 2.4.1 Adaptive Honeypots

The goals of honeypot deployments can vary greatly, and most honeypots offer a wide variety of options and different functionalities in order to provide flexibility to this end. As identified in *Section 2.2.3.2*, increasing the interaction capabilities of a honeypot results in more extensive and detailed information but also more potential damage to the system. Different trade-offs in honeypot design are clearly more suitable for some deployment scenarios than others.

At the time of writing, there were over 1,000 open-source honeypot projects listed publicly on GitHub<sup>28</sup>. These range from open-sourced production honeypots to major open-source collaborations and hobbyist projects. Research honeypots discussed in journals and conference papers add even more to this number, and there is little doubt that many proprietary production solutions also exist. [39]

Some of the more significant honeypot projects encountered during this research are described below. These include research honeypots and community-based honeypot developments.

---

<sup>28</sup>On 24th March 2018, a search for the query 'honeypot' provided 1,405 repositories as results.

#### 2.4.1.1 Kippo

Kippo is an open-source SSH honeypot written in Python which is no longer under active development. It is a medium-interaction honeypot developed by an online development community, and is designed to capture the entire session interaction of an attacker. [47] It emulates a Debian Linux installation, and provides a wide variety of features as follows:

- Kippo provides an out-of-the-box configurable file system and shell for each attacker to interact with. It does not limit the number of simultaneous sessions, and can present a mock file system and shell to each attacker independently.
- Kippo implements a falsified SSH service to which attackers can connect.
- Kippo includes configuration options for permitted usernames and password combinations which can be controlled by the user.
- All interactions with Kippo are logged, including brute-force login attempts, source IP addresses, attacker protocol information, commands executed and records of any attempted downloads.

The fact that Kippo is implemented as a Python application means that attackers will not interact directly with the underlying OS, adding an extra layer of protection for the host system. The fact that it doesn't call on any external software for its core operations makes it much less vulnerable to third-party compromises.

Kippo does not have the capability to execute malware, but can be used in conjunction with other solutions such as Cuckoo Sandbox<sup>29</sup> in order to execute and analyse malware safely in a controlled environment.

Unfortunately, Kippo is an example of a honeypot which is susceptible to fingerprinting due to the fact that it emulates a real system: In its emulation of the OpenSSH service, it is possible to trigger characteristic responses identifying the device as a Kippo honeypot

---

<sup>29</sup>Cuckoo Sandbox is an open-source malware analysis system, which allows malicious files to be executed inside an isolated environment in order to analyse their behaviour and properties. [48]

using carefully crafted messages. [37] This flaw led to the eventual cessation of active development on the honeypot, rendering it largely unusable in the development of new honeypot-based solutions.

#### 2.4.1.2 Cowrie

Cowrie is a medium-interaction honeypot which was originally forked from the Kippo project by security researcher Michel Oosterhof. It is actively maintained and supported by a dedicated community, who provide quick response times to queries and bug-fixes. It is widely used in both research and production projects as a medium-interaction honeypot. [49] [50] [51]

In addition to the capabilities and features provided by the Kippo honeypot, Cowrie has some additional features [52] [53] including but not limited to the following:

- Cowrie enables attackers to gain access to the honeypot using an emulated telnet client as well as SSH, supporting the two protocols which are most widely-targeted by IoT botnets. [20] [54]
- Cowrie logs all attack session information in JSON format, which is a widely-used format that can be processed by many tools;
- Cowrie provides integration with a number of useful tools including VirusTotal malware database and the Mailoney SMTP honeypot;
- Cowrie also improves substantially on its predecessor, Kippo, resolving many of the major fingerprinting issues.

Interestingly, there is also some activity with respect to containerisation of the Cowrie honeypot. A basic Cowrie container image has been developed by contributors to the Cowrie project, [55] which illustrates an increasing awareness of the need to migrate security applications into container-driven architectures.

#### **2.4.1.3 Honeyd**

Honeyd is an important predecessor to many of today’s honeypots, and is likely the best-known honeypot implementation. It is an open-source, virtualised low-interaction honeypot released originally in 2002, and is currently in maintenance mode. It was originally developed by Niels Provos.

Honeyd is flexible and adaptive in that it can be configured both to run arbitrary services and to advertise itself as running different operating systems, learning the service “personalities” by reading Nmap fingerprint files in order to mimic them. [56] It cannot however emulate system components such as file systems or command execution. As explained in an article [57] by creator Niels Provos, Honeyd ”listens to network requests destined for its configured virtual honeypots” and ”responds according to the services that run on the virtual honeypot. Before sending a response packet to the network, the packet is modified by Honeyd’s personality engine to match the network behaviour of the configured operating system personality.”

Additional features of Honeyd include the ability to create virtual routing topologies, and the ability to enhance configured routes with realistic latency and packet loss characteristics in order to appear more convincing. Though it has provided inspiration for finding adaptive solutions for honeypots, the fact that it is no longer being actively developed renders it largely unusable in new deployments.

#### **2.4.1.4 IoTpot**

IoTPot is a research honeypot developed by Yin Minn Pa Pa *et al.*, which they describe as ”a novel honeypot that emulates interactions of the telnet protocol and a variety of IoT devices”. [58] It is a reactive IoT honeypot that targets the study of telnet-based attacks. It is particularly interesting because of its ability to react to the requests sent by attackers, providing them with a dynamically generated response deemed to be most likely to match the CPU architecture of the system that is being targeted in the attack.

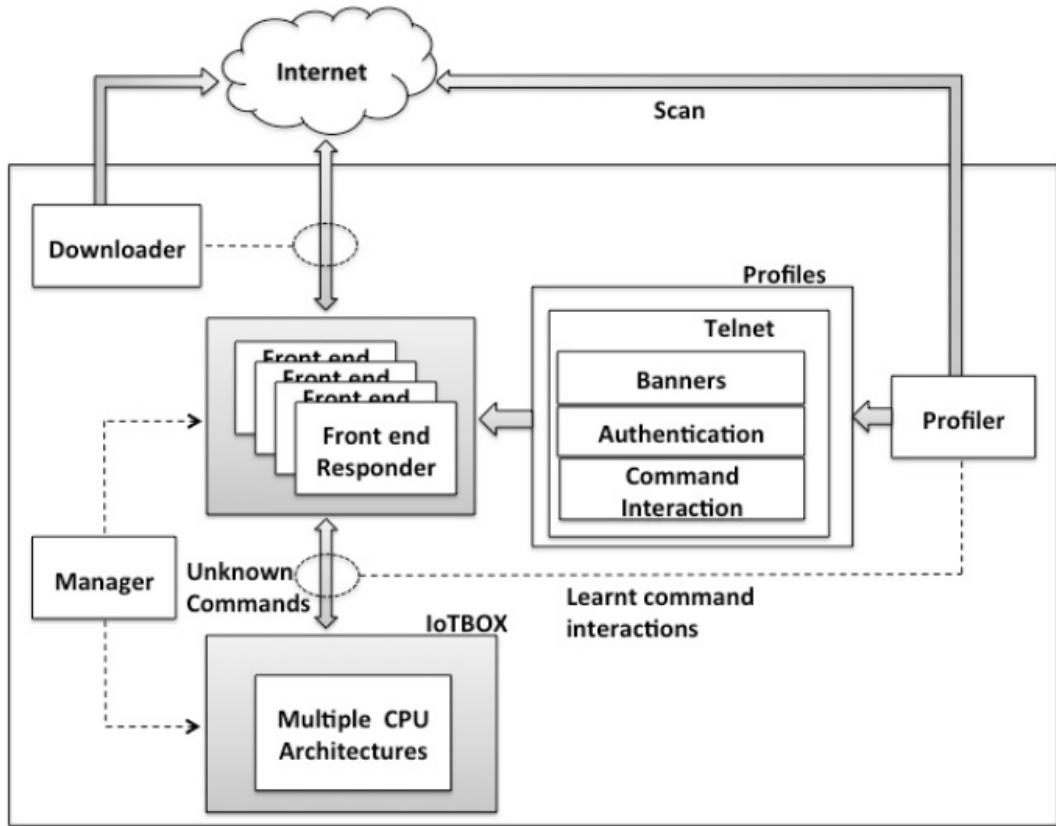


Figure 2.9: The IoTPot Honeypot Architecture

This diagram is originally from a paper published by the researchers who developed the IoTPot honeypot, and illustrates the operation of the IoTPot system. [58]

IoTPot was designed in recognition of the fact that there are too many different types of IoT device to be able to develop honeypots for all of them. The principles behind the operation of IoTPot are very much based on a honeypot network called SGNET, where low-interaction sensors forward attacks that they receive to a high-interaction honeypot to handle. [59] This is similar to the function of the *front-end responder* in IoTPot, which is the component of the system that interacts directly with attackers. A diagram of the system architecture can be seen in figure 2.9.

- The *front-end responder* component deals with negotiating the Telnet protocol with an attacker based on what it believes they are looking for. It emulates a number of different device profiles, and for each device profile deals with authentication attempts, command interactions and emulation of device-specific telnet options.

- If a command is issued by the attacker that the *front-end responder* doesn't know how to handle, it connects via Telnet to the *back-end responder*, a component called *IoTBox*. This component contains emulated operating systems for a number of different CPU architectures, meaning that the command can be executed on each of these and an appropriate response returned to the attacker as a result.

Most honeypots are far more passive in their approach to handling incoming commands from attackers, and are incapable of providing any more than one valid response to a given request. At the time that it was developed, the researchers claimed that there existed no other IoT honeypot "that can mimic IoT devices of many different CPU architecture while listening on 23/TCP with the ability to learn unknown command interactions."

As an adaptive telnet honeypot, IoTPot appears to be very effective. However, it is relatively restricted in that it can only deal with telnet-based attacks, limiting the attacks it can capture. According to the paper, the researchers "plan to extend IoTPot to support more protocols that are likely the target of attacks" in the future.

#### 2.4.1.5 IoTCandyJar

IoTCandyJar is a recent IoT honeypot developed by Luo *et al.* who classify it as an *intelligent-interaction* honeypot rather than as low, medium or high-interaction. [60] The researchers explain that "the goal of intelligent interaction is to learn the 'correct' behaviors to interact with clients from zero knowledge about IoT devices". Their approach is to utilise machine learning techniques to determine the most appropriate tactics to prolong attacks.

The IoTCandyJar honeypot is composed of a number of primary modules, which can be seen in figure 2.10 as follows:

- The *IoTScanner* module leverages the fact that there are powerful scanning tools freely available to identify IoT devices, using *Shodan*, *Censys*, *Masscan* and *Zoomeye* scanning tools to identify IoT devices on the internet. It actively probes these

devices, collecting their responses to various requests that have previously been captured by IoTCandyJar.

- The *IoTLearner* module builds intelligence into the honeypot, using the responses captured by the *IoTScanner* module to train a model using machine learning algorithms. The accuracy of the responses provided to requests improve with increasing numbers of interactions with attackers: If an attacker sends a second request after receiving a response from IoTCandyJar, the response is deemed to have been correct.
- The *IoTDatabase* stores all of the captured requests and responses, and stores any information relating to the machine learning model.

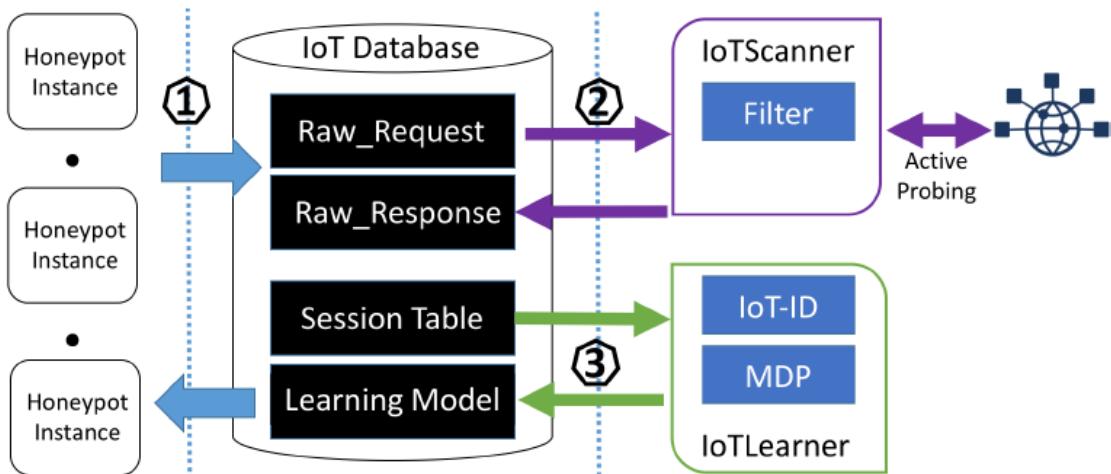


Figure 2.10: The IoTCandyJar Honeypot Architecture

This diagram is originally from a paper published by the researchers who developed the IoTCandyJar honeypot, and illustrates the operation of the IoTCandyJar system. [60]

The major selling point of IoTCandyJar is the dynamic approach that it takes to reacting to probing by IoT botnets. Similarly to IoTPot, IoTCandyJar aims to provide the best response to an attacker based on what they are determined to be seeking. The major difference between these two honeypots is in the approach they take to achieving this: Where IoTPot determines the most appropriate response by generating the correct

response through execution of the request, IoT Candy Jar learns the best response to a request based on interacting with attackers. Responses to attacker requests are generated based purely on what the system has learned from previous interactions with attackers, and not through any execution of requests. [60]

#### 2.4.1.6 Zigbee Honeypot

Dowling *et al.* developed a research honeypot targeted at attackers of Zigbee devices, which are devices commonly used in WSNs. [14] The development of this honeypot was motivated by the fact that as these IoT devices are being more widely-deployed and their vulnerabilities are also becoming well understood: Therefore, an assessment of the threat to these devices is essential.

In their implementation of the honeypot, Dowling *et al.* included a number of interesting measures to initiate the interest of attackers in the honeypot:

- The host name for the honeypot was configured to be *zigbee-gateway* as a means of assessing whether there is any awareness of Zigbee devices amongst attackers;
- The researchers implemented a means of transmitting of fictitious medical traffic, which is transmitted in cleartext over an unfiltered network.

The generation of potentially interesting network traffic in order to entice attackers to attack the honeypot is a thought-provoking feature of this solution. Though the research conducted is largely focused on identifying whether there is any awareness of Zigbee devices specifically, the idea of adapting the honeypot to the attackers it wishes to attract is a useful one.

#### 2.4.2 Containerised Honeypots

Related projects identified in the domain of containerisation of honeypots are limited in number, primarily because the security domain is still catching up with the migration of

infrastructures to container-based architectures. The research conducted in these related projects focus on both the benefits of using containers for hosting honeypots, and the challenges that are faced in doing so.

#### **2.4.2.1 Distributed Virtual Honeynets**

In 2014, Pisarcik *et al.* developed a distributed honeynet of high-interaction honeypots using containers for OS-level virtualisation, an approach which at the time was relatively unexplored in research. [61] Their motivations for developing such a solution were:

- The fact that administering honeypot-driven systems is time-consuming, motivating the use of containers to simplify their management; and
- The ability of honeypots to correlate attack events in order to determine whether they are localised or distributed in nature, quickly allowing for identification of attack trends.

Though the approach to designing and evaluating their honeynet solution is worthy of note, the more significant contribution made by this research is in relation to the containerisation of honeypots and honeynets. The research highlights that when compared to virtual machines or bare-metal systems, OS-level virtualisation incurs very little performance or maintenance overhead. [61] As well as this, they note that using containers adds an additional layer of deception to honeypots: Since they are isolated environments sharing the kernel of a real OS, they are more likely to appear as a legitimate system when fingerprinted.

All of these features would make it seem that containers are a perfect deployment solution for honeypots: However, they also address the potential security pitfalls of using containers to host honeypots, particularly because of the risk to the host system by virtue of sharing its OS kernel with containers. These are interesting insights regarding the practicality of using containers in honeypot-driven solutions.

#### **2.4.2.2 Using Linux Containers for Deceptive Honeypots**

In a paper published in 2017, Kedrowitsch *et al.* explore the use of Linux-based containers (LXCs) in honeypot deployments. [62] However, the use of LXCs here is examined from the perspective of their ability to evade detection by malware, which has often been found to change its behaviour after detecting a VM-based environment. [63] The motivation behind this is to identify whether container environments will be feasible in the long term as an approach to hosting honeypots without being detected by malware.

The researchers performed a number of experiments where they compared the fingerprinting weaknesses of LXCs to those of bare-metal systems and VMs. They focused on the typical techniques that malware employs in fingerprinting VMs: Comparing the response generated for a particular input against the response that would be expected from a bare-metal system. In particular, they chose to test and compare the following properties of VMs and containers as a means of fingerprinting:

- Variability and execution time in CPU clock sampling;
- Reported CPU information;
- Instruction execution time.

When evaluating their experiments, it was found that containers were much less susceptible to fingerprinting than their VM counterparts, which was expected due to the fact that LXCs are executed directly on bare-metal hardware as isolated processes whereas VMs are not. However, it was strongly concluded that if a malware author wanted to detect a container environment as a means of fingerprinting a system, they would have no difficulty in doing so.

#### **2.4.3 Honeypot-Driven Cyber-Incident Monitors**

Projects listed in this section are relevant since they make the use of honeypots much more accessible to individuals and organisations. They recognise the importance of data

aggregation and visualisation from multiple honeypots in order to provide important data to key decision makers, encouraging the use of active network defence by making it much easier to manage and benefit from.

#### 2.4.3.1 TraCINg

TraCINg is a honeypot-driven cyber incident monitor developed by academic researchers Vasilomanolakis *et al.* in 2015, [43]. Its development was motivated by the observation that the consolidation of data gathered from honeypots in different deployment contexts can allow attack data to be correlated, enabling the identification of emerging outbreaks of related attacks.

The TraCINg system gathers data from a large number geographically distributed honeypots with the aim of correlating attack events. Many of these honeypots are deployed on cloud platforms, which was the approach determined by the researchers to provide the greatest diversity of deployment locations and maximum system uptime for uninterrupted monitoring. Arbitrary open-source honeypots can be used with the TraCINg system, provided that their data is logged in JSON format.

This incident monitoring solution is not intended to be highly deployable or low-cost for use in production systems, but addresses the consideration of aggregating and summarising complex data in a meaningful manner through visualisation, enabling the succinct description of important data for key decision-makers. Their proposal for correlation of data from large deployments distributed honeypots is very interesting: However, their solution uses two low-interaction honeypots, limiting the level of detail of the attack data captured. An improved system would be able to capture more data about attack events in order to gain greater insights into the motivations and methods of attackers.

#### 2.4.3.2 TPot

The TPot project is a honeypot-based incident monitor that was developed by Deutsche Telekom, and which they use in their production networks for early threat detection.

They decided to open-source their development with the aim of making the deployment of honeypots in production networks more feasible for organisations. [50]

TPot supports multiple open-source honeypots hosted in containers. The system consists of a number of components, the most relevant of which are listed as follows:

- **Containerised Honeypots**

The TPot system leverages the benefits of containers listed in *Section 2.3.2.2* in order to provide a highly deployable honeypot system. The containerised honeypots supported by the TPot platform are all actively maintained and target a variety of different interactivity levels and attackers<sup>30</sup>.

- **The ELK Stack**

The Elasticsearch-Logstash-Kibana (ELK) stack is a collection of log management tools that perform search, logging and visualisation functions respectively. In TPot they are used to analyse, index and represent logged attack data captured by the honeypots.

An infographic provided by the TPot developers can be seen in figure 2.11. All container images for the honeypots are built based on the Alpine Linux<sup>31</sup> container image, which has a particularly small image size meaning that its storage footprint is minuscule. A benefit of running the honeypots inside containers that the developers highlight on their webpage is that they can "run multiple honeypot daemons on the same network interface while maintaining a small footprint and constrain each honeypot within its own environment". [64]

The proposition of leveraging the benefits of containers in order to host honeypots is an intriguing one, particularly since the difficulties of deploying and maintaining honeypots have been a huge barrier to their widespread adoption to-date. The work on this project

---

<sup>30</sup>The honeypots supported by the TPot platform at the time of writing are Conpot, Cowrie, Dionaea, Elasticpot, eMobility, Glastopf, Honeytrap, Mailoney, rdpv and VNCIowpot. [50]

<sup>31</sup>Alpine Linux describes itself as a security-oriented, lightweight Linux distribution based on *musl*, *libc* and *busybox*.

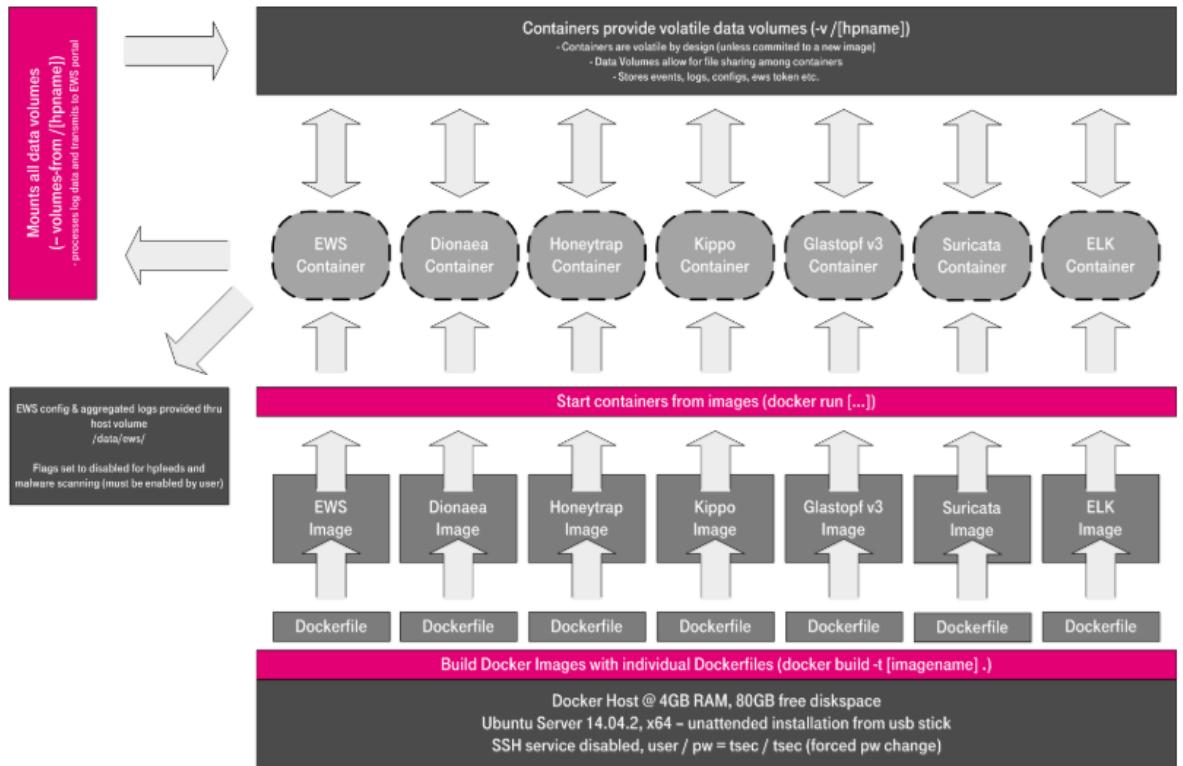


Figure 2.11: Diagram of the TPot Architecture

This diagram is provided by the developers of the TPot project as an explanation of the operation of the system. [64] Each honeypot is defined by a *Dockerfile*, which is built into an *image*. This image is then used to run the honeypot, and data generated by the honeypot is stored in volumes shared with the host system. The data can then be passed to other systems such as Deutsche Telekom’s in-house data aggregation tool *EWSPoster* as shown.

by Deutsche Telekom strengthens the case for the coupling of containerised honeypot deployments and visualisation tools.

#### 2.4.3.3 Modern Honey Network

The Modern Honey Network (MHN) is a production system developed by Anomali Inc. to manage enterprise honeypot deployments. Their decision to open-source the system was motivated by similar reasons as those given by the TPot developers, being that the deployment and maintenance of honeypot deployments has always been ”a complicated process reserved for security companies and security researchers.” [51] The result is that the MHN simplifies the process of deployment and management of honeypots, making it

more feasible for organisations to use active network defence.

The MHN is a centralised server system which is concerned with managing the deployment of honeypots as well as the aggregation of their data. It has support for a large number of open-source honeypots, including the Cowrie honeypot<sup>32</sup>. All of these honeypots are low or medium interaction open-source honeypots, which the MHN team say were selected to minimise the risk associated with a deployment of honeypots in a production environment. Honeypot data collected by the MHN system can be visualised with Splunk, a data analysis and visualisation tool: However, unlike TPot's support for the ELK stack, Splunk is not shipped as part of the MHN system.

## 2.5 Summary

Based on the background provided in this section, it is obvious that the field of cybersecurity is dynamic and full of challenges. As aptly phrased by Fred Schneider, "like good health, cybersecurity is never going to be a *solved problem*".

Many of these challenges are the result of gross negligence on the part of manufacturers and system designers; Others are due to obvious oversights and mistakes resulting from a lack of usability; Yet others are a consequence of not understanding the implications that insecure technologies can have for privacy and safety. What all of these present is an opportunity for improvement of current approaches to the implementation of technologies and connected systems.

The work done by those researchers and developers behind the *closely related projects* described in *Section 2.4* indicates the awareness and interest that exists around tackling the challenges that are being faced in the security of modern systems. It is a similar interest and awareness on the part of the author which motivates the decision to improve on the current state of the art in this area.

---

<sup>32</sup>The full list of supported honeypots is as follows: Amun, Cowrie, Conpot, Dionaea, ElasticHoney, Glastopf, Shockpot, and Wordpot. MHN also supports threat detection tools including Snort, Suricata and p0f.

# Chapter 3

## Problem Formulation

As discussed at length in *Chapter 2*, the rapid advancement of security threats in response to technological developments and innovations makes it crucially important for mitigation techniques and tools to keep pace. One thing is abundantly clear: That there is an urgent need for re-evaluation of current best-practice in cybersecurity defences, particularly for critical service infrastructures whose general lack of implemented security mechanisms is positively dangerous. It is this consideration that is at the heart of formulating objectives for this research.

*Section 3.1* explores a number of the major challenges identified in *Chapter 2*, seeking out the heart of the issues and hypothesising about how these can be resolved in tandem.

*Section 3.2* addresses the identified challenges in the formulation of proposed work for this research, and outlines the basis of the research objectives. How these research objectives will be addressed in practise is the subject of *Chapter 4*.

### 3.1 Identified Challenges

A number of challenges currently facing the security of modern infrastructures and systems were explored in *Chapter 2*. The challenges found to be of most interest to this research are outlined in the following subsections.

### 3.1.1 Incident Monitoring in Critical Service Infrastructures

As explained in the context of the WannaCry ransomware attacks on the NHS in *Section 2.1.3.2*, IT systems in many organisations are comparable to a *black-box system*<sup>1</sup>, an illustration of which can be seen in figure 3.1. Such a system gives no indication of what is happening internally until it begins to behave unexpectedly, meaning that when issues arise they are difficult to diagnose and almost impossible to resolve. However, ineffective feedback mechanisms are often as bad as having no mechanism in place: Passive mechanisms like firewalls are imprecise in their suspicion of malicious activity, meaning that when a significant event does occur it can easily go unnoticed. [31] It is crucially important that *effective* feedback mechanisms are employed which are simple to use and capable of communicating key information to those who can act upon it.

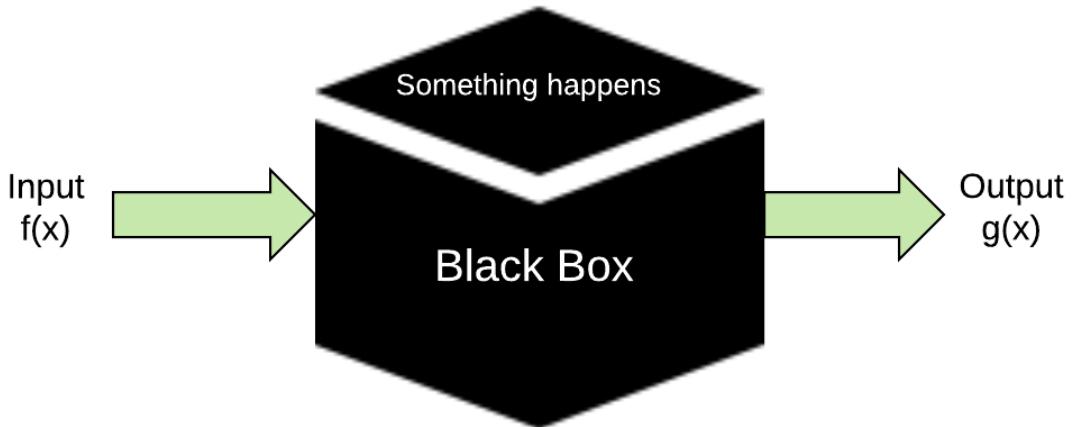


Figure 3.1: Illustration of a Black-Box System

An illustration of a black-box system, where a known input  $f(x)$  is provided to the system, unknown operations are performed on it, and a known output  $g(x)$  is generated as a result.

It is obvious that in order to implement effective threat mitigation mechanisms, the most relevant risks to that system must first be understood: Something that was highlighted in an interview with Stuart Reed, senior director at NTT security, who argues

---

<sup>1</sup>This is a term commonly used in control systems theory meaning that the inputs and outputs of a system are known whilst the internal workings are not.

that "the best starting place is to understand where... risks come from". [65] To illustrate this, consider critical service infrastructures such as the NHS: These are most at risk from those with a political motivation to do damage to a nation-critical infrastructure, such as nation-state attackers. While such infrastructures generally rely on firewalls which may protect against script-kiddies and other unsophisticated attackers, nation-states generally employ sophisticated technical attackers who are capable of discovering entirely new flaws and exploits in systems. This motivates and highlights the need for a technology that is able to provide intelligence on these active, dynamic threats for critical service infrastructures.

### 3.1.2 The Need for Active Defence Mechanisms

As discussed in *Section 2.2.2.3*, although the use of passive intrusion detection technologies such as firewalls hold an important place in securing IT systems, they have limited diagnostic capabilities since they can only mitigate against *known threats*. In an analogy comparing cybersecurity to health, Fred Schneider explains that "as with medical problems, some attacks are best addressed in a reactive way". [34]

Honeypot-driven solutions are promising in what they can offer with regards to active network defence and rapid incident response. Modern IT infrastructures are sorely in need of such *adaptive* approaches to defence: Defence mechanisms that, in the face of uncertainty about the nature of attacks, can continue to provide threat mitigation. It is clear from the discussion provided in *Section 2.2.3* that honeypot solutions have the ability to address this requirement.

- The information that honeypots can capture enables continuous improvement of infrastructure design and defences;
- The low number of false positives that result from honeypots make identifying threats straight-forward: All logged events can generally be assumed to be threats;

- Honeypots are capable of alerting first-responders to highly-probable threats as soon as they occur, enabling rapid response before an attack has the opportunity to spread.

However, these benefits are identified in recognition of the fact that the widespread deployment of honeypot-based systems to-date has been impeded by the maintenance overhead that they incur for system administrators, as was identified by the developers of Deutsche Telekom’s TPot as outlined in *Section 2.4.3.2*. In order for such a system to be practical and feasible to deploy in a production system, it must be:

1. Inexpensive to deploy, operate and maintain;
2. Effective and reliable in securing a network;
3. Simple to use.

In order to be viable for widespread adoption as a security service, a honeypot-based deployment solution must address all of these requirements.

### **3.1.3 The Effectiveness of Honeypots in Enticing Attacks**

As identified from the closely related work explored in *Section 2.4*, although there has been excellent progress made in the enhancement of honeypot design so that attack sessions are prolonged and more attack data is captured for analysis, there is little conclusive research found regarding how to enhance the design of honeypots in order to initiate the interest of attackers.

In almost all research found, there was a heavy focus on encouraging the prolonged interaction of attackers with honeypots rather than on attracting that interaction in the first place. [53] [58] [60] [14] Dowling *et al.* explored some interesting measures to entice attackers seeking to compromise Zigbee devices, but the impact that these measures had on the number of attacks attracted was not a metric of interest in their research. [14]

The effectiveness of honeypot-based technologies in defending systems is influenced greatly by their ability to attract attacks away from valuable systems in the environment they serve to protect. Understanding how honeypots can be designed to be as irresistible as possible to attackers would further their adaptivity in rapidly alerting first responders to the fact that they have been attacked, allowing action to be taken before a subsequent attack is targeted at valuable systems.

## 3.2 Proposed Work

It is clear that taking an active approach to mitigating risks to critical service infrastructures is the way forward, and that the capabilities of honeypots can aid greatly in achieving this. Though there are a number of existing approaches both in research and production outlined in *Section 2.4* that have contributed excellent work towards such a solution, there exists no one solution which can provide a practical, feasible means of employing active network defence as a single networked deployable unit.

Given the challenges identified in *Section 3.1*, two research objectives were formulated as a proposal to tackle them.

### 3.2.1 Objective 1: A Honeypot-Driven Cyber Incident Monitor for Critical Service Infrastructures

This objective addresses the need for a feasible active intrusion detection solution to enable rapid response in critical service infrastructures by enabling action to be taken on threats with minimal operational overhead.

The requirements of such a solution include:

1. A deployment solution that is cost-effective, quick to deploy and quick to scale, whilst compatible with existing IT infrastructure.

2. A number of highly-attractive honeypots, which can effectively lure attackers away from valuable machines in the network in which they are deployed.
3. A centralisation point for data collected from any deployed honeypots, such that administrators may be given a holistic view of their system in order to facilitate them in making quick, well-informed decisions.
4. An effective and reliable alert system, such that the key parties can be alerted as soon as possible after a threat event is detected by the system.

The constituent components of such a system are addressed in the following sections.

### **3.2.1.1 Honeypots for Active Network Defence**

As outlined at length in *Section 2.2.3*, honeypots enable the identification of vulnerabilities and flaws in the system in which they are deployed and are capable of mitigating against risks that were previously unknown to those administering the system, unlike passive defence mechanisms like firewalls. This makes them an obvious choice for the implementation of active network defence in critical service infrastructures.

In the proposed incident monitoring system, the use of a number of honeypots in a honeynet configuration has been identified as a ideal approach to providing effective incident monitoring. There are multiple benefits of a honeynet-based approach here, including the potential to study attack propagation through a network and the fact that there is an increased honeypot attack surface to distract an attacker away from valuable systems in the same environment. The honeypots in the proposed system would act as sensors, providing a feedback mechanism regarding threat events in the network by logging attack activities for analysis.

### **3.2.1.2 Containerised Honeypots**

Containers have been identified as an effective deployment option for the proposed system, given the properties that were outlined in *Section 2.3.2.2*, since they cater for many of

the requirements of a scalable honeypot-based incident monitoring system as outlined in the previous section.

By deploying honeypots in inter-networked containers it becomes possible to provide a flexible, low-cost, low-maintenance deployment solution for organisations. A single container will only use a relatively small proportion of a machine’s CPU power and storage, and a number of containers on a single host can use the same physical network interface whilst implementing their own independent network stack. [62] Importantly, any container can be deployed and removed with minimal operational overhead because of their portable configuration. In the case of critical service infrastructures, using such an architecture would enable recovery of any containerised component of the system after compromise with minimal downtime.

From the perspective of overhead and maintenance, containers are an excellent choice as a hosting platform for applications in today’s IT infrastructures. The migration of security applications to container-based architectures is both beneficial and essential for organisations hosting their infrastructure in containers: The success of containerised honeypots in the TPot platform illustrates this. [50] As well as this, containers enable a degree of reproducibility not usually achievable in computer science research, as pointed out by Cito *et al.*. [66] All of these benefits give a strong basis for the use of containers in hosting a honeypot-driven solution.

### **3.2.1.3 Visualisation of Honeypot-Generated Data**

Analysis of honeypot data is simplified greatly through use of visualisation tools, which enable classification and visualisation of complex attack data in order to make meaningful deductions.

As part of a honeypot-driven incident monitoring solution, it is clear that information gathered from honeypots would benefit from being aggregated, analysed and visualised in a central management system that consolidates the data and perform processing operations on it before making it available to a user: Both the MHN and TPot projects

exemplify this approach. [50] [51]

Such a management system is a crucial component of the cyber incident monitoring system, where attack data is gathered centrally and made accessible to system administrators who require a holistic view of the state of the system. A management system performing this function should be directly reachable by the honeypots, but obscured from attackers within them.

#### **3.2.1.4 Notification System for Threat Events**

The challenges faced in the aftermath of the WannaCry ransomware infection of NHS systems have illustrated the importance of having a feedback mechanism in place, which is capable of notifying first-responders in the event of undesirable behaviour being detected. An effective cyber incident monitor should provide this feedback mechanism, allowing threat data to be presented to those who can act on it as soon as it is detected.

### **3.2.2 Objective 2: A Proposal for Adaptive Honeypots**

It is clear that the effectiveness of honeypots in intrusion detection depends heavily on their design, which in turn depends on the context of their deployment and the value that they are required to deliver. The single point of commonality between all honeypot deployments is that they must be *adaptive* in order to deceive attackers effectively, assisting them in what they believe to be the successful progression of their attack whilst quietly monitoring their actions.

This need for adaptivity is beginning to be addressed in research more commonly, but there are gaps in current research that have provided an opportunity for improvement. In particular, all of the closely related honeypot projects identified in *Section 2.4.1* which address the adaptivity of honeypots focus on approaches to *maintaining* the interest of an attacker, rather than on initiating the interest in the first place. [53] [58] [60] Designing adaptive honeypots such that they advertise the most commonly sought-after

characteristics of victim systems is a problem of major interest.

### **3.2.2.1 Measuring the Effectiveness of Honeypot Characteristics**

Conducting a robust evaluation of the effectiveness of particular honeypot characteristics in attracting the interest of attackers is an interesting but difficult problem. The effectiveness of a honeypot is difficult to quantify, but it is clear than an effective honeypot will provide effective intrusion detection and, depending on the context of its deployment, capture relevant attack data in a certain level of detail.

Honeypot effectiveness can potentially be measured through such properties as the number of attacks that are attracted, the number of true positives detected, the ability to deduce attacker motivation based on data captured, etc. [37] Credible measurement of any of these properties is largely dependent on maintaining a consistent research environment within which relative performance can be evaluated accurately.

A honeynet is an effective way of accurately and simultaneously measuring the appeal of certain characteristics of honeypots in experiments, since multiple characteristics can be trialled in the same environment over the same period of time. By using a honeynet configuration in the research environment, data can be captured from multiple honeypots under the same environmental conditions, and the relative performance based on slightly different configurations can be measured.

### **3.2.2.2 Targeted Honeypots for the Internet of Things**

From the outset of this research, one of the most interesting and active categories of botnet identified was that of IoT botnets, a revolutionary and recent phenomenon which is cause for serious concern.

Proposing an approach to designing adaptive honeypots with a focus on making them maximally attractive to IoT botnets would contribute to current knowledge of these attackers by understanding more about their preferred vulnerabilities and target systems. The configurability and informative capabilities of honeypots makes them an attractive

option in this regard. Targeting these attackers in the design and implementation of the proposed system would be a valuable additional contribution to the field of security as an application of the cyber incident monitor.

# Chapter 4

## Design

To meet the research objectives outlined in *Chapter 3*, the requirements defined in *Section 3.2* must be addressed in the design of the proposed system. With these in mind, an initial system design was devised.

There are countless factors that must be taken into consideration when designing a research environment, and particularly when a flawed design can have far-reaching implications and consequences for the security of others unaware of the project in the first place. In order to develop solutions that were both secure and effective, every aspect of the design required scrutiny and careful planning.

*Section 4.1* details the *initial* design decisions that were made to address the functional requirements of the research environment.

*Section 4.2* addresses a number of identified challenges that required additional analysis to fully satisfy the requirements of the proposed system.

*Section 4.3* provides a summary, recognising in particular the unpredictable nature of research as it evolves.

## 4.1 Functional Considerations

This section describes many of the functional design considerations and decisions made relating to the implementation of the research environment. These include initial design decisions regarding the environment deployment platforms, the configuration of the honeynet and the design of the monitoring system.

### 4.1.1 Deployment Platform

Selection of the hosting infrastructure and platforms for the proposed system was an important consideration in the design which would have a significant impact on the successful development of cyber-incident monitor as well as on the design proposal for adaptive honeypots.

#### 4.1.1.1 Choice of Hosting Solution

The benefits of outsourcing the hosting of services to a third-party cloud platform were discussed at length in *Section 2.3.1*. As explained by Vasilomanolakis *et al.*, "cloud services provide resilience and uptime reliability ... which ensures uninterrupted monitoring." [43]

Amazon Web Services (AWS) is a subsidiary of Amazon Inc., which describes itself as "a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow." [67]

The AWS Elastic Compute Cloud (EC2) service is a very flexible means of quickly setting up configurable server instances on-demand. Though other hosting solutions including DigitalOcean and Trinity College's OpenNebula infrastructures were considered, the flexibility of options offered by AWS as well as the fact that it is an industry leader in providing cloud services resulted in it ultimately being the hosting platform of choice for this research.

#### **4.1.1.2 Use of Containers**

After first deciding that containers were the platform of choice for deploying honeypots in the proposed system, it was decided that Docker container technology would be used to implement the research environment. Docker is the industry standard for container solutions and is an open-source technology with a global support community.

Some of the benefits of using of Docker containers in particular for the implementation of the research environment include:

- The fact that each container has an independent networking stack, simplifying the implementation of a honeynet compared to virtual networks or bare-metal.
- The independence and flexibility of isolated container storage.
- Docker repositories, the use of which means that a rigorously-audited base OS image can be used as a foundation for the honeypot containers and then built upon with customised configuration options.
- The existence of research regarding the security of Docker technology. [68]
- The presence of comprehensive documentation for the Docker ecosystem as well as a globally active support community.

All of these factors would enable confident decisions to be made regarding the use of Docker in a security application.

#### **4.1.1.3 Host Operating System**

It was decided that Linux-based operating systems would be used for both the EC2 VM hosts and the container images used in the research environment, for a number of reasons.

1. Linux is regarded to be the most commonly deployed operating system for servers

and embedded devices<sup>1</sup>. Targeting the solution at deployment in Linux-based infrastructures is thus most likely to be the real deployment scenario for such a system.

2. Linux is natively supported by almost all open-source and industry tools, which would enable the use of almost any technology in the implementation of the project.
3. It is widely accepted that as a very popular OS for powerful servers, a very large proportion of cyber attacks are targeted at Linux-based systems.

#### 4.1.1.4 Choice of Honeypot

Motivated by the level of activity of IoT botnets, it was decided that an IoT honeypot was the most appropriate choice for this project. The Cowrie honeypot, which was discussed in detail in *Section 2.4.1.2*, is a mature open-source IoT honeypot. It was identified as an ideal option for targeting IoT botnets in the proposed deployment, particularly since it is an SSH/telnet honeypot: The two protocols which have been found to be most highly targeted by recent IoT botnets. [20] [54]

The fact that Cowrie is medium-interaction means that the risks of compromising a real system are mitigated whilst also enabling a relatively high level of interactivity with an attacker. As an open-source project it is also highly configurable, both offering a high degree of customisability and the ability to extend the source code. This is important in order to successfully conduct experiments with a view to proposing effective design approaches for IoT honeypots.

Crucially, Cowrie allows for a high level of interactivity with attackers, whilst also mitigating the risks of compromising a real system such as not providing the ability to execute code but instead storing the SHA-checksum of any attempted downloads. The fact that it is being actively maintained and supported, and that the rudiments of a containerised solution for the honeypot are already in place, are all selling points for the use of the Cowrie honeypot in this research.

---

<sup>1</sup>Though it is not possible to determine the exact number, in the research conducted by Yinn Min Pa Pa *et al.* it was measured that 91% of hosts found to be scanning the darknet were running Linux. [58]

#### 4.1.1.5 Container Capabilities

It is clearly important to consider how restricted the capabilities of honeypot containers should be in order to capture attacks, whilst also keeping the systems secure. Since the Cowrie honeypot will be running in these containers, if an attacker managed to circumvent the Cowrie application they should be restricted in what activities they can engage in when interacting directly with the underlying container. Additionally, if an attacker was to discover they were being monitored by a honeypot, it is plausible they could launch an attack on the platform on which it is hosted. As quoted from Upi Tamminen a.k.a. *desaster*, the original developer of the Kippo honeypot, "By running kippo, you're virtually mooning the attackers. Just like in real life, doing something like that, you better know really well how to defend yourself!" [69]

These considerations are addressed more fully in an exploration of the security considerations of using Docker in *Section 4.2*. However, overall precautionary measures identified regarding restricting container capabilities include:

- Running the honeypot containers as a non-root user, restricting the default privilege level of the containers.
- Installation of a minimal quantity of Linux utilities such that an attacker who manages to interface directly with the container will have very few tools at their disposal.

#### 4.1.1.6 Honeynet Design

As part of the design of the honeynet, a component that provides the functionality of a *honeynet gateway* as described in *Section 2.2.3.4* is required. This device is the first component of the honeynet that an attacker will interact with, and so should be maximally attractive as a gateway router which if compromised, promises the attacker the compromise of many more devices.

In order to develop an effective solution using a honeynet, the final design should incorporate measures to meet a number of requirements.

- It should be simple to configure and maintain control of each component of the honeynet independently.
- Honeypots in the honeynet should be isolated from the host system as much as possible.
- It should be easy to scale the honeynet on-the-fly by adding or removing honeypots.
- The configuration of the honeynet should be portable such that it is straight-forward to remove and redeploy the honeynet environment.
- It should be possible for an attacker interacting with the honeywall component to map out the configuration of the remainder of the honeypots in the honeynet, facilitating attack propagation and the capture of valuable attack data. It should not however be possible to detect that the system is a honeynet.

It was well-understood that deploying and configuring a network of honeypots to meet all of these requirements would be an error-prone and time-consuming process, particularly given limited prior experience with the Linux networking stack.

Given that it had already been decided to host the project on an AWS EC2 VM instance, hereafter referred to as the *honeypot instance*, this instance would need to be dedicated to hosting multiple containers, inter-networked to appear as though they are individual machines within the same network. The idea of a *honeywall* discussed in Lance Spitzner's 2003 paper [38] is clearly a component of major interest in the implementation of the honeynet: This component must be capable of being used as a *jumpbox*<sup>2</sup> with outbound network connection capabilities in order to give an attacker access to the rest of the honeynet.

---

<sup>2</sup>A jump-box is a term commonly used in networking to refer to a dedicated device on a network that is used to manage devices in a separate security zone.

#### **4.1.1.7 Ease of Deployment**

As outlined in *Section 3.2.1*, in order for a honeypot-based system to be feasible to use in a production environment it is key that it is easy to deploy and maintain. In particular, if a honeypot becomes compromised in an attack it should be simple to remove and redeploy it in the system whilst also persisting any data captured by the removed honeypot. Containers have been identified as providing many of these required functions.

One of the significant benefits of container technologies is the ease with which identically configured environments can be deployed. However, the same ease of deployment for a fully-networked system is not something that can be catered for by container technologies. This made it important that a deployment mechanism for the entire system be developed such that it could be removed and reinstated in minimal time.

The identified approach to achieving this was to capture the configuration of the system in a set of bash scripts, which could be used to automate the deployment of the entire honeynet environment. These scripts could be maintained and updated as the configuration of the system evolved, such that if the system was removed it could be identically redeployed using the scripts. These scripts should also include some simple usability measures such as a *usage* function<sup>3</sup> to assist in the efficient and simple deployment of the system.

#### **4.1.2 Incident Monitoring**

The incident monitoring component of the proposed system would handle the aggregation, processing and visualisation of data as well as the generation of threat detection alerts. A number of decisions needed to be made regarding how best to achieve this, as explained in the following subsections.

---

<sup>3</sup>Such a function, when specified as a parameter to the script, will typically list the arguments and execution options that are available from that script.

#### **4.1.2.1 Isolating the Monitoring System**

It is clearly crucial to ensure that the monitoring system which an administrator will rely on for key threat information is minimally susceptible to compromise. Given that the proposed honeypot-driven system is actually intended to invite attacks, this becomes an even more important consideration.

It was decided on this basis that physically isolating the monitoring system as much as possible from the honeypot system would reduce the likelihood of such an event occurring. Hosting the monitoring system on an independent remote EC2 instance would deliver some of this isolation, and so was included as part of the system design. This component is hereafter referred to as the *management instance*. This brings some questions regarding obscuring communication between the honeypot instance and the remote monitoring system into the picture, since it is undesirable for an attacker to notice regular communications between one of their victims and another valuable device.

#### **4.1.2.2 Secure Transfer of Honeypot Logs**

As data that an attacker would not want captured in relation to their activities, it should not be possible for an attacker to view or to redirect the logs generated by the honeypots to another destination when they are being transported between the honeypots and the monitoring system. Thus, the mechanism for transferring logs between the honeypot instance and the management instance needed to be able to support both origin/destination authentication as well as confidentiality and integrity.

Filebeat is a widely used open-source log-shipping agent which transports log files from client servers to another host. This is exactly the kind of functionality that is required to send the logs generated by the honeypots from the honeypot instance to the management instance. As a technology that is widely supported and used, Filebeat seemed an obvious choice for transporting the log files in the research environment.

Importantly, with Filebeat it is possible to add the required layer of security by gen-

erating an SSL certificate and key pair for the management server. The certificate can then be shared with the honeypot instance, ensuring that Filebeat sends encrypted data only to the trusted management server and vice-versa. [70]

#### 4.1.2.3 Visualisation of Honeypot Log Data

Visualisation is clearly a key component of the proposed cyber-incident monitor, enabling greater usability and informational value through succinct description of attack data captured by the honeypots.

The ELK stack is a well-established means of log analysis, aggregation and visualisation. Based on its successful use in the TPot development [50] and the fact that it has been endorsed as a visualisation approach for the Cowrie honeypot, [53] it was decided that the ELK log management stack would be used to process and visualise log data generated by the honeypots<sup>4</sup>. The Filebeat application discussed in *Section 4.1.2.2* above is also coincidentally developed by the same group, meaning that the interoperability of these tools should be straight-forward to achieve.

#### 4.1.2.4 Intrusion Alert/Notification Mechanism

PSAD is an open-source intrusion detection tool that is used to detect port scans and other malicious traffic on Linux systems. It operates by monitoring the networking logs of the device, and from this determining whether a scan or attack event has occurred.

PSAD was identified as being a suitable alert mechanism for the purposes of the cyber-incident monitor, since by examining the networking logs of the honeypot instance email notifications could be configured to be sent to a specific email address upon noticing suspicious activity. This would enable timely notification of undesirable activity in a production environment so that remedial action can be taken as soon as possible.

Prior to this, another alerting mechanism built by the same group as the ELK Stack

---

<sup>4</sup>Though alternative visualisation tools such as Splunk used by the MHN project were also explored, [51] the ELK stack was generally found to be less complex and so was the favoured option.

and Filebeat was considered as a potential notification option. The XPack plugin is capable of generating notifications based on the logs received by the ELK stack. [71] However, after consideration it was realised that this would mean that an alert would only be sent to an administrator after the honeypot logs had been processed by the management instance, meaning that there was potential for an unbounded delay in time-to-notification<sup>5</sup>. This realisation motivated the decision to use PSAD, which would generate the notification as soon as suspicious activity was identified.

## 4.2 Challenges

Additional to the functional design considerations and decisions outlined in the previous section, a number of specific challenges to the implementation of the proposed system were encountered. These were carefully explored and resolved as part of the design of the system.

### 4.2.1 Fingerprinting Honeypot Environments

As highlighted in *Section 2.2.3.5*, the fingerprinting of honeypot environments by attackers presents an ongoing challenge for honeypot designers.

As was discussed by Barron *et al.* in their study of attacker behaviours using the Cowrie honeypot, it is likely that an informed attacker would be able to fingerprint the Cowrie honeypot as not being a legitimate environment and disconnect. [49] For example, by default the Cowrie honeypot assigns hostname *svr04* to the honeypot. An attacker interacting with this honeypot who is aware of the default properties of Cowrie will certainly be aware of this default value and detect that they are interacting with a Cowrie honeypot.

---

<sup>5</sup>It is possible that in an unreliable network, logs generated by the honeypots that contain crucial attack information may be delayed in reaching the management server, meaning that crucial time would be wasted before an alert can be sent to a system administrator. In this time, it is possible that the attacker could have compromised the entire system, rendering this option infeasible.

However, in order to conduct a fair comparison between the impact of differently configured characteristics of honeypots in the experiments in this research, it was important to have a plain, out-of-the-box Cowrie installation in the honeynet as a control. Thus it was concluded that in spite of fingerprinting concerns, in each experiment iteration there would need to be one such Cowrie honeypot deployed in the honeynet.

#### 4.2.2 Ethics of Honeypots

There are a number of potential ethical issues associated with a project using honeypots that have already been outlined in *Section 2.2.3.5*. A further ethical concern is the ability for a honeypot to be compromised, and then used as an attack platform to launch attacks on other devices.

It is difficult to balance obtaining useful results from research with the inherent risks of implementing a honeypot. This consideration required careful deliberation, after which the following conclusions have been drawn.

- This risk can safely be viewed as the general risk associated with using devices connected to the web, and not necessarily related to the use of honeypots in particular. This view is in line with that held by Nawrocki *et al.*, who discuss the issue in a survey on honeypot software. [37]
- Although it is never possible to entirely eradicate the risk of attack propagation, the honeypot components of the project should be implemented so that they provide mitigation against this risk. The containers in which the Cowrie honeypots would be hosted would be heavily based on the official *Docker-Cowrie* Docker image<sup>6</sup> [55] developed and endorsed by the maintainers of the Cowrie development community. This container image runs the Cowrie honeypot under a non-root user account with minimal services available within the container, so that even if an attacker did

---

<sup>6</sup>The version of the Docker-Cowrie image off which the images in this research were based was that of January 29th 2017.

bypass the Cowrie application and gain direct access to the container it is unlikely that they would be able to launch any attacks on other devices.

- Dedicated research environments where honeypots are isolated from important devices and networks are also effective for protecting external systems. The fact that AWS instances are being used to host the research environment means that the system is being hosted in an environment with enterprise-level security. Given this fact, it is highly unlikely that an attack would have the ability to do any significant damage if it did propagate outside of the system.

#### 4.2.3 Persisting Volatile Container Content

A characteristic of Docker containers is that by default, their contents are not persisted once they stop running: As explained by the developers of the TPot project, "all data in docker is volatile. Once a docker container crashes, all data produced within its environment is gone and a fresh instance is restarted." [64] This is a design feature of containers: As well as enabling a container to be run without any affinity to a specific host, by separating the file system of a container from that of its host it is possible to achieve enhanced security through isolation.

In this system, the persistence of logs generated by honeypots running inside these containers is crucial. The solution to this data volatility issue, according to the TPot developers, is to have "persistent storage ... on the host in order to make (the data) available and persistent across container or system restarts." [64] This can be achieved through the use of *Docker volumes*, a persistence mechanism available for container directories. Docker volumes are the recommended approach to persisting data that is generated and used by Docker containers. When a container accesses a volume, Docker's storage driver is bypassed and the container interacts directly with the mounted volume on the host file system.

It was decided that given these considerations, Docker volumes should be used to

persist the log data generated by the honeypot containers in the research environment.

#### 4.2.4 Docker Security Considerations

As with any technology, there are security concerns about Docker. Though Docker Inc. have a historically excellent record for keeping on top of security of containers and providing extensive documentation, [72] there are inherent risks in using containers that have been highlighted in literature. [46] [68] [62] [73] [61] These have been very carefully contemplated during the undertaking of the project.

The greatest security concern identified regarding the use of Docker containers is the potential for an attacker to propagate the attack to the system hosting the containers: Compromise of this system is highly undesirable.

- Inherently, Docker containers share the kernel of the host, and many claim that this increases the attack surface of Docker containers when compared to VMs: As described by Chelladzhurai *et al.*, VMs are only able to communicate with the VM kernel, and not with the host. [73] This is a concern that is also raised by Kedrowitsch *et al.* and Pisarcik *et al.* in their use of containers for deploying honeypots. [62] [61]
- As described in *Section 4.2.3*, a number of Docker volumes are being used to mount directories on the host system to each container. This presents a potential security risk, since an attacker inside a honeypot container could potentially interface directly with the linked host volume and place whatever they want into the mapped directory.
- The Docker engine is known to automatically create a number of virtual networks on the host on which it is running, so that containers can communicate with the host and with each other. Any connectivity between containers and their host is a risk to the host, since it could potentially result in an attacker connecting into the host machine over a Docker network and compromising a real server.

These concerns are not insubstantial, and required careful consideration to handle correctly. The result is a number of reasonable precautionary measures to be incorporated into the initial design as follows:

- The isolation characteristic described in *Section 2.3.2.2* in relation to containers addresses the concerns regarding the increased attack surface of containers compared to VMs. Since each container runs in its own namespace and uses its own network stack, provided that careful consideration is given to the shared resources to which it has access and to the privilege level allocated to the container it is possible to sandbox the container quite well from the host machine. This is discussed at length in the Docker security documentation. [72]
- All honeypot containers to be used in the research should be configured to run the bare minimum of services required to facilitate attacks. This is another measure designed to ensure that the activities of an attacker inside a container are relatively limited.
- The containers to be used in the implementation of the honeynet should be built upon well-maintained base OS images from the official Docker Hub repositories. [74] This will ensure that the most recent security updates available for the base images are inherited by all containers as a robust security foundation.
- An effective measure identified by Chelladhurai *et al.* [73] is to restrict the networking capabilities of containers as much as possible, allowing communication only between trusted containers. To facilitate this, careful attention needs to be given to the configuration of the containers such that they do not expose any unnecessary ports or belong to any unnecessary networks.

#### 4.2.5 Enticing IoT Botnet Attacks

Clearly in order to attract the attention of IoT botnets, identified as the target of choice for the honeynet deployment, the honeypots should be as attractive as possible based on the characteristics of devices that have been found to attract IoT botnets. [20] [23] [54] In their paper, the developers of IoTPot discuss the importance of (i) supporting all options that attackers want to use, (ii) providing realistic login interfaces and (iii) facilitating logins, in the successful design of an IoT honeypot. [58]

Upon analysis of related publications and research, a number of IoT device characteristics which are likely to attract IoT botnets were identified as follows:

- The ability to use brute-force authentication to access a device, i.e. weak or default device credentials.
- The presence of either or both of the SSH and telnet protocols on the device.
- An unpatched or vulnerable version of either services present on the device, or of the OS.
- The presence of "interesting" network traffic to or from the IoT device, particularly if that traffic is not encrypted, similar to what was implemented by Dowling *et al.* [14]
- The ability to fingerprint the device as a product of a particular vendor whose devices are widely known to be vulnerable.

These characteristics could all be considered as potential variables in the experiments to be conducted, assisting in the proposal of an effective, adaptive design for honeypots targeting IoT botnets.

#### 4.2.6 Identifying Attacks from IoT Botnets

A further difficulty in evaluating any results captured by the honeynet is that of how attacks by human attackers can be distinguished from automated attacks by bots. This distinction is particularly important towards the second objective of the research, where a proposal for designing effective honeypots targeted at IoT botnet attacks is desired.

In a paper detailing observations about attacker behaviours elicited through the use of a large deployment<sup>7</sup> of Cowrie honeypots, Barron *et al.* note that "bots perform specific environment-agnostic actions... human attackers are affected by the underlying environment, e.g., executing more commands on honeypots with realistic files and folder structures". [49] Some of their more specific conclusions are that:

- Humans care about user files with realistic looking names and data, whereas bots do not;
- Humans make spelling mistakes when using a shell, whereas bots do not;
- Bots will behave in the same way regardless of the host that they are running on (i.e. environment agnostic);
- The time taken between subsequent commands is greater for a human attacker than for a bot, whose sequence of attack commands is automated;
- Most attackers are bots.

These observations are useful in distinguishing human attackers from bots. In order to distinguish IoT botnets from other botnets, observations from literature regarding active IoT botnets offer the most value in discriminating between IoT bots and other bots. [20] [54] For example, both the Mirai and Hajime IoT botnets are known to test for the presence of the BusyBox shell<sup>8</sup> on victim devices. [75] This test is conducted in order

---

<sup>7</sup>The experiments conducted by Barron *et al.* included 102 individual Cowrie honeypots.

<sup>8</sup>BusyBox is a lightweight shell which combines miniature versions of common UNIX utilities into a single small executable, making it very popular for use in IoT devices.

to verify that the device is running a real Linux shell: As explained by Edwards *et al.* in a paper discussing the Hajime IoT botnet, [54] ”a proprietary CLI is likely to reject the command, but a legitimate Linux shell would execute Busybox ... letting Hajime know that it has a bona fide Linux shell”. These kinds of indicators that can be observed through examining logged attack sessions will also aid in identifying the type of botnet behind any attacks captured by the honeynet.

#### 4.2.7 Credible Evaluation of Honeypot Experiments

As already highlighted in *Section 3.2.2.1*, the use of a honeynet in the proposed research environment enables credible evaluation of the relative performance of different characteristics of honeypots. The design of such an experimental setup however requires that there are as few variables between honeypots as possible. Thus, in the honeynet it is necessary to include:

- A plain, baseline ”control” device with no enhancements, as discussed in *Section 4.2.1*;
- A variety of improved honeypots, each exhibiting a different variation of the same potentially attractive characteristic.

Taking this approach means that for the purposes of evaluation, the sources of variability in the experiment are maximally controlled: All honeypots will be exposed to attack under the same environmental conditions, with a single controlled variation per honeypot. This approach eliminates the need to account for many otherwise significant environmental variables, such as if the honeypots were each evaluated independently on different host servers. Such variables could otherwise bring the credibility of claims about relative effectiveness of different honeypot characteristics into question.

## **4.3 Summary**

It was recognised from the beginning that given the unpredictability of research, and in particular the unpredictability of attacks, the requirements of the system and the resulting design were likely to evolve as the project progressed. However, a solid, robust and considered design based on informed decisions, and supported by previous research and current best practice, are sure to contribute positively to the effective implementation of the overall solution.

As part of a research progress update for this system 3 months after it had begun, a conceptual diagram similar to that provided in figure 4.1 was produced to give a high-level overview of the system architecture and operations.

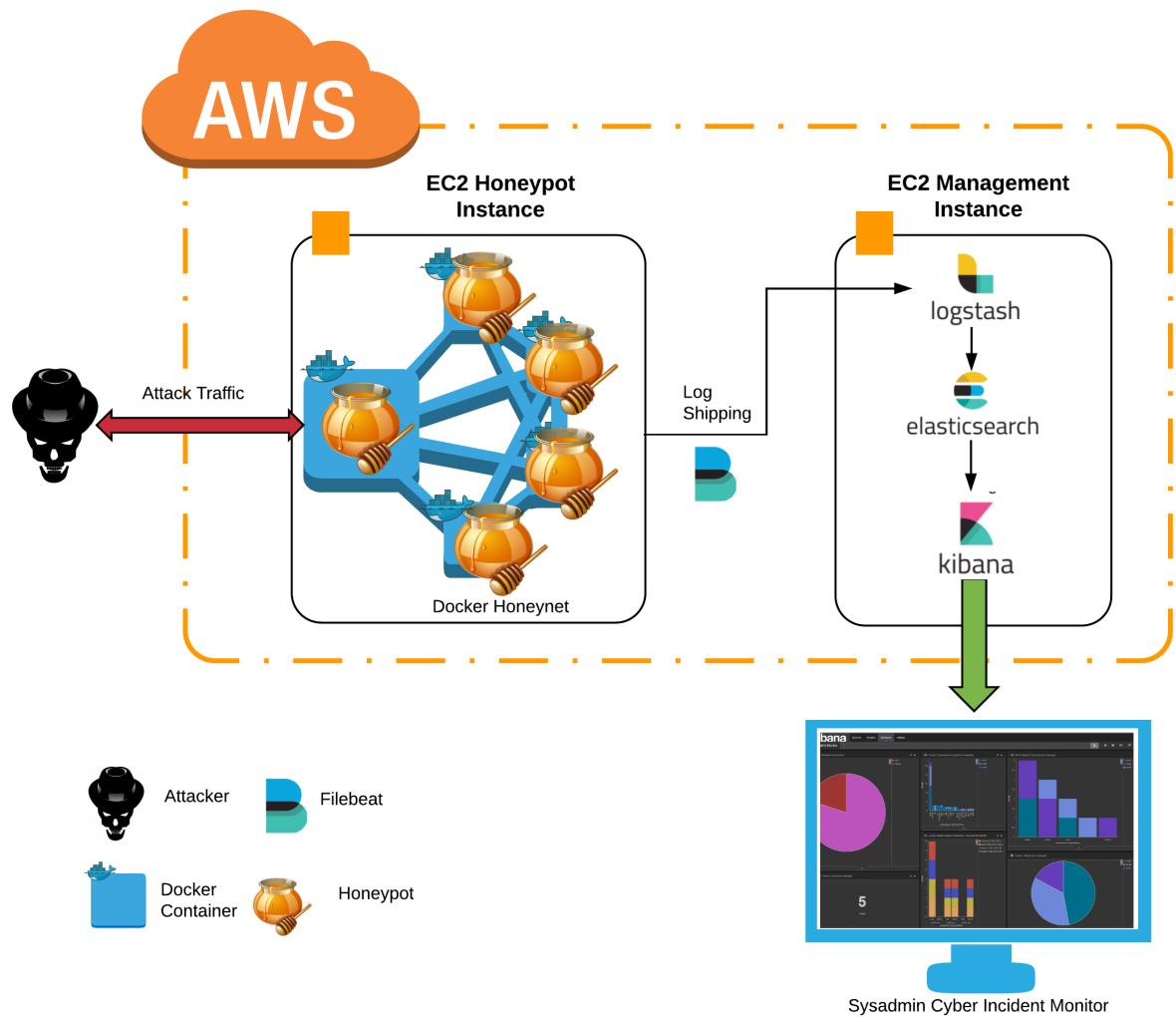


Figure 4.1: Overview of the Proposed System Design

A high-level overview of the intended architecture and components of the system. The components include 2 AWS EC2 servers, a container network consisting of containerised honeypots hosted on the EC2 honepot instance, an ELK log processing and visualisation pipeline hosted on the EC2 management instance, and a web console access to the visualisations generated by the system.

# Chapter 5

## Implementation

In this chapter, the implementation of the various components of the research environment are described in-depth. This includes the challenges that were encountered in the implementation as well as alterations that were made to the system design as issues were unearthed.

In the interest of succinctness, low-level details of installation of tools etc. are not explained in great detail. Instead, attention in this section is given to the implementation of components of the system, detailing decisions that were made and the challenges that were faced in doing so.

*Section 5.1* describes an initial exploration of the major components that would be involved in the implementation of the proposed system: Namely AWS EC2, the Cowrie honeypot and Docker. The knowledge gained from this investigation is highlighted.

*Section 5.2* explains the process of deploying the AWS EC2 instances that would be used to host the research environment, and how they were configured to facilitate the requirements of the system.

*Section 5.3* details the configuration of the core component of the proposed system: The Docker honeynet. Details of the initial implementation of the containers and their inter-networking are described, before discussing a major challenge that was encountered in this process. A re-evaluated design and implementation is illustrated in detail, before

finally explaining how the entire system was consolidated into a single deployable unit.

*Section 5.4* explains the installation and configuration of the log processing and visualisation tools used, including how the honeypot data was provided to this system.

*Section 5.5* describes the configuration of PSAD, the alert generation system identified in the design phase as the tool of choice for notifying administrators of threats to their system.

Finally, *Section 5.6* provides a brief summary of what was achieved through the implementation of the system.

## 5.1 Understanding Containerised Honeypots

In order to first understand the capabilities of AWS EC2, the Cowrie honeypot and Docker, some initial exploration of how these components work in practice formed a useful starting point. An AWS EC2 instance was first deployed, followed by a full installation of a Cowrie honeypot and finally some experimentation with Docker and its components.

### 5.1.1 Deploying an AWS EC2 Instance

A single AWS EC2 instance was provisioned under the AWS 12-month free tier<sup>1</sup> to facilitate exploring the Cowrie honeypot and the use of Docker containers.

#### 5.1.1.1 Provisioning an Instance

Provisioning an EC2 instance on the free-tier is a straight-forward process, which involves creating an AWS account and following an EC2 deployment setup wizard, selecting the system specifications required. In this case, an Ubuntu 16.04 LTS server instance was provisioned with the same resource allocations as those provided with the free tier.

---

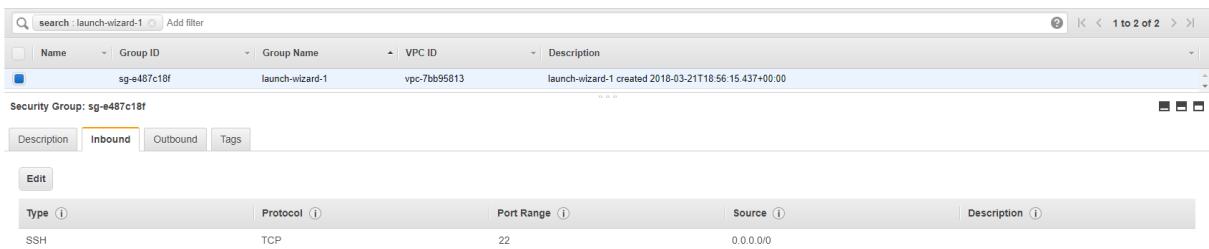
<sup>1</sup>The AWS free tier is an offer that allows certain AWS services to be used up to a limit for a defined period. For the EC2 service, the free-tier includes 750 hours of compute time on an EC2 server instance with 1 virtual CPU and 2GB of RAM for up to 12 months.

### 5.1.1.2 Key Authentication

As part of the deployment of an EC2 instance, an SSH public key<sup>2</sup> is generated. This key can then be used to securely authenticate with the instance over SSH in place of password authentication.

### 5.1.1.3 Security Groups

AWS EC2 uses *security groups* to control the inbound and outbound traffic to an instance, and every EC2 instance must have at least one security group associated with it. Security groups act as a virtual firewall, with a set of security rules that define address ranges, protocols and ports that are permitted for communication with that instance. The default EC2 security group allows for SSH access over port 22/TCP from all IPv4 addresses. This default security group can be seen in figure 5.1.



The screenshot shows the AWS EC2 Security Groups console. At the top, there is a search bar with the placeholder 'search : launch-wizard-1' and a 'Add filter' button. Below the search bar is a table header with columns: Name, Group ID, Group Name, VPC ID, and Description. A single row is visible in the table, representing the 'sg-e487c18f' security group. The details are: Group Name: 'launch-wizard-1', VPC ID: 'vpc-7bb95813', and Description: 'launch-wizard-1 created 2018-03-21T18:56:15.437+00:00'. Below the table, the text 'Security Group: sg-e487c18f' is displayed. Underneath this, there are tabs for 'Description', 'Inbound' (which is selected), 'Outbound', and 'Tags'. At the bottom of the screenshot, there is an 'Edit' button and a table showing the current inbound rule: Type: SSH, Protocol: TCP, Port Range: 22, and Source: 0.0.0.0/0. There is also a 'Description' column for the rule.

Figure 5.1: AWS EC2 Default Security Rules

The default security group associated with an AWS EC2 instance when it is launched. The only security rule that is defined for this security group is SSH access over port 22/TCP from all IPv4 addresses.

## 5.1.2 The Cowrie Honeypot: A First Look

By installing and configuring a plain Cowrie honeypot on the deployed AWS EC2 instance, it would be possible to understand the basic setup and configuration of the Cowrie honeypot before attempting to encapsulate it in a container environment. This was the

---

<sup>2</sup>The encryption used by AWS for the generation of these keys is 2048-bit SSH-2 RSA encryption.

next step undertaken in understanding the details of how the proposed system would be implemented.

### 5.1.2.1 Installing Cowrie

As discussed in *Section 2.4.1.2*, the Cowrie honeypot is an emulated of a Debian installation written in Python. A comprehensive installation guide for the Cowrie honeypot is provided on the Cowrie Github page. [76] By following this guide, it was relatively straight-forward to set up a Cowrie honeypot running as a Python *virtualenv*<sup>3</sup>. The crucial requirement was to ensure that Cowrie is run by a non-root user, since it should be difficult for an attacker to achieve privilege escalation if they manage to interface directly with the host environment. Cowrie provides a safety net in this regard: If one was to attempt to launch the honeypot as a root user, it will fail to launch and output the log message '*ERROR: You must not run cowrie as root!*'

The most significant configuration step in this process was to enable Cowrie to receive attack traffic destined for the EC2 host instance. By default, Cowrie listens for incoming traffic on ports 2222/TCP and 2223/TCP of its host, and thus for it to receive traffic from privileged ports 22/TCP (SSH) and 23/TCP (telnet) some level of traffic forwarding was required. There are two approaches to achieving this recommended in the Cowrie installation guide: [76]

#### 1. Port Forwarding

Port forwarding involves using *iptables* rules to forward incoming traffic on ports 22/TCP (SSH) and 23/TCP (telnet) to ports 2222/TCP and 2223/TCP.

#### 2. Direct Listening using Authbind

As privileged ports, listening directly to incoming traffic on 22/TCP and 23/TCP requires root-user privileges. Cowrie will not run with such privileges, and so the

---

<sup>3</sup>Python *virtualenv* is a tool for creating isolated Python environments. It is used in Cowrie to create a virtual honeypot environment. [77]

*authbind* Linux utility can be used to allow it to listen directly to traffic on these ports.

In this initial installation of the Cowrie honeypot, listening as non-root using the *authbind* utility was the approach taken.

The last step in this process is to update the configuration of the SSH daemon on the host instance so that it is still externally accessible by administrators. This is required since all incoming traffic on the default SSH port 22/TCP is now being forwarded to Cowrie, and involved a simple edit to the */etc/ssh/sshd\_config* file, changing the *Port* value from 22/TCP to a different available port number.

### 5.1.2.2 Configuring Cowrie

There are three primary components of Cowrie that can be used to customise the honeypot environment.

- ***userdb.txt***

This file is used by Cowrie to determine what username-password combinations the honeypot should accept from an attacker. The default values for a clean Cowrie installation can be seen in figure 5.2.

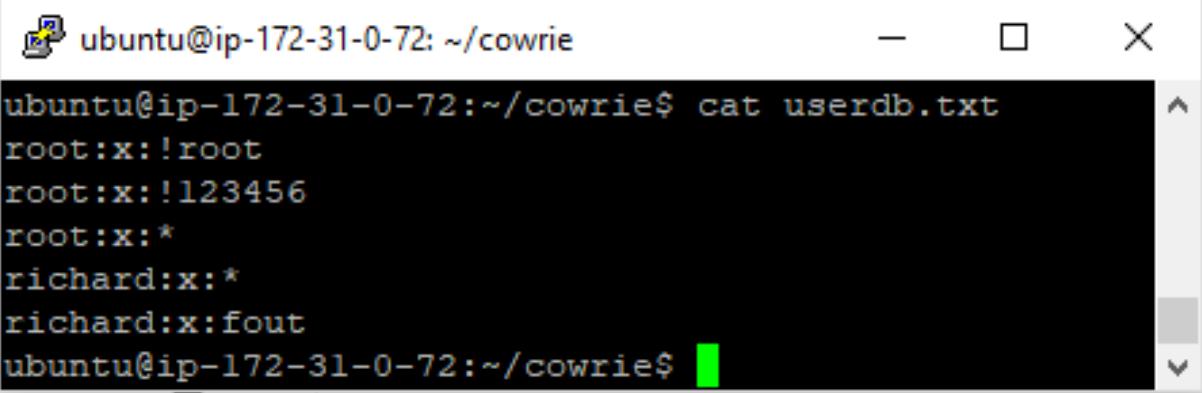
- ***cowrie.cfg***

This file is the main configuration file that is used by Cowrie to determine how to set up the honeypot environment. It comes with a multitude of settings, including parameters for integration with external tools<sup>4</sup> and other parameters to control how the honeypot environment appears to an attacker.

- ***honeyfs/***

---

<sup>4</sup>Currently, the external tool support for Cowrie listed on the Github repository README is listed as follows: Cuckoo Sandbox, ELK stack, Graylog, Kippo-Graph, Splunk, SQL (MySQL, SQLite3, Re-thinkDB). [52]



The screenshot shows a terminal window titled "ubuntu@ip-172-31-0-72: ~/cowrie". The command "cat userdb.txt" is run, displaying the following content:

```
root:x:!root
root:x:!123456
root:x:*
richard:x:*
richard:x:fout
```

Figure 5.2: Cowrie Honeypot Default Passwords

This image shows the default username-password combinations that are provided in Cowrie's *userdb.txt* file. The first entry *root:x:!root* specifies that username *root* should be accepted, but not when the password specified is also *root*. The third entry *root:x:\** specifies that username *root* should be accepted, with any other passwords for which there is no rule specified.

It is possible to configure the file system contents of the honeypot environment through the addition of files and directories to the *honeyfs/* directory.

For this initial investigation into the operation of Cowrie, the configuration of all of these components were left almost completely unaltered with default values.

### 5.1.2.3 Exposing the Honeypot to Attacks

In order to understand the logging data produced by Cowrie, the honeypot was set up and exposed to the public internet. The installed Cowrie honeypot was configured with a name corresponding to a particular model of TP-Link IP camera<sup>5</sup>, devices which are commonly targeted by IoT botnets. The default EC2 security group was left unaltered, meaning that the honeypot was exposed to the entire IPv4 address space on port 22/TCP (SSH).

The environment was exposed for 1 hour, and in that time logged brute-force authentication attacks from 4 different IP addresses. These attacks were identified to be bot

---

<sup>5</sup>The TP-Link-TL-SC3171 model was chosen, since upon conducting a brief web search this model was immediately identified as having a Remote Code Execution (RCE) vulnerability: A highly desirable target. [78]

attacks<sup>6</sup> whose source IP addresses were identified to be from multiple different countries including Egypt, Russia, China and Vietnam<sup>7</sup>.

One of the attack sessions was of particular interest, since upon comparison of the attempted login credentials it was observed that several of the passwords were the same as those identified by Antonakakis *et al.* in the source code of the Mirai botnet<sup>8</sup>. [20] This was an interesting find, and illustrates the sheer scale of the infection of devices by this botnet and its variants.

The Cowrie log, which captured the attack session data in JSON format, gave an insight into the level of detail of information that Cowrie can capture. The information captured in these sessions included:

- The source IP and port of the connection;
- The remote SSH protocol version and encryption algorithm used;
- The username-password combinations that were attempted, and whether the attempt succeeded or failed;
- How long the connection was maintained for, and the reason why the connection was terminated<sup>9</sup>;
- Timestamps corresponding to each event, and a unique session ID per connection.

As information that would later be processed by the ELK stack tools in the incident monitoring system, seeing the level of detail of JSON logging provided by Cowrie was use-

---

<sup>6</sup>Based on the timestamps logged by Cowrie, subsequent login attempts were spaced broadly 1 second apart: A speed which would be difficult for a human attacker to achieve consistently, leading to the classification of these attacks as bot attacks.

<sup>7</sup>The source location was determined by using *Maxmind GeoIP Database* which approximates the geolocation of IP addresses. [79]

<sup>8</sup>A number of the default passwords observed by Antonakakis *et al.* in their research can be seen in *Section 2.1.2.3*. Of these, passwords *admin*, *7ujMko0admin*, *1234*, *admin1234*, *default*, *password* were passwords in common with the bot attack observed here. The highly unusual password *7ujMko0admin* would appear to indicate that this attack was conducted by a device that had been compromised by Mirai or one of its variants.

<sup>9</sup>For example, this could be something like too many incorrect login attempts, in which case the message "too many bad auths" would be logged by Cowrie.

ful towards understanding what visual data could potentially be generated from honeypot logs.

### 5.1.3 Building Docker Containers

The Docker ecosystem is straight-forward in principle, but is composed of a number of distinct components that together enable the stateless hosting of applications in containers. Understanding the components of this ecosystem and how they interoperate would be essential to effectively using this technology to implement the honeynet.

The basic Docker-Cowrie source code provided by the Cowrie Project [55] was used to understand the construction of container environments and how to interoperate them before progressing to the implementation of the containerised honeynet.

#### 5.1.3.1 Docker Images

A Docker container environment is completely specified by its image definition. This image definition can be defined using a *Dockerfile*. The Dockerfile defines what the environment inside a container looks like, and consists of a set of text commands that are used to build the image<sup>10</sup>. Any external files referred to within the Dockerfile form part of the *build context* of the image.

The lifecycle of a container is often referred to as the *build-and-run* cycle, illustrated in figure 5.3<sup>11</sup>.

##### 1. Building a Docker Image

By issuing the *docker build* CLI command with reference to a Dockerfile, a container image is generated by the Docker engine using the Dockerfile and the build context. The instructions in the Dockerfile are executed one-by-one and committed

---

<sup>10</sup>The full set of commands that can be specified in a Dockerfile can be found in the Docker documentation. [80]

<sup>11</sup>This illustration is adapted from a diagram in a presentation given by the authors of the paper *Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research*. [66] [81].

incrementally to the new image before finally generating an ID for the image. The new image usually uses a *base image* as a foundation, which is a top-level image upon which other images can be built.

## 2. Running a Docker Container

If the *docker create [arguments] [image-ID]* CLI command is issued with reference to an image ID, a writeable container layer is created based on that image and is prepared for running using any arguments provided. The resulting container layer is tagged with a container ID. However, the container is not automatically launched after being created: Issuing a *docker start [container-ID]* CLI command will launch the container.

If instead the *docker run [arguments] [image-ID]* command is issued with reference to the image ID, the same writeable container layer is created and the container is immediately launched.

## 3. Stopping or Removing a Container

After being launched, a container can be stopped using the *docker stop [container-ID]* command. The container can also be removed using the *docker container rm [container-ID]* command, but only if either (i) it is not running, or (ii) it is running but the removal is forced by specifying additional arguments.

### 5.1.3.2 Entrypoint Scripts

Different elements of a container environment must be configured at particular points in the container lifecycle: Certain configuration options can be specified pre-runtime, whilst others must be specified at runtime. For example, one cannot update a container's internal routing table until that container is actually running, and hence this configuration cannot be directly specified as part of the static Docker image. This seems a nuisance since it

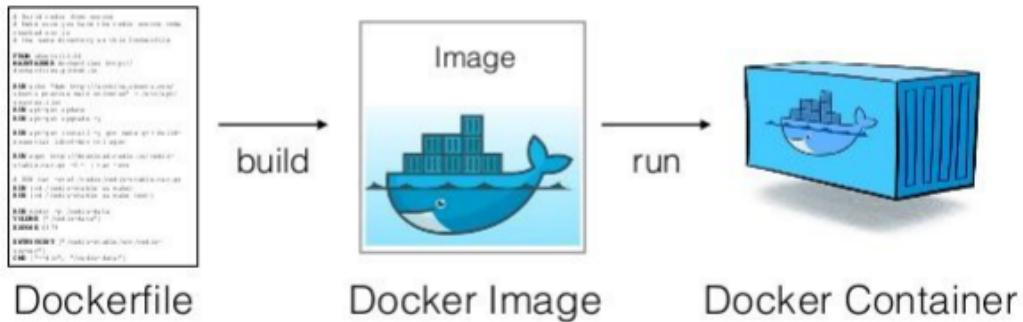


Figure 5.3: The Docker *build-and-run* Cycle

An adapted illustration of the Docker *build-and-run* cycle. The Dockerfile is used to build a Docker image, which in turn is used to run a Docker container.

means that not all environment configuration can be directly defined within the container image, but it is here that the use of the Docker *ENTRYPOINT* command becomes useful.

The *ENTRYPOINT* command can be used in a Dockerfile to specify a script which should be run when a container using that image is launched. This means that in the previous example, a script defining updates to the container's internal routing table can be specified as the *ENTRYPOINT* command in the Dockerfile. After launching the container, this script will be executed and the routing table will be updated. This is a useful and flexible option in the configuration of a container environment.

### 5.1.3.3 Docker Volumes

Docker volumes were discussed in *Section 4.2.3* as a means of persisting volatile content generated by a container. They can be mounted by using the *VOLUME* command in the Dockerfile for a container image, and the corresponding host directory to which they are mapped is specified as an argument to the *docker create* or *docker run* CLI commands at a later stage. Once a container is stopped, any volumes that are mounted to it and their contents are persisted: It is only when the container is completely removed that the volume is also removed.

Figure 5.4 shows a conceptual view of how volumes can be used to persist content generated within a container, on the host system.

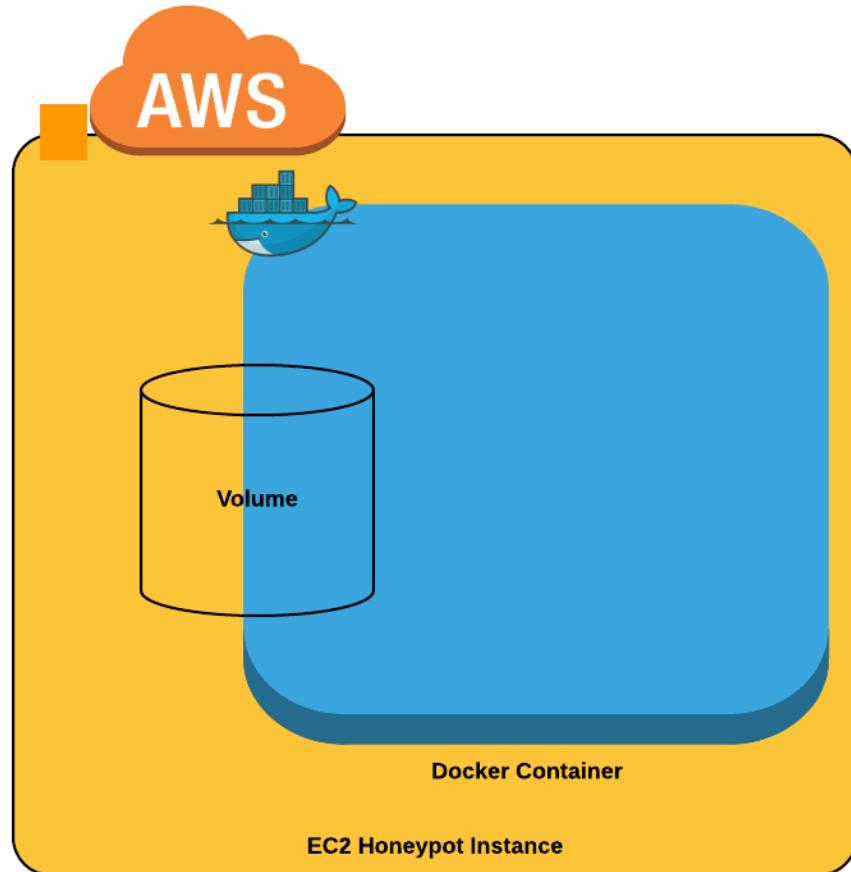


Figure 5.4: A Conceptual Representation of Docker Volumes

A conceptual view of how Docker volumes operate. Files in this volume are accessible both from the host and the container in a specific directory on each system. Changes can be made dynamically to the contents of either directory and are immediately available in the corresponding mapped directory.

#### 5.1.3.4 Docker Networking

Containers, like a physical machine, have their own networking stack. Each container has its own ports, and can have virtual interfaces connecting them to virtual Docker networks. This is the key piece of functionality that makes Docker so well-suited to the implementation of the honeynet in the proposed system.

There are two different ways of exposing ports through the *docker create* and *docker*

*run* CLI commands, which have different effects.

- Using the *-expose* argument exposes the specified container port to the host machine and to other containers. It is not however published to the host's network interfaces.
- On the other hand, using the *-publish* argument publishes the specified container port to the host's network interfaces meaning that it is accessible through these ports on the public internet.

An *EXPOSE* argument can also be specified in the Dockerfile of an image, meaning that when a container is built from that image the specified ports are exposed for *inter-container communication* only. This is the default option specified in the Docker-Cowrie Dockerfile built by the Cowrie Project developers. [55]

All containers, unless a network is otherwise specified, are automatically connected to a default Docker bridge network<sup>12</sup> called *bridge* once created. This network allows the containers to communicate directly with each other and with the host machine through a virtual network interface called *docker0*. Additional networks can be manually defined as bridge or overlay<sup>13</sup> networks.

#### 5.1.4 Conclusions

Having explored the operation of AWS EC2, the Cowrie honeypot and the Docker container ecosystem, a solid understanding and some basic configuration had been achieved regarding the next steps in the implementation of the research environment described in *Chapter 4*. In particular, by the time all of the aforementioned Docker functionalities had been explored a rudimentary deployment script named *do\_honeypot.sh* had been developed for the rapid removal and deployment of containers. This script could, based on

---

<sup>12</sup>As explained in the Docker documentation, [82] "a bridge network is a Link Layer device which forwards traffic between network segments... In terms of Docker, a bridge network uses a software bridge which allows containers connected to the same bridge network to communicate."

<sup>13</sup>An overlay network is described in the Docker documentation as "a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it ... to communicate securely. [83]

simple parameters, perform functions including:

- Building the basic Docker-Cowrie image;
- Creating a container based on the pre-built image, given a unique container name;
- Launching and stopping a container given its name;
- Entering a running container, presenting an internal container CLI to the user;
- Providing usage instructions on how to use the script to manage containers on the system.

This, coupled with the knowledge gained about the operation of Cowrie and AWS EC2, formed a solid starting point from which to begin implementing the research environment.

## 5.2 Host Server Deployment

There were two AWS EC2 server instances deployed for the implementation of the system: A honeypot server instance and a management server instance, as designed in *Chapter 4*. The same deployment process described in *Section 5.1.1* was followed. Both instances were hosted in the Amazon US East (Ohio) availability zone<sup>14</sup> running Ubuntu 16.04 LTS operating systems. Each instance was allocated different hardware resources depending on their requirements. In both cases, the hardware requirements were such that those provided by the AWS 12-month free tier were not sufficient, meaning that more expensive cost-incurring instances were needed.

### 5.2.1 The EC2 Honeypot Server Instance

In order to host a number of Docker containers as part of the proposed containerised honeynet, a relatively powerful server instance was required. Based on the initial ex-

---

<sup>14</sup>As explained in the AWS EC2 documentation, "Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of regions and Availability Zones. Each region is a separate geographic area. Each region has multiple, isolated locations known as Availability Zones." [84]

perimentation with running Docker containers on the free-tier instance, an EC2 instance with 16GB RAM and 4 virtual CPUs was provisioned in order to allow multiple Docker containers to be hosted<sup>15</sup>.

To be able to attract the attention of IoT bots scanning the public internet for vulnerable devices, the SSH (22/TCP) and telnet (23/TCP) default ports needed to be visible on this server instance. In total, just these two ports were exposed to the entire public internet by configuring security rules<sup>16,17</sup>.

One other port was opened on the server instance but with greater access restrictions. As described in *Section 5.1.2.1*, the purpose of this port is to ensure that administrative access to this server instance would be maintained. The port was chosen at random as 45678/TCP, and was exposed only to the IP address of the management instance to obscure the fact that this port was open as much as possible.

All security rules configured for the honeypot server instance can be seen in figure 5.5.

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	SSH attack access
SSH	TCP	22	::/0	SSH attack access
Custom TCP Rule	TCP	23	0.0.0.0/0	Telnet attack acce...
Custom TCP Rule	TCP	23	::/0	Telnet attack acce...
Custom TCP Rule	TCP	45678	172.31.216/32	Admin SSH access

Figure 5.5: AWS EC2 Honeypot Server Instance Security Rules

The 'Cowrie Instance Rules' security group, showing the rules defined for access to the AWS EC2 honeypot server instance. For each of SSH (22/TCP) and telnet (23/TCP), these ports were exposed to the entire IPv4 and IPv6 address spaces, whilst port 45678/TCP was only exposed to the IP address of the management server instance.

<sup>15</sup>The free-tier specifications of 2GB RAM were determined to be insufficient for hosting multiple Docker containers, since it was found that multiple Docker containers launched simultaneously would cease to run due to memory shortages.

<sup>16</sup>It was deemed unnecessary to allow access on other ports, since for the purpose of collecting data for experiments the activities of interest were limited to attacks over SSH and telnet as are typical of IoT botnets.

<sup>17</sup>During the initial development of the system, access to these two ports was restricted through the AWS EC2 security rules to a select IP address for testing purposes.

### 5.2.2 The EC2 Management Server Instance

Since the management server instance was required to host the ELK log processing stack, there were certain minimum hardware resources required to be able to run these tools. Due to cost considerations, a minimal set of resources were allocated to meet the requirements: A server with 4GB of RAM and 2 virtual CPUs<sup>18</sup>. The installation and configuration of the tools to be hosted on this EC2 instance are described later in *Section 5.4*.

On this server instance, there were three ports which were exposed using security rules:

- Port 22/TCP for SSH administrator access;
- Port 5045/TCP, which was a port randomly chosen to allow the honeypot server instance to transfer logs to the management server instance;
- Port 80/TCP, to allow HTTP access from a web browser to view the visualisations that would later be available from this server.

All security rules configured for the management server instance can be seen in figure 5.6.

## 5.3 Building the Docker Honeynet

One of the initial considerations that was described in *Section 4.1.1.7* was that the Docker honeynet should be deployable automatically by recording all commands required to build the networked environment in bash scripts. Throughout the development of the Docker honeynet described in this section, the *do\_honeypot.sh* deployment script described in *Section 5.1.4* was incrementally expanded to include multiple functions and verbose output messages for easy deployment, removal and maintenance of the honeynet.

---

<sup>18</sup>The default RAM allocated to Elasticsearch, which is more resource-hungry than Logstash and Kibana, is 1GB. [85] Upon the first installation of Elasticsearch on a server with 2GB RAM, it was found that this memory allocation was insufficient since Elasticsearch would continuously run out of heap memory. In a comprehensive tutorial by DigitalOcean which explains the steps required to configure these tools on an Ubuntu installation, 4GB of RAM was deemed sufficient. [86] This led to the eventual decision to allocate a server with 4GB of RAM.

Name	Group ID	Group Name	VPC ID	Description
sg-7c20f417	Management Server	vpc-7bb95813		Firewall rules for ELK, alerting, etc. Includes SSH rules.

Security Group: sg-7c20f417				
<input type="button" value="Description"/> <input checked="" type="button" value="Inbound"/> <input type="button" value="Outbound"/> <input type="button" value="Tags"/>				
<input type="button" value="Edit"/>				
Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	134.226.36.90/32	Browser Access
Custom TCP Rule	TCP	5045	172.31.0.72/32	Filebeat/Logstash ...
SSH	TCP	22	134.226.44.0/24	SSH access

Figure 5.6: AWS EC2 Management Server Instance Security Rules

The 'Management Server' security group, showing the rules defined for access to the AWS EC2 management server instance. Port 80/TCP here is seen to be exposed to a single IP address, from which access to the visualisations generated could be accessed via a web session. Port 5045/TCP is exposed to the IP address of the honeypot server instance to allow the Filebeat log-shipping application to transfer encrypted log files to the management server instance. Port 22/TCP is also exposed to a single IP address to allow administrative access to the instance.

### 5.3.1 The Cowrie Container

It was decided in the design phase that the Docker image for the Cowrie honeypot containers would be heavily based off the existing image developed by the maintainers of the Cowrie Project, which had already been explored in the experimentation phase. [55] This image definition provides a sound basis for building an enhanced Cowrie container image which can facilitate the inter-networking of multiple containers in a honeynet configuration.

#### 5.3.1.1 Building an Enhanced Docker Image

The Dockerfile developed by the maintainers of the Cowrie Project performs the following operations when being built into an image:

- The *debian:jessie-slim* base OS image is downloaded and used as a foundation for the container image.
- All dependencies required by Cowrie are installed, most of which are build dependencies.
- The Cowrie application is configured in accordance with the mandatory installation

steps specified on the Cowrie Github repository, [76] before removing all unnecessary build packages.

- A non-root user *cowrie* is created, and set as the default user for the container.

The Dockerfile also specifies a command to launch Cowrie at runtime under the *cowrie* user, as well as specifying a working directory */cowrie/cowrie-git/*, mounting two Docker volumes and exposing ports 2222/TCP and 2223/TCP, the default host ports on which Cowrie listens for incoming traffic.

The creation of a default non-root user and removal of unnecessary build packages make the container environment suitably restrictive in the event that an attacker was able to bypass the Cowrie application and interface directly with the underlying container. Additional customisations were added to this base image definition to facilitate functions required by the Cowrie honeynet.

### 5.3.1.2 Additional Linux Utilities

The Cowrie application is capable of emulating many common Linux utilities including *cat*, *wget*, *echo* and many others. As an open-source project it is even possible to add new utility emulations if desired. Thus, there are very few packages that actually need to be installed on the Cowrie container, and those that are required are mostly build dependencies. [55] The only additional dependency required was *authbind*, to allow Cowrie to receive incoming attack traffic from its host container.

In the customised Cowrie Dockerfile, an *entrypoint script* was specified to be executed at container runtime. This script was used to configure *authbind* as in *Section 5.1.2.1* so that Cowrie could listen to traffic on ports 22/TCP (SSH) and 23/TCP (telnet) of the container. The script is shown in Snippet 5.1.

```

#!/bin/bash

# Allow cowrie to listen as non-root on ports 22 and 23
touch /etc/authbind/byport/22 && \
    touch /etc/authbind/byport/23 && \
        chown cowrie:cowrie /etc/authbind/byport/22 && \
            chown cowrie:cowrie /etc/authbind/byport/23 && \
                chmod 777 /etc/authbind/byport/22 && \
                    chmod 777 /etc/authbind/byport/23

# Tell Cowrie that authbind is being used
sed -i 's/AUTHBIND_ENABLED=no/AUTHBIND_ENABLED=yes/' \
    /cowrie/cowrie-git/bin/cowrie

## Start the cowrie service
su - cowrie -c '/cowrie/cowrie-git/bin/cowrie start -n'

```

Listing 5.1: The entrypoint script specified in the customised Cowrie Dockerfile. This shows the configuration of *authbind* by creating 2 new empty files using *touch*, the assignment of ownership of these files to the *cowrie* user, and full read-write-execute permissions being given to *cowrie* user for both of these files. This allows the *cowrie* user to use *authbind* to receive incoming traffic on these ports. Cowrie then needs to be notified that *authbind* is being used, achieved by setting the *AUTHBIND\_ENABLED* configuration parameter. Finally, the script launches the Cowrie honeypot under the *cowrie* user.

### 5.3.1.3 Sharing and Persisting Container Data

As identified in *Section 4.2.3* and explored in *Section 5.1.3.3*, volumes are useful for dynamically sharing information between a container and its host. The use of volumes in the Cowrie containers in this system serves two purposes:

1. Cowrie configuration files can be specified on the host, and accessed by a container through a shared volume;
2. Logs generated inside a container by Cowrie can be accessed by the host through a shared volume, making them both accessible and persistent outside the container.

Thus, in the honeynet deployment script an extended function was defined for the creation of Cowrie containers, specifying a number of volumes to be mounted to them.

- The *dl/* directory in the Cowrie container was mounted as a volume to allow container access from the host. This is the default location where checksums of attempted downloads by attackers are stored.
- The *log/* directory in the Cowrie container was mounted as a volume to allow container access from the host. This is the default location where log files generated by Cowrie are stored.
- The *data/* directory in the Cowrie container was also mounted as a volume to allow host access from the container. This is the default location for the *userdb.txt* password configuration file used by Cowrie.

As well as using volumes to make configuration files available inside Cowrie containers, the *docker cp* command was used to copy the *cowrie.cfg* file from the host into the container without using a volume<sup>19</sup>.

---

<sup>19</sup>This was necessary because of an intricacy in the mounting of nested volumes from the host to a container: If the directory containing this file was mounted as a volume to the container, an issue would arise where the volumes corresponding to the nested */data*, */log* and */dl* directories would be overwritten.

#### 5.3.1.4 Port Publication

Because all containers need to be able to communicate with each other over ports 22/TCP and 23/TCP to enable the desired attack propagation between honeypots in the honeynet, the *EXPOSE* command was used in the Cowrie Dockerfile. This meant that when launched, ports 22/TCP and 23/TCP on any Cowrie container would be visible to any other containers on the default Docker *bridge* network.

In the case of the honeywall Cowrie container, the situation was slightly different: In order to be able to receive attacks, it needed to be able to access incoming traffic from the network interface of the host. To achieve this, when creating the honeywall container the *-publish* argument was used to map ports 22/TCP and 23/TCP on the host network interface to the corresponding ports on the container. The use of the *-publish* argument also meant that the container would automatically be assigned a virtual network interface on the *bridge* Docker network, such that it would be able to receive the incoming network traffic on these ports from the host instance. Figure 5.7 illustrates how this looks conceptually.

A separate function was added to the *do\_honeypot.sh* script to create and launch this container as distinct from an ordinary Cowrie container which would not be publishing these ports.

### 5.3.2 The Docker Honeynet

The Docker honeynet is one of the novel components of the system design, and would consist of new Docker network with a number of Cowrie containers connected to it. The honeywall Cowrie container would act as a gateway between the EC2 honeypot instance and this Docker network, meaning that it would need to have virtual interfaces on both the *bridge* Docker bridge network and the new Docker network hosting the other Cowrie containers.

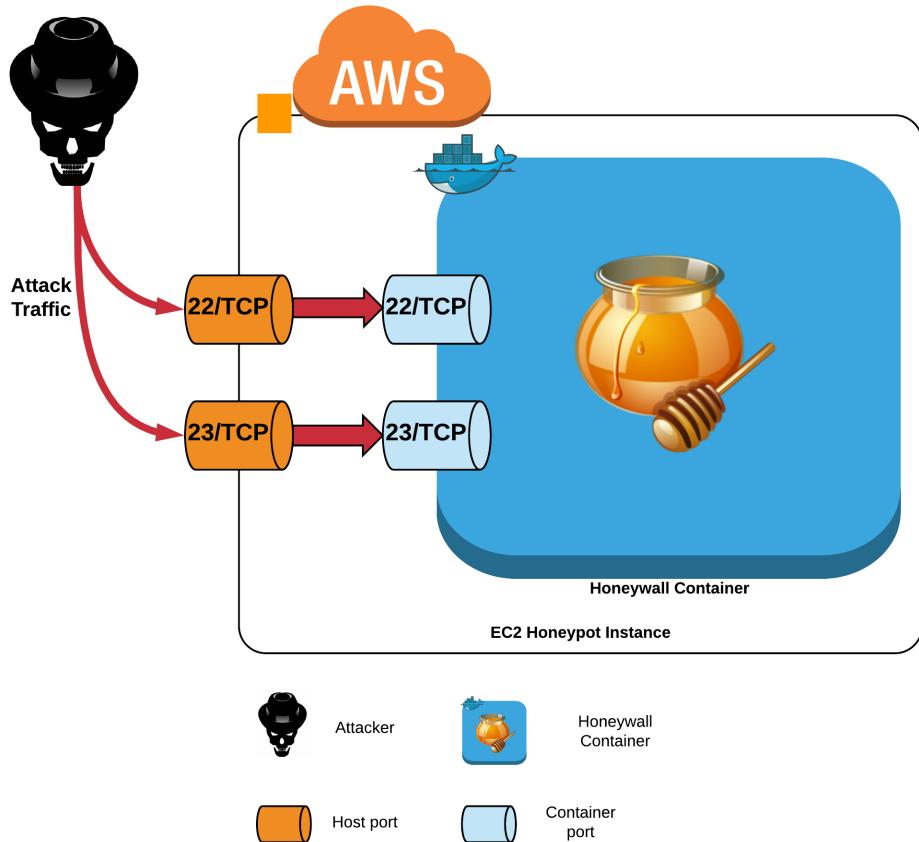


Figure 5.7: Port Publication on the Honeywall Container

An illustration of the publication of ports 22/TCP, 23/TCP on the honeywall container to ports 22/TCP, 23/TCP on the host. Any traffic received by the host on these ports is mapped directly to the corresponding ports on the container, essentially making these container ports directly available from the Internet.

### 5.3.2.1 Defining a Docker Bridge Network

The new Docker network that would host the Cowrie honeypot containers was defined as a bridge network with address 10.0.0.0/8<sup>20</sup>. This network address was chosen because it is an address reserved for use in private networks, i.e. those not directly connected to the public internet. [87] The network was given the name *dmz* corresponding to the notion of a DMZ explained in *Section 2.2.1*.

In the *do\_honeypot.sh* deployment script, a new function was added to allow a user to automatically define this network without needing to provide configuration details. This

---

<sup>20</sup>A /8 subnet can have up to 16,777,214 nodes with different IPv4 addresses. Though this scale is certainly not required here, the fact that these subnets are supported in Docker allows for many containers to belong to a single network.

function can be seen in *Snippet 5.2*.

```
# Local DMZ bridge network to which all containers are connected
create_dmz_net() {
    # Define the network
    network_exists=$( docker network ls | grep "dmz" )
    if [[ -n "$network_exists" ]]; then
        echo "DMZ network defined"
    else
        echo "Creating DMZ network (10.0.0.0/8)"
        docker network create -d bridge \
            --subnet 10.0.0.0/8 \
            --attachable \
            dmz
    fi
}
```

Listing 5.2: This script shows the commands used to create the virtual Docker bridge network, *dmz*. The script uses the *docker network ls* command to check whether the network already exists, and if not creates it. The *-d bridge* argument specifies that the Docker bridge network driver should be used to create the network, and the *attachable* argument indicates that containers can be added to the network manually on-the-fly.

### 5.3.2.2 Connecting Containers to the Honeynet

As discussed in *Section 4.2.4*, ensuring that the honeypot containers did not have any unnecessary networking capabilities needed to be considered in the implementation. By specifying that Cowrie containers should be connected to the *dmz* network as part of the *do\_honeypot.sh* deployment script, they would not be added to any other networks by the

Docker engine, ensuring this.

Connecting these containers to the *dmz* network involved adding a single extra argument to the *docker create* CLI commands for the Cowrie and honeywall containers in the *do\_honeypot.sh* script: *-network "dmz"*. When executed, the *docker create* command would now randomly assign an IP address on the *dmz* network to the Cowrie containers, and a specific IP address to the honeywall container<sup>21</sup>.

### 5.3.2.3 Network Configuration of the Honeywall Container

The function of the honeywall Cowrie container differs from that of other Cowrie containers in that it should act as a gateway between the Cowrie honeynet and the EC2 host. This requires that the honeywall Cowrie container have two virtual network interfaces: One on the default *bridge* network in order to communicate with the host instance, and the other on the *dmz* honeynet.

Configuring the honeywall container as the gateway for the *dmz* network was not as easy to achieve as it first seemed. The issue encountered was as follows:

- The IP address of the network gateway must be specified at the time of creation of the network. It is not possible to specify a network component as the gateway, i.e. the honeywall container cannot be specified as the gateway, but in theory its IP address could be.
- Clearly, a container cannot have an IP address for a network that has not already been created: Thus, a container cannot first be given an IP address and then that IP address specified as the gateway during the creation of the network.

In essence, it is not possible through Docker to the honeywall container as the gateway for the *dmz* network, since the network must already be defined with a gateway before the container can be a part of it in the first place. As an issue for which no reference

---

<sup>21</sup>The address 10.0.0.254/8 was specified for the honeywall container, since this address is commonly configured as the address for routing devices in real-world networks and so was determined to add to the attractiveness of this container from an attackers perspective.

documentation existed, overcoming this issue took substantial troubleshooting and trial-and-error. Eventually, it was resolved by manual manipulation of the *iptables* rules and routing table configuration once the container was running.

To ensure that this configuration would always be present when this container is launched in future, a new Dockerfile was created for the honeywall container. The Dockerfile contained the same configuration options as that of the other Cowrie containers, but used a different entrypoint script at runtime that would configure the *iptables* rules and routing tables.

### 5.3.3 Design Pivot: A New High-Interaction Honeypot

It was discovered during the configuration of the honeywall container that the Cowrie honeypot does not have outbound networking capabilities such that it can make outbound SSH or telnet connections. This was something that was not fully understood about the Cowrie honeypot initially<sup>22</sup>, unearthed through implementation issues and verified by the maintainer of the Cowrie honeypot, Michel Oosterhof. In response to an email query on the subject, he outlined the following:

*"Cowrie by design doesn't do any outbound connections from within the honeypot... So if you want communication between the instances, that'll be a fairly significant programming exercise. You'll need to learn some Python/Twisted and basically implement an SSH/Telnet client inside Cowrie that can connect to the next instance. Or you'll have to create some way to bypass this and make it appear that this is what happens to the attackers."*

When this realisation was made, the original approach to implementing the honeynet required immediate re-evaluation. The ability to make outbound network connections

---

<sup>22</sup>It was understood before beginning the implementation of the system that the Cowrie honeypot *did* have this functionality, since it is known that issuing a *wget* download request inside the Cowrie honeypot downloads the requested resource and computes its SHA-checksum whilst appearing to the attacker to have failed. Such functionality would require the honeypot to make an outbound network connection, and the same approach was assumed to have been taken to the implementation of the SSH and telnet services in Cowrie.

from the honeywall container is essential so that an attacker can access the honeynet. However, the honeywall container should also be a maximally restricted environment to minimise the risk of attack propagation outside of the system. These considerations motivated a re-assessment of the system design.

### 5.3.3.1 Re-Evaluation of Initial Design

Given the time within which the research was being conducted, it would not be possible to extend the Cowrie source code to give the honeypots the ability to make outbound network connections over SSH and telnet. Instead, a decision was made to implement a new high-interaction Docker honeypot<sup>23</sup> to facilitate the propagation of attacks beyond the honeywall: A component referred to hereafter as the *router container*.

As a high-interaction honeypot, the router container faced an array of new challenges.

- Since attackers would no longer be interacting with an emulated environment inside the Docker container, real libraries and toolkits would have to be made available to encourage them to interact with the system.
- The Cowrie application gives a simple configuration option for specifying username-password combinations that should be allowed or disallowed for authentication by the honeypot. This is not so easy to replicate in a real system, where a user account can only be configured to accept either (i) a single correct password<sup>24</sup> or (ii) no password to authenticate a login.
- Attackers will always seek to gain root access to a system in order to have unrestricted control over it. Cowrie emulates this root access effectively, but this cannot be falsified in a real system and would require giving real root privileges to the

---

<sup>23</sup>It was decided to implement an entirely new high-interaction honeypot since there are very few high-interaction honeypots that exist, and none that were found to be actively maintained. In their survey of honeypot software, Nawrocki *et al.* found that the majority of honeypot solutions are low-interaction, and that when it comes to high-interaction solutions "only few exist". [37]

<sup>24</sup>This also presents an additional issue, since if only a single correct password is permitted per user the probability of an attacker actually attempting this password is low.

attacker<sup>25</sup>. This is highly risky, since there is no limitation on what actions they can then perform inside the system.

- An approach to logging network traffic and command execution would need to be identified in order to capture attack data, since these capabilities of the Cowrie honeypot would also no longer be available to this container.

The approaches taken to deal with each of these new difficulties are detailed in the subsequent sections.

### 5.3.3.2 Defining a New Container Image

Since the requirements of the router container would be significantly different to that of the highly-restricted Cowrie container, a new Docker image definition was required.

It was decided that in order to have access to a wide variety of powerful Linux utilities<sup>26</sup>, the *ubuntu:latest* Docker base OS image would be used as a foundation for the router container image. The container would also need to expose ports 22/TCP and 23/TCP as with the Cowrie containers, and so these were specified in the Dockerfile using the *EXPOSE* command.

The most generic Linux utilities that were specified as part of the Docker-Cowrie image were also included in this new image: In particular, the *apt-utils* and *build-essential* packages, which are required as basic utilities in almost all Debian-based Linux environments.

Once the basic Dockerfile was defined, the *do\_honeypot.sh* script was updated to use the new router container image instead of the old honeywall container image. The *docker create* arguments for adding the container to the *dmz* network and publishing ports 22/TCP

---

<sup>25</sup>This would be required since if attackers are not permitted to login as root, it is less probable that any attackers will ever reach the Cowrie honeynet.

<sup>26</sup>This variety would be required to facilitate an attacker in accessing whatever utilities they required in order to progress with their attack: It would be undesirable for the attacker to disconnect from this container without progressing to the Cowrie honeynet, since experimental data would then never be captured by the honeynet.

and 23/TCP to the host interface were left unchanged, since the router container would still require this configuration.

### 5.3.3.3 Configuring SSH and Telnet

Since there was now no emulation of SSH and telnet connectivity, it was necessary to install and configure utilities for these on the router container. The installation of the required packages was specified as part of the router container Dockerfile, and their runtime configuration properties specified in an entrypoint script.

#### 5.3.3.3.1 SSH

The installation of the *openssh-server* utility was specified in the Dockerfile. Then, the entrypoint script was used to update the configuration of the SSH daemon *sshd* to facilitate SSH logins to the container. Some noteworthy configurations specific to enabling easy access by attackers to the container include the following:

- The *PermitRootLogin* field was updated to allow SSH logins as user *root*;
- The *PasswordAuthentication* field was updated to allow authentication by password and not just by using an encryption key;
- The *PermitEmptyPasswords* field was updated to allow authentication with an empty password, if this was valid for a particular user.

Lastly, the entrypoint script was updated to restart so that the updated configuration would take effect.

#### 5.3.3.3.2 Telnet

The installation of the *telnet*, *openbsd-inetd* and *telnetd* package utilities were also added to the Dockerfile. A number of configuration steps were also added in the entrypoint

script to be applied at container runtime: In particular, the allocation of a number of *tty*<sup>27</sup> to allow multiple simultaneous telnet sessions to take place.

A telnet configuration file called *telnet* was also added to the container, which included a number of essential configuration options as well as some logging parameters. This required use of the Dockerfile *COPY* command to copy the file into the container's */etc/xinetd.d* directory. As with the configuration of SSH, the telnet daemon is restarted to allow the changes to take effect.

#### 5.3.3.4 Device Banner Configuration

As part of the *enumeration* phase of an attack, an attacker will often use a scanning tool such as *Nmap* or *Masscan* to fingerprint the target system. These scanning tools usually send a message probe over particular protocols with the intention of receiving an acknowledgement (ACK) from the target, allowing the attacker to deduce some properties of the system. In the case of IoT devices, being able to fingerprint the device type and model is very useful to an attacker in identifying a vulnerable target, and so the router container needed to be capable of presenting such information.

There are two primary messages that are presented to those initiating a user authentication connection with a Linux-based system: The *issue.net* and *motd*.*d* files.

- **issue.net / issue**

This file is used to set the banner that is sent in a protocol negotiation, e.g. for a client requesting to connect to the system via telnet, this would be the information that is used to greet the client prior to the login step. For SSH, this file is called *issue* whereas for telnet the file is called *issue.net*.

- **motd / legal**

This file is used to display a message to a client after they have successfully logged

---

<sup>27</sup>In Linux systems, a *tty* is a representation of a text input/output device to the system, such as a terminal session.

into the system. Often, administrators will use this to present a notice about permitted uses of the system after a user has successfully logged in. For SSH, this file is called *legal* whereas for telnet the file is called *motd*.

In the Dockerfile for the router container, commands were added to copy customised *issue.net*, *issue*, *legal* and *motd* files defined on the host into the container. Some additional updates were then made to the entrypoint script so that these banners would be presented in connection negotiations. Figure fig:router-telnet-banner shows how these banners appear to an individual attempting to gain access to the system via telnet.

```
ubuntu@ip-172-31-0-216:~$ sudo telnet ec2-18-217-52-41.us-east-2.compute.amazonaws.com
Trying 172.31.0.72...
Connected to ec2-18-217-52-41.us-east-2.compute.amazonaws.com.
Escape character is '^]'.
*****
* Copyright (C) 2008-2013 Huawei Technologies Co., Ltd.      *
* Without the owner\x27s prior written consent,            *
* no decompiling or reverse-engineering shall be allowed. *
* Notice:                                              *
*   This is a private communication system.               *
*   Unauthorized access or use may lead to prosecution. *
*****
Warning: Telnet is not a secure protocol, and it is recommended to use STelnet.

Login authentication

ce8cd9f47f78 login: root
Password:
root@ce8cd9f47f78:~# █
```

Figure 5.8: The Router Container Honeypot Telnet Banner

This screenshot shows a login attempt via Telnet to the EC2 honeypot server instance. It can be seen that a banner advertising a Huawei device is presented as part of the login prompt. It can be observed that the "attacker" proceeds to successfully log in as user *root*.

### 5.3.3.5 Privileges and Authentication

As identified in *Section 5.3.3.1*, for an attacker to interact with the system they would need to believe that they have gained root-level access to the router container. This is not something that can be falsified without the use of an emulated environment like Cowrie, and so the decision was made to allow root access by default for all user accounts on this container.

A related issue is the fact that arbitrary user-name password combinations could not be accepted in the same way on a real Linux environment as they could be with the emulated Cowrie environment.

- In order for an attacker to log in to the router container under a particular user, an account needs to exist for that user.
- As well as this, it is only possible to specify one valid password per user account.

The reality of this was that the number of username-password combinations that could be accepted for this honeypot would be greatly restricted.

The solution to these issues was to define a number of user accounts with usernames commonly observed in brute-force authentication attacks, and give each of these users root privileges by default. Since the exact username-password combinations used in the system would be determined through experiments, a single account for user *admin* was created as part of the Dockerfile.

In order to give root privileges to user accounts, they needed to be declared as *sudo* users<sup>28</sup>. This involved the installation of the *sudo* package as part of the Dockerfile, and the use of the *COPY* command in the Dockerfile to copy a pre-configured sudoers file into the container from the host. This sudoers file was configured such that it granted root-level privileges to the *admin* user for all utilities in the container.

This configuration should now allow an attacker authenticated as user *admin* to successfully carry out unrestricted activities inside the container.

### 5.3.3.6 Logging Attack Events

As with the Cowrie honeypot containers, it was essential that the router container could monitor the activities of an attacker on the system. This container is a high-interaction honeypot: A fully-fledged container with a real networking stack, toolkits and file system.

---

<sup>28</sup>A sudo user is one which has root privileges on the system, with unrestricted read, write and execute capabilities.

As described in *Section 2.2.3.2.1*, these honeypots offer the highest value in terms of capturing information about interactions with attackers since attacker activities are usually unrestricted. The most important elements of attack activities that needed to be captured in logging were the commands executed by attackers, and details of the attacker's connection session.

#### 5.3.3.6.1 Network Event Logging

The most commonly used logging utility for network events is *syslog*. Syslog is IETF standard Linux logging utility, which logs network and process events that occur on a system. It is heavily used by system administrators for system management and security auditing purposes. [88] It was identified as a straight-forward logging solution for capturing information about attacker connections to the router container, since it is capable of capturing logging information for both SSH and telnet.

In order to use syslog, the *rsyslog* package was added to the Dockerfile as part of the image installation for the router container. A custom-defined configuration file called *rsyslog.conf* was then added to the container image, and in the entrypoint script for the container the *rsyslog* utility was launched<sup>29</sup>.

In order to persist the logs generated by *rsyslog* inside the router container, a volume was mounted to the router container in the */var/log* directory as part of the function to create the router container in the *do\_honeypot.sh* script. SSH and telnet events are logged to a number of different log files within this directory, including *auth.log*, *messages*, *secure* and *syslog*.

#### 5.3.3.6.2 Keylogging

*Logkeys* is an open-source Linux keylogger. [89] As one of the only identified Linux

---

<sup>29</sup>The launching of the *rsyslog* utility was the last operation specified in the entrypoint script in order. This was to ensure that none of the environment configuration updates applied by the script were included in the log files, meaning that if an attacker was to inspect the syslogs on the container they would not see any records relating to the configuration of the environment, reducing the likelihood of them fingerprinting it as a honeypot.

keyloggers freely available and relatively recently maintained, it was originally selected as a keylogging solution for monitoring the activities of attackers inside the router container.

There were a number of significant difficulties encountered with the use of *logkeys*: Seemingly, these were due to its incompatibility with Docker.

- There was no single reference which specified all of the dependencies required to be installed in order for *logkeys* to operate. Since Docker containers take a lean approach to sharing libraries and packages with their host<sup>30</sup>, many of these dependencies had to be installed from the *apt* repositories. These dependencies were eventually identified through investigating *dumpkeys*, an ancient Linux keylogger which shares a lot of the same library dependencies. [90]
- The fact that *logkeys* depends on being able to reference a keyboard device driver on the host machine presented a number of issues to the use of keylogging. Input device drivers are stored in Linux systems in a device file, which appears as an ordinary file under the */dev/input* directory. Although a *-device* argument exists for the *docker create* CLI command which can be used to reference a device driver on the host, the keyboard input device remained inaccessible by *logkeys* inside the Docker container despite over a week of dedicated effort.

After exhausting all identified venues for implementing keylogging with *logkeys*, a number of additional options were investigated.

- The idea of using the Linux *.bash\_history* file for keylogging was explored. This file is created by the Bash utility, and captures all executed command-line inputs. This initially seemed a reasonable approach to keylogging: However, the fact that commands are not logged with timestamps would render the data largely unusable for identifying distinct attack sessions, making this an unsuitable approach.

---

<sup>30</sup>In general, Docker base images such as the Ubuntu Docker image are shipped only with the bare minimum of packages included, and require a lot of additional packages to be installed in order to run and manage applications.

- A manual keylogging approach was attempted, inspired by a recommendation on the online forum *askubuntu.com*. [91] This keylogging approach involved using pure shell command manipulation, leveraging off SSH and telnet environment variables generated as part of *syslog* network event logging. However, this approach was found to be unreliable and brittle, and so was also abandoned.

With the time constraints involved in the implementation of the system, the idea of keylogging was eventually put to one side as future work to improve the router container honeypot.

#### **5.3.3.7 Package Utilities**

As identified in the design re-evaluation in *Section 5.3.3.1*, a number of Linux utilities that an attacker would expect to find in a legitimate system needed to be made available in the container, since the emulation of these utilities provided by Cowrie were no longer available. In particular, it was important to facilitate an attacker in probing the *dmz* network in order to discover the Cowrie honeypot containers, and so providing common networking packages was a must.

The initial utilities that were added to the Dockerfile for the router container were *net-tools*, *nmap*, *traceroute*, *inetutils-ping*, *iptables* and *tcpdump*. These are all networking packages that would allow an attacker to perform actions such as viewing and manipulating the container’s routing table, scanning nearby hosts and capturing packets on the network.

#### **5.3.4 Finalising the Docker Honeynet**

Now that all required components of the Docker honeynet had been implemented, all that remained was to ensure that the network configuration was operating correctly. An attacker should be able access the router container from the public internet and from there, interact with any available Cowrie containers on the *dmz* network.

The targeted network configuration for the deployment is illustrated in figure 5.9.

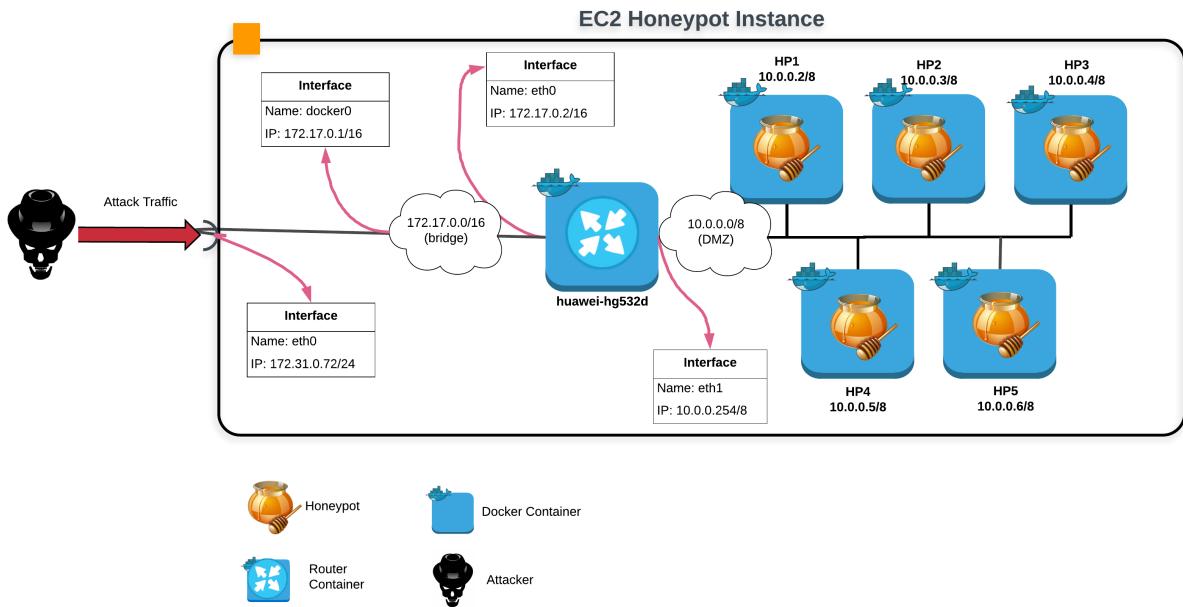


Figure 5.9: Docker Network Configuration of the EC2 Honeypot Instance

This figure illustrates the desired network configuration for the honeynet deployment on the EC2 honeypot instance. It can be observed that there are two bridge networks: *bridge* and *dmz*. The host instance is forwarding its incoming traffic on ports 22/TCP and 23/TCP to the *docker0* virtual interface on the *bridge*. This interface connects the host to the router container's virtual interface *eth0*. The router container also has a second virtual interface, *eth1* which connects it to the *dmz* network. It is through this interface that the Cowrie honeypots in the *dmz* network can be accessed.

### 5.3.4.1 The Docker *bridge* Network Configuration

For the router container to be accessible from the public internet, its configuration on the *bridge* Docker network needed to be set up to facilitate this. The first step was to inspect the configuration of the *bridge* network and the routing table on the EC2 instance itself. The desired configuration is illustrated in figure 5.10.

The internal routing table of the EC2 honeypot instance can be seen in table 5.1, showing the networks associated with its physical network interface *eth0* and Docker virtual network interface *docker0*. By design, the host cannot interface directly with the *dmz* bridge network: As discussed in *Section 4.2.4*, by preventing the host and containers from being connected to unnecessary networks the risk of these systems being compromised in

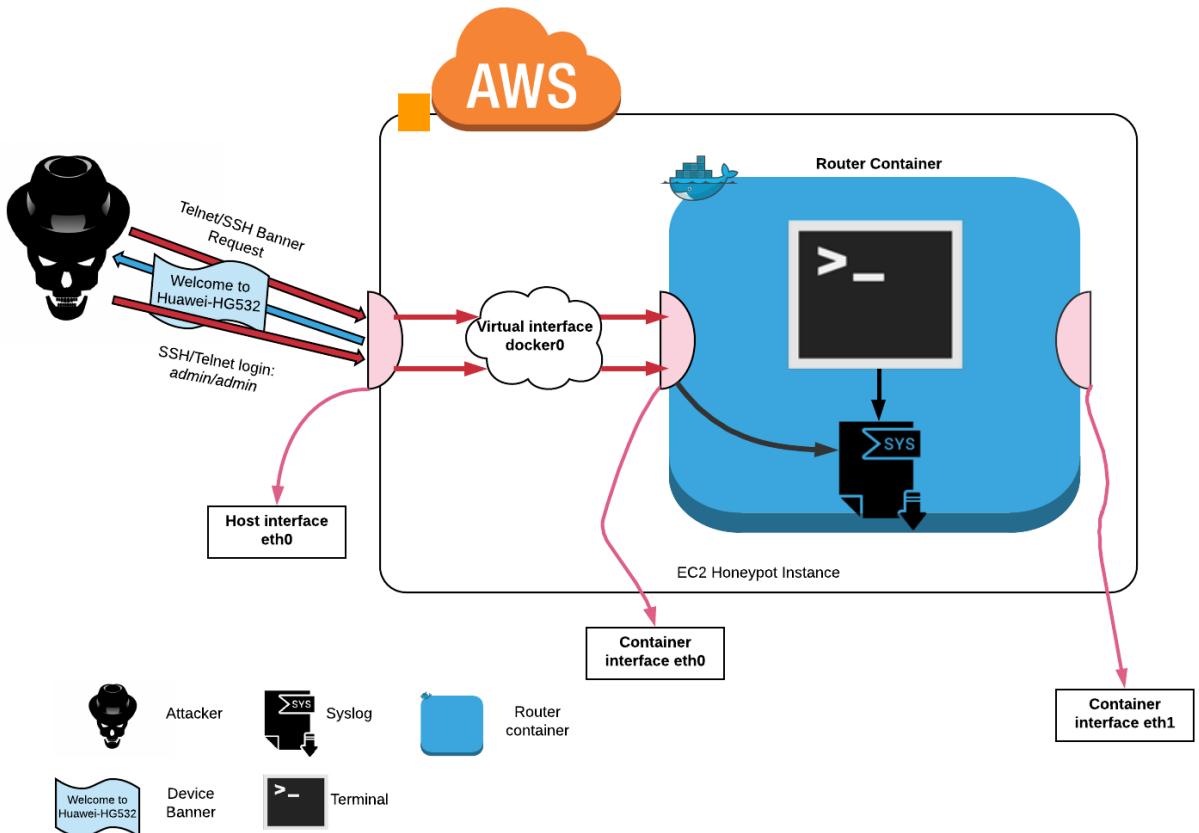


Figure 5.10: Routing of Attack Traffic to the Router Container

An illustration of the desired routing of attack traffic from the host network interface to the router container. The attacker is shown to attempt a brute-force authentication attack over either SSH or telnet, which is forwarded from the real host interface *eth0* to the *docker0* virtual interface on the *bridge* network. From this interface, the traffic is forwarded to the *eth0* router container interface. For the response to the attacker, the flow of traffic through the network is in reverse order through these interfaces.

an attack is minimised.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.31.0.1	0.0.0.0	UG	0	0	0	eth0
172.17.0.0	0.0.0.0	255.255.0.0	U	0	0	0	docker0
172.31.0.0	0.0.0.0	255.255.240.0	U	0	0	0	eth0

Table 5.1: The routing table produced by the command *route -n* executed on the honeypot instance, which hosts the Docker honeynet. Two interfaces can be observed: *eth0* and *docker0*. *eth0* connects the host to the 172.31.0.0/24 network, which is an external network managed by AWS. The *docker0* interface corresponds to a default bridge network called *bridge* with address 172.0.0.0/16, which containers are connected to by default if not specified otherwise.

By executing the command `docker network inspect bridge`, the properties of the Docker *bridge* network can be inspected. The output of executing this command can be seen in figure 5.11. It can be seen that there is only 1 container on this network, corresponding to the router container *router*. If the command `docker container inspect router` is subsequently executed, the output obtained<sup>31</sup> demonstrates the following expected properties:

- The *router* container is connected to the *bridge* network with an IP address of 172.17.0.2/16;
- The router container is also connected to the *dmz* network with an IP address of 10.0.0.254/8;
- Ports 22/TCP and 23/TCP of the *router* container are published to ports 22/TCP and 23/TCP of the host interface.

The correct routing of attack traffic through the *bridge* network could be tested from outside the EC2 honeypot server instance, by connecting over SSH or telnet to the instance and verifying that the router container login prompt is presented. Figure 5.12 shows the console output when an SSH connection is made to the EC2 honeypot server instance, demonstrating the correct routing of this traffic to the router container.

#### 5.3.4.2 The Docker *dmz* Network Configuration

To verify that attack traffic could reach the Cowrie containers in the *dmz* network, the configuration of the router container and the *dmz* network needed to be verified. An attacker, once inside the router container, should be able to connect through an interface on the router container to any Cowrie container on the *dmz* network. The desired configuration is illustrated in figure 5.13.

The internal routing table of the router container can be seen in table 5.2, showing the networks associated with the container's two virtual interfaces *eth0* and *eth1*. These

---

<sup>31</sup>The volume of output generated by this command is not feasible to include as a figure in this document, and so a description of the most important contents of the output was deemed sufficient.

```

ubuntu@ip-172-31-0-72:~$ docker network inspect bridge
[{"Name": "bridge",
 "Id": "9e4e185d5b52d3fa678358733c43580c553406bad95c73c6303760bf7828d9b7",
 "Created": "2018-04-19T03:50:06.459723552Z",
 "Scope": "local",
 "Driver": "bridge",
 "EnableIPv6": false,
 "IPAM": {
   "Driver": "default",
   "Options": null,
   "Config": [
     {
       "Subnet": "172.17.0.0/16",
       "Gateway": "172.17.0.1"
     }
   ]
 },
 "Internal": false,
 "Attachable": false,
 "Containers": {
   "57b65a1a4a2c7c2d65f8585669f7024f3f512218dabe483747523a417c7d3b02": {
     "Name": "router",
     "EndpointID": "212936f1ddcbc4a582d3211ed293c0a2af8c4b85a502dd723fb21e226647cbae",
     "MacAddress": "02:42:ac:11:00:02",
     "IPv4Address": "172.17.0.2/16",
     "IPv6Address": ""
   }
 },
 "Options": {
   "com.docker.network.bridge.default_bridge": "true",
   "com.docker.network.bridge.enable_icc": "true",
   "com.docker.network.bridge.enable_ip_masquerade": "true",
   "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
   "com.docker.network.bridge.name": "docker0",
   "com.docker.network.driver.mtu": "1500"
 },
 "Labels": {}
}
]
ubuntu@ip-172-31-0-72:~$ 

```

Figure 5.11: Console Output of *docker network inspect bridge*

An image showing the console output obtained by executing the command *docker network inspect bridge*. It can be observed that there is only 1 container on this network: The *router* container.

networks correspond to the *bridge* network 172.17.0.0/16 and the *dmz* network 10.0.0.0/8.

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	172.17.0.1	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	10.0.0.254	255.0.0.0	U	0	0	0	eth1
172.17.0.0	0.0.0.0	255.255.0.0	U	0	0	0	eth0

Table 5.2: The routing table produced by the command *route -n* executed inside the router container. Two interfaces can be observed: *eth0* and *eth1*. *eth0* connects the router container to the 172.17.0.0/16 *bridge* network, whereas *eth1* connects it to the 10.0.0.0/8 *dmz* network.

```

ubuntu@ip-172-31-0-216:~$ sudo ssh root@ec2-18-217-52-41.us-east-2.compute.amazonaws.com
Warning: the ECDSA host key for 'ec2-18-217-52-41.us-east-2.compute.amazonaws.com'
differs from the key for the IP address '172.31.0.72'
Offending key for IP in /root/.ssh/known_hosts:1
Matching host key in /root/.ssh/known_hosts:22
Are you sure you want to continue connecting (yes/no)? yes
SSH-2.0-HUAWEI-1.5
root@ec2-18-217-52-41.us-east-2.compute.amazonaws.com's password:
*****
* Notice: *
* This is a private communication system. *
* Unauthorized access or use may lead to prosecution. *
*****
root@57b65a1a4a2c:~#

```

Figure 5.12: Attempted SSH Login to the Router Container

An image showing the console output obtained when a connection attempt is made over SSH to the EC2 honeypot instance. It can be observed that a Huawei banner is presented, and that the attacker is permitted to log in as *root*.

From inside the router container, an attacker should be able to access a domain on the public internet in order to perform downloads to progress their attacks on the system. This is achieved through the configuration of Network Address Translation (NAT) rules in *iptables*. The rules shown in snippet 5.3 were used to achieve this objective, enabling an attacker to perform downloads from external domains. This temporarily caused a difficult-to-diagnose issue with the *iptables* of the EC2 instance<sup>32</sup>, which was eventually resolved by executing the command *iptables -t nat -flush* to remove some conflicting NAT rules on the host.

```

iptables --table nat --append POSTROUTING
    --out-interface eth0 -j MASQUERADE

iptables --append FORWARD --in-interface eth1 -j ACCEPT

```

Listing 5.3: The *iptables* rules that were configured to enable NAT inside the router container.

The routing table generated inside a Cowrie container can be seen in table 5.3. In this

---

<sup>32</sup>The issue manifested as an inability to connect to containers on ports 22/TCP and 23/TCP, where Nmap would show these ports as being ‘filtered’.

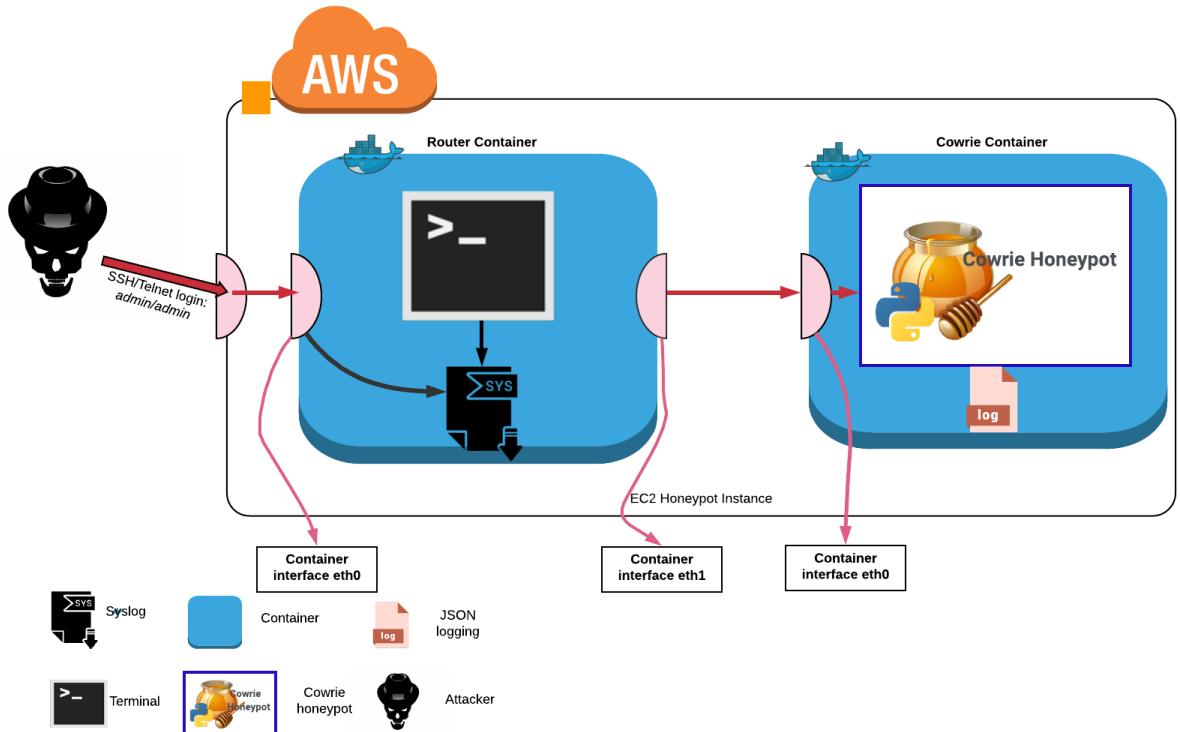


Figure 5.13: Routing of Attack Traffic to a Cowrie Container

An illustration of the desired routing of attack traffic from the host network interface to a Cowrie container. Assuming that the attacker has already successfully compromised the router container through the process in figure 5.10, they should be able to connect to the Cowrie container over SSH or telnet in the same way through the router container's *eth1* interface on the *dmz* network.

case, the container is connected to only one network: The *dmz* network, which it accesses through its virtual interface *eth0*.

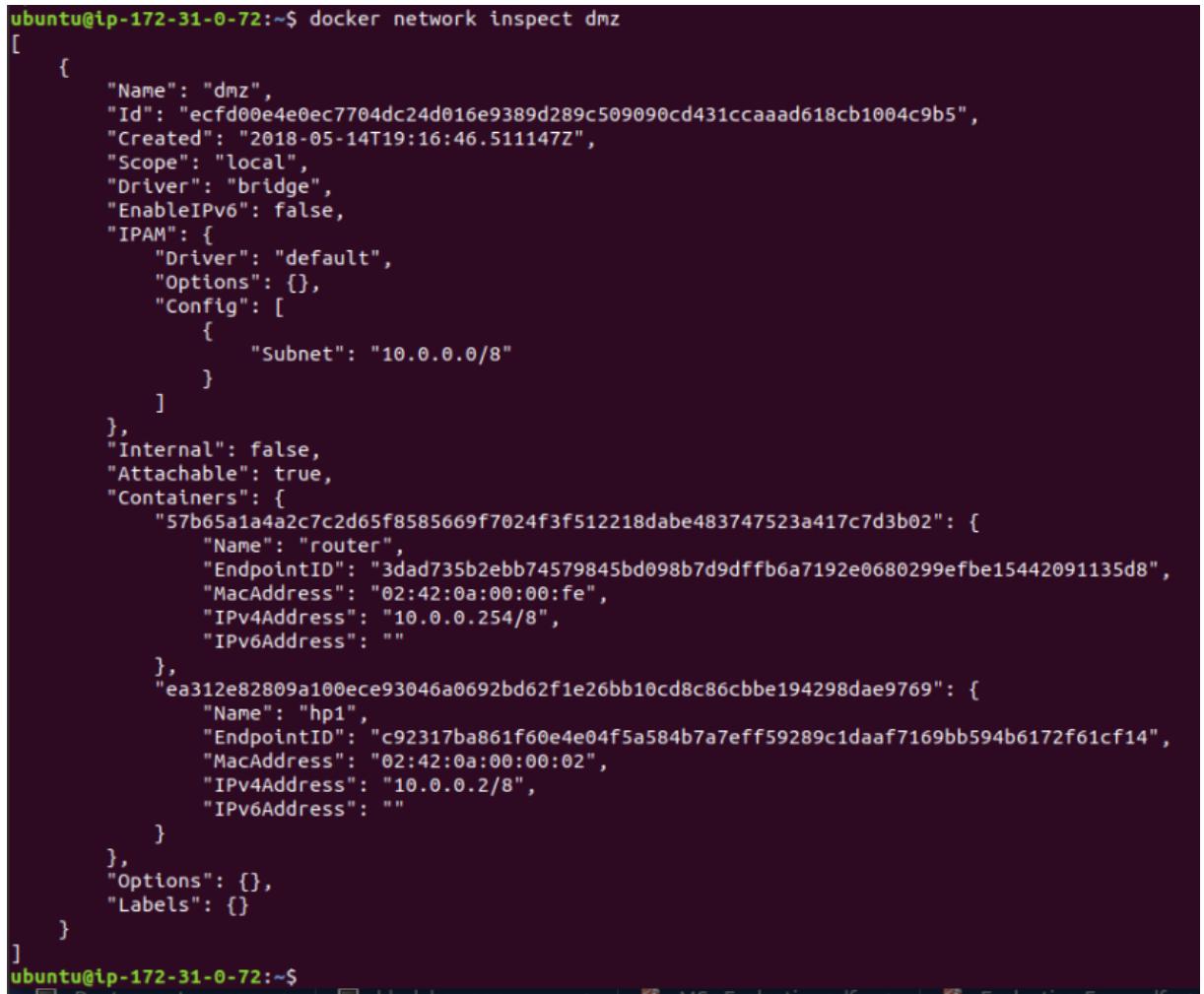
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
0.0.0.0	10.0.0.254	0.0.0.0	UG	0	0	0	eth0
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	eth0

Table 5.3: The routing table produced by the command *route -n* executed inside a Cowrie container. One interface can be observed: *eth0*. *eth0* connects the container to the 10.0.0.0/8 *dmz* network.

By executing the command *docker network inspect dmz*, the properties of the Docker *dmz* network can be inspected. The output of executing this command can be seen in figure 5.14. It can be seen that there are 2 containers on this network, corresponding to the router container *router* and a Cowrie container *hp1*. If the command *docker container*

*inspect hp1* is now executed, the following can be observed:

- The *hp1* container has an IP address of 10.0.0.2/8 on the *dmz* network;
- Ports 22/TCP and 23/TCP of the *hp1* container are exposed on the network, but have no mapping to the host.



```
ubuntu@ip-172-31-0-72:~$ docker network inspect dmz
[{"Name": "dmz", "Id": "ecfd00e4e0ec7704dc24d016e9389d289c509090cd431ccaaad618cb1004c9b5", "Created": "2018-05-14T19:16:46.511147Z", "Scope": "local", "Driver": "bridge", "EnableIPv6": false, "IPAM": {"Driver": "default", "Options": {}, "Config": [{"Subnet": "10.0.0.0/8"}]}, "Internal": false, "Attachable": true, "Containers": {"57b65a1a4a2c7c2d65f8585669f7024f3f512218dabe483747523a417c7d3b02": {"Name": "router", "EndpointID": "3dad735b2ebb74579845bd098b7d9dffba7192e0680299efbe15442091135d8", "MacAddress": "02:42:0a:00:00:fe", "IPv4Address": "10.0.0.254/8", "IPv6Address": ""}, "ea312e82809a100ece93046a0692bd62f1e26bb10cd8c86cbbe194298dae9769": {"Name": "hp1", "EndpointID": "c92317ba861f60e4e04f5a584b7a7eff59289c1daaf7169bb594b6172f61cf14", "MacAddress": "02:42:0a:00:00:02", "IPv4Address": "10.0.0.2/8", "IPv6Address": ""}}, "Options": {}, "Labels": {}}], ubuntu@ip-172-31-0-72:~$
```

Figure 5.14: Console Output of *docker network inspect dmz*

An image showing the console output obtained by executing the command *docker network inspect dmz*. It can be observed that there are 2 containers on this network: The *router* container, and the *hp1* Cowrie container.

The correct routing of attack traffic from the *router* container through the *dmz* to the *hp1* Cowrie container could be tested from inside the router container. Once inside the *router* container, connecting over SSH or telnet to the *hp1* honeypot and being presented

with the Cowrie login prompt verifies the correct configuration of the network. Figure 5.15 shows the console output when an SSH connection is attempted from the *router* container to the *hp1* container, demonstrating the correct routing of traffic within the *dmz* network.

```
root@65960460d0e1:/# ssh admin@10.0.0.2
Password:
Password:
Password:
admin@10.0.0.2's password:
Permission denied, please try again.
admin@10.0.0.2's password:

root@65960460d0e1:/#
```

Figure 5.15: A Denied SSH Login from the Router to a Cowrie Honeypot  
This screenshot shows a simulated attack session from the *router* container to the *hp1* Cowrie honeypot container. It can be observed that the "attacker" attempts to log in to the *hp1* container as user *admin*. The first password provided for login is rejected, and a second prompt for the correct password is requested by the *hp1* Cowrie honeypot. In this case, the attacker decides to cancel the login attempt rather than proceeding with a second login attempt.

## 5.4 Visualisation of Attack Data

As decided in *Section 4.1.2.3* as part of the design process, the data generated by honeypots in the Docker honeynet would be visualised using the ELK log processing stack on a separate EC2 server instance. The deployment of the EC2 management server instance was already described in *Section 5.2.2*, and this section details the installation and configuration of the tools on this instance to enable the processing and visualisation of the honeypot data.

### 5.4.1 Management Server Instance

The EC2 management server instance had already been deployed with sufficient hardware resources to meet the requirements of the ELK stack. Since the ELK stack is widely used for log processing and visualisation, there were plentiful resources available from which to learn, showing how to configure these open-source tools to work together.

Some complementary tools were also used in order to facilitate the use of the ELK stack for analysis of logs from a remote machine: In particular, Nginx and Filebeat. The configuration and interoperation of all of these tools is described in the following subsections.

#### 5.4.1.1 Elasticsearch

Elasticsearch is an open-source search and analytics engine built on the Apache Lucene API. The fact that it supports analysis of JSON-formatted files made it an ideal choice for the analysis and indexing of the Cowrie logs, which are logged in JSON format.

To set up the environment to run Elasticsearch, Java 8 was first required as a prerequisite installation. Once Java 8 has been installed, Elasticsearch was then also installed. Next, in order to configure Elasticsearch securely external access to the application needed to be restricted. This is important, since Elasticsearch provides a HTTP API that can be queried to perform manipulation on the data stored in the Elasticsearch index. If access to this was not restricted, the data in the index could be modified or removed without authorisation.

By editing the `/etc/elasticsearch/elasticsearch.yml` configuration file for Elasticsearch, external access was completely restricted by setting the access address `network.host` to `localhost`<sup>33</sup>. This meant that Elasticsearch could only be accessed from the management instance itself on localhost, on the Elasticsearch default port 9200/TCP.

---

<sup>33</sup>The IP address 127.0.0.1 is known as the localhost address.

#### **5.4.1.2 Kibana**

The Kibana visualisation engine was the second component of the ELK stack to be installed and configured. Kibana is an open-source analytics and visualisation platform, designed specifically to work with the Elasticsearch search platform. The use of Kibana in this project was intended to make analysis of the honeypot log files painless and immediate.

Kibana provides a web interface through which visualisations can be accessed. By configuring rules to visualise incoming log data, Kibana makes it possible to understand crucial information about the state of the system instantaneously.

Similarly to Elasticsearch, it was important to set up Kibana to have restricted accessibility from external systems. In order to achieve this whilst still allowing an administrator to access the visualisations through the web interface, the Nginx reverse proxying tool was used.

#### **5.4.1.3 Nginx**

Nginx is an open-source tool focused on providing services like web serving, reverse proxying, caching, and load balancing. [92] In this system, it was used to allow access to the Kibana web interface using reverse-proxying.

An authentication step was configured with a single valid username-password credential to allow for authenticated login from a web browser. A small amount of additional configuration was then required to direct incoming HTTP traffic to the Kibana application<sup>34</sup>.

#### **5.4.1.4 Logstash**

Logstash is an open-source log pipeline tool which accepts, processes, transforms and outputs data from log files provided to it, and was the final component configured on the

---

<sup>34</sup>This involved the addition of the EC2 management server instance's public IP address to a Nginx configuration file in the *sites-available* directory.

management server instance.

As discussed in *Section 4.1.2.2*, an SSL certificate was generated by using the private IP of the server in the SAN field. This would later be used by Logstash to verify the origin of log data shipped by Filebeat.

In order to configure the processing components of Logstash, a number of configurations needed to be defined to deal with each of the incoming log formats: Input, filter and output configurations. These were all specified as part of the same file, *03-cowrie.conf*.

- **Input**

Inputs are a field used to pass data into the Logstash pipeline. In this system, a single input was defined to accept data of type *beats*, corresponding to data shipped by Filebeat.

A port number was specified from which this data should be received by Filebeat. The path to the generated SSL certificate was also specified, so that the data received on this port could be verified by Logstash.

- **Filter**

Filters are an intermediary processing step in the Logstash pipeline, where operations can be applied to the input data. An excellent tutorial was identified which explained a number of different filtering operations that could be applied to the Cowrie logs. These formed the basis of the filtering operations configured in this field<sup>35</sup>. [93]

- **Output**

Outputs are last field in the Logstash processing pipeline, piping the results of the processing to a designated output process. In this case, the output process was Elasticsearch, which had been configured to listen for incoming data on localhost,

---

<sup>35</sup>It should be noted that although substantial effort was expended to perform the similar filtering operations on the *syslog* files generated by the router container honeypot, there were multiple compatibility issues encountered which resulted in the failure to visualise these logs on the Kibana web interface.

port 9200. Some additional options were specified regarding the formatting of the data to be stored in the Elasticsearch index.

### 5.4.2 Honeypot Instance

To provide the EC2 management instance with honeypot data to visualise, the logs generated by the honeypot containers on the EC2 honeypot instance needed to be aggregated and then transferred securely to the management instance. This was achieved as described below.

#### 5.4.2.1 CRON

A CRON job was configured to run every 60 seconds on the honeypot instance, shipping the honeypot logs from their respective container volumes to a defined location on the honeypot instance: The */var/logs/* directory. This meant that logs generated by all honeypots present would be aggregated in a single location on the honeypot instance.

```
# m h  dom mon dow    command
* * * * * /bin/cp -a  /home/ubuntu/cowrievolumes/*/
              /var/log/cowrie/ >> /var/log/cowrie/cronlog.log
* * * * * /bin/cp -a  /home/ubuntu/router/*/
              /var/log/cowrie/router/ >> /var/log/cowrie/cronlog.log
```

Listing 5.4: This snippet shows 2 tasks scheduled using CRON, which copy the contents of the honeypot container volumes to the */var/logs* directory. These jobs are run every 60 seconds, the maximum possible frequency for a CRON job.

#### 5.4.2.2 Filebeat

Filebeat is an open-source application which was discussed in *Section 4.1.2.2* in the context of securely transferring honeypot logs from the EC2 honeypot instance to the EC2

management instance using SSL certificates. The SSL certificate previously generated by the management server was shared with the honeypot instance using Secure Copy (SCP).

The Filebeat configuration file, `/etc/filebeat/filebeat.yml`, was configured to inspect the `/var/log` directory on the EC2 honeypot instance for the presence of log files any time updated log files were added to it by CRON. It was then configured to send the logs to the EC2 management instance using private IP addressing, specifying the destination port to be the same as that configured in the *input* field of the Logstash configuration file. Lastly, Filebeat was configured to use the SSL certificate generated by the EC2 management instance for all data transferred.

## 5.5 Alert/Notification System

As another important element of the cyber incident monitor identified in *Section 4.1.2*, the configuration of a threat notification system was the final step in the implementation of the cyber incident monitor.

### 5.5.1 PSAD

As explained in *Section 4.1.2.4*, PSAD was identified as the tool of choice to enable alerting of system administrators to potential attacks happening on the honeypot host. It was deployed on the honeypot instance in order to provide an instantaneous notification to system administrators of potential attacks. A basic configuration generated email alerts when probing of the honeypot instance was detected, sending notification emails to the address `itsmyjobtofixtheproblem@outlook.com`, which could be specified in the `/etc/p-sad/psad.conf` configuration file. It was possible to configure PSAD to check the state of the network event logs as frequently as desired, for which a reasonable frequency was chosen to be every 5 seconds.

With PSAD it is also possible to completely blacklist or whitelist<sup>36</sup> certain IP addresses

---

<sup>36</sup>Whitelisting an IP address means that it is added to a list of addresses that are considered trustworthy.

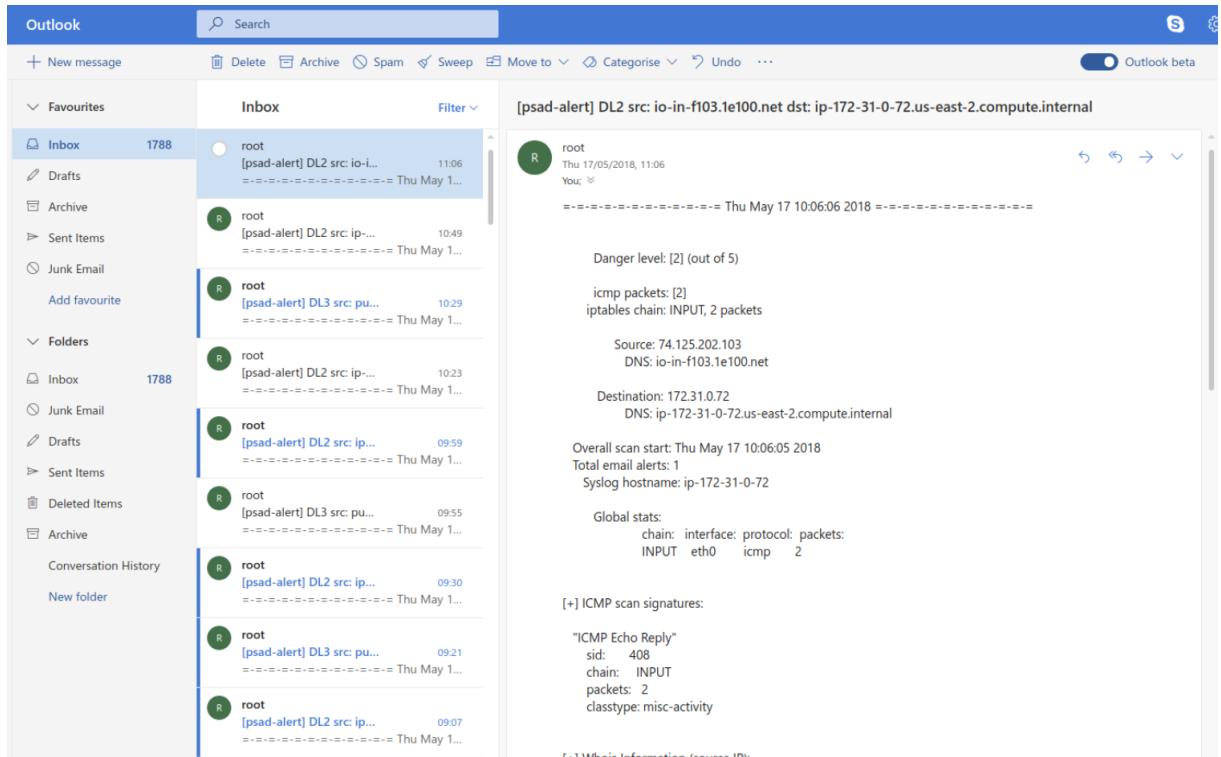


Figure 5.16: An Email Alert from PSAD

This screenshot shows the email inbox to which PSAD was configured to send email alerts. It can be seen that a port scan generated an email alert shown in the right-hand pane of the email inbox. PSAD has detected that the source of the scan is a domain *io-in-f103.1e100.net* which has sent a number of ICMP packets to the EC2 server instance, triggering this alert.

by editing the */etc/psad/auto\_dl* configuration file. This functionality was used to reduce the number of false positives being generated by PSAD<sup>37</sup>.

## 5.6 Summary

By the time all of the system components had been configured and implemented as described in this chapter, an end-to-end system cyber incident monitoring system was in place.

- Attack sessions could be simulated, connecting to the EC2 honeypot instance over

<sup>37</sup>After PSAD was initially configured, there were many email alerts generated from a particular server called *chillipepper.canonical.com*. Upon contacting Canonical Ltd. about the possibility of their server being compromised and used to launch attacks, it was found that the alerts were being generated by benign Network Time Protocol (NTP) time synchronisation traffic.

SSH or telnet and being presented with the router container login prompt. After authenticating with the router container, it was possible to connect in the same way to any container in the honeynet and interact with a Cowrie honeypot.

- All activity generated from interactions with the Cowrie containers was being successfully shipped, processed and visualised, with the visualisation dashboard accessible from a web browser.
- The entire Docker environment could be completely removed and redeployed with an identical configuration if required.

Thus, the objective of developing a honeypot-driven cyber incident monitor had largely been achieved at this point, with much remaining scope for configuration of both the honeypots and the visualisations once attack data had been obtained. This system would be used to conduct a series of experiments focused on determining effective design for adaptive honeypots, detailed in *Chapter 6*.

# Chapter 6

## Evaluation

This chapter focuses on providing an evaluation of the implemented cyber incident monitoring system, as well describing the design and conducting of experiments as part of the proposal for adaptive honeypot design.

*Section 6.1* provides an in-depth explanation of how the experiments were designed for evaluating the impact of attractiveness honeypot characteristics on the number of attacks received.

*Section 6.2* discusses how the experiments were conducted and their limited findings, detailing the nature of numerous setbacks that occurred and how these were dealt with. An explanation for the early conclusion of these experiments is also provided.

*Section 6.3* discusses the limited findings of the experiments as well as a qualitative evaluation of the cyber incident monitoring solution produced.

Finally, *Section 6.4* provides some brief conclusions regarding the evaluation of the honeypot experiments.

### 6.1 Design of Experiments

The conducting of experiments in this research is focused on the delivery of the second objective outlined in *Section 3.2.2*: To determine an improved design for effective, adaptive

honeypots. It was decided that the measure of effectiveness in these experiments would be defined by how many attacks each honeypot received. From this, it would be possible to present findings and draw conclusions regarding the development of effective adaptive honeypots.

The design of the experiments was influenced greatly by the time that remained to complete the entirety of the research: A period of just four weeks. A series of experiment iterations were devised to be run over the course of these 4 weeks. Table 6.1 describes the variables involved in each iteration, and the intended execution of these iterations with respect to time.

	Vendor A			Vendor B			Vendor C			Control		
	Week 1			Week 2			Week 3			Week 4		
	LC	SA	DT	LC	SA	DT	LC	SA	DT	LC	SA	DT
<b>HP1</b>	A	X	P	A	X	P	A	X	P	A	X	P
<b>HP2</b>	B	Y	Q	B	Y	Q	B	Y	Q	B	Y	Q
...	C	Z	R	C	Z	R	C	Z	R	C	Z	R
<b>HPN</b>	D	W	S	D	W	S	D	W	S	D	W	S
<b>Control</b>	*	*	*	*	*	*	*	*	*	*	*	*

Table 6.1: A set of planned experiment iterations in order to propose an optimal honeypot design. Three characteristics of the N Cowrie honeypots HP1, HP2 ... HPN are considered: The login credentials accepted (*LC*), the services advertised to be running on the container (*SA*), and the type of device advertised (*DT*). For each week, only one characteristic of the router honeypot is varied: The device vendor, corresponding to columns *Model A*, *Model B*, etc. Each variation of a Cowrie honeypot characteristic (*LC*, *SA*, *DT*) should be in place for 2 days, with all 12 experiment iterations taking place over 4 weeks. There is one day allocated per 7-day week for setting up the experiments, represented by the empty column for each week. In each iteration there is a single *Control* Cowrie honeypot used as an indication of baseline performance, and in *Week 4* a *Control* router honeypot is also employed for the same reason.

An explanation of how this plan translates into the configuration of the research environment is explained in the following subsections.

### 6.1.1 Router Honeypot Container

As the gateway between the public internet and the Cowrie honeynet, it is obvious that the router container should aid an attacker in accessing the deployed system. Facilitating easy access to this container is likely to increase the chance of a successful venture into the Cowrie honeynet, enabling valuable data to be captured. Thus, it was important to configure the router container to be as attractive and open as possible to the targeted attackers, IoT botnets.

The system had already been designed and implemented to facilitate exactly these requirements, and so the router container could easily provide:

- Remote access over either SSH or telnet from the public internet;
- Authentication using a number of valid username-password combinations;
- Root privileges inside the system;
- A powerful OS and toolkits;
- A network of nearby hosts to which an attack could be propagated;
- An attractive device banner indicating a device type.

In order to gain some insights regarding honeypot design from the router container and not just from the Cowrie honeypots, it was decided to vary a characteristic of the router container honeypot as part of the experiments: The device vendor advertised<sup>1</sup>. All other characteristics of the router container would be held constant, such that the same username-password credentials, services, toolkits, etc. would be available on this honeypot in all experiment iterations.

---

<sup>1</sup>In order to facilitate easy access to the Cowrie honeynet whilst also gaining some insights from allowing attackers to access the router container, the device vendor advertised in the SSH and telnet banners of the router honeypot was deemed to be a suitable and useful characteristic to measure. If this characteristic was found to impact upon the number of attacks received, it would give an indication of vendor awareness in IoT botnets.

With reference to the experiment iterations plan in table 6.1, the following experiments were designed for the measurement of the impact of device vendor on the number of attacks received:

- For each of *Week 1*, *Week 2* and *Week 3* shown in the table, the router container would be configured to advertise a different brand of router through SSH and telnet banners. Each banner will be left on the router container for 6 consecutive days of that week, with every other variable in the router honeypot environment being held constant.
- In *Week 4*, a *Control* router container would be used: One which doesn't advertise any brand or model, but which simply uses the default 'Ubuntu 16.0.4 LTS' SSH and telnet banner with hostname *router*. By having a control, it is possible to compare the results of the other three weeks against a baseline to determine whether or not it matters that a device is advertised as being from a particular vendor.

For the 3 device banners to be configured on this honeypot for *Week 1*, *Week 2* and *Week 3*, it is desirable to base those device brands used in the banner configuration on real-world data. Masscan is a powerful asynchronous port scanner which was used by the researchers behind the IoTpot honeypot to capture device banners from devices on the public internet listening on the telnet 23/TCP port. [58] The researchers used these banners to customise the honeypots to look like a real telnet-enabled device on the Internet. The effectiveness of this approach in their evaluation of IoT botnet behaviour motivated the decision to use Masscan to capture device banners in this research for use in the experiments. The most commonly occurring device vendor banners would be used as a basis for the experiments, since a more popular device vendor can be assumed to be more likely to be attacked than a less popular one.

### 6.1.2 Cowrie Honeypot Containers

In the Cowrie honeynet, there is substantially more scope for measuring the impact of varying honeypot characteristics since there are a greater number of honeypots available: As motivated in the formulation of the research objectives in *Section 3.2.2.1*, a honeynet provides a solution to the accurate measurement of relative effectiveness of honeypots in the same environment, since the effects of time can be largely disregarded<sup>2</sup>.

For each of the 4 weeks for which the experiments would be run, it was decided that 3 honeypot characteristics would be varied for a period of 2 days per week each. These characteristics are:

1. The login credentials accepted by the honeypot;
2. The services running on the honeypot's container ports<sup>3</sup>;
3. The type of device advertised by the honeypot.

To illustrate how this translates into an experiment iteration, consider column *LC* in *Week 1* in table *6.1* as an example.

- *LC* corresponds to the variation of the *login credentials* accepted by honeypots HP1, HP2, ... HPN over a 2-day period.
- For each of honeypots HP1, HP2, ... HPN, the login credentials accepted should vary. Other variables should be held constant<sup>4</sup>.

It was decided that five Cowrie honeypots would be deployed in each experiment iteration, one of which would be the control honeypot. The presence of a control iteration

---

<sup>2</sup>Since all honeypots in the honeynet should be simultaneously exposed to the same environmental conditions, the effect of time variability in the experiments is minimised.

<sup>3</sup>In the research conducted by Dowling *et al.* in the implementation of a Zigbee honeypot, the generation of unencrypted network traffic was an interesting honeypot characteristic used in an attempt to attract the attention of attackers. [14] Measuring the impact that such a measure has on the number of attacks received by a honeypot was deemed a useful characteristic to measure.

<sup>4</sup>This means that if the *login credentials* (*LC*) are the characteristic being varied, both the *services advertised* (*SA*) and the *device type* (*DT*) should be held constant

is important, since it allows for definitive conclusions to be drawn as to whether or not the results obtained by customised honeypots have any impact on the number of attacks they receive.

## 6.2 Conducting Experiments

The experiment iterations were planned to be conducted in the order in which they are specified in table 6.1, starting with varying the login credentials of the Cowrie honeypots and the addition of a banner advertising a device manufacturer.

### 6.2.1 Capturing Device Banners

Masscan was used to scan the entire IPv4 address space for devices listening on ports 22/TCP (SSH) or 23/TCP (telnet) over the course of two days. The telnet and SSH banners of any device that responded were captured in log files. The IP addresses of these devices were also captured.

The banners were extracted from the log files into a new file, where they were counted using shell commands. In total, 71,149 banners were captured after scanning port 23/TCP (telnet) and 179,849 banners were captured after scanning port 22/TCP (SSH).

Many of the banners obtained specified vague information regarding device models and manufacturers, with many not being identifiable from the device banners at all. In order to make the results obtained relatively usable for falsifying honeypot device banners, the Masscan results were used in conjunction with Google searches to identify vulnerable device models.

### 6.2.2 Experiment 1, Attempt 1

The first experiment iteration undertaken corresponds to the *Week 1, LC* experiment shown in table 6.1. This involved the variation of login credentials accepted by the Cowrie

honeypots deployed in the Docker honeynet, and the configuration of a vendor-specific device banner on the router container.

#### 6.2.2.1 Configuration of the Router Honeypot

Huawei devices were encountered 42 times explicitly in telnet banners, and 6 times explicitly in SSH banners<sup>5</sup>. This made Huawei the most popular router manufacturer that could be identified from the captured banners. However, since no discernible models could be identified from these banners some research was conducted into vulnerable Huawei router models, immediately identifying their HG-532d model as being highly vulnerable to exploit. [94] [95] The device manual was found to contain the default username-password credentials *user/user*. Thus for this experiment iteration, the router container was advertised as a Huawei HG-532d router.

In order to facilitate as many successful logins as possible, the router container was configured to allow passwordless login for 14 different user accounts. These user accounts were added based on results captured from a temporary deployment of a plain Cowrie honeypot on an AWS EC2 instance, which was set up to capture credentials that are actually being used by IoT botnets for brute-force authentication<sup>6</sup>.

#### 6.2.2.2 Configuration of the Cowrie Honeypots

As discussed, five Cowrie honeypots were deployed as part of this experiment iteration. One of these honeypots was the control, a plain Cowrie honeypot with no customisations added to it.

Since the login credentials were the characteristic being varied across the remaining

---

<sup>5</sup>It is assumed that not all Huawei devices scanned will have provided an explicit Huawei banner in response to the Masscan probes. Thus, there could have been many more Huawei devices present in the results found which could not be identified simply from their banners.

<sup>6</sup>The full list of user accounts added were *root*, *admin*, *cisco*, *guest*, *admin1*, *support*, *ubnt*, *default*, *Admin*, *service*, *supervisor*, *Administrator*, *administrator*, and *user*. There were also a number of additional usernames captured: However, these contained numbers which are invalid characters for Ubuntu usernames, meaning that they could not be added as an user. The *user* username was added based on the fact that the Huawei HG532d device manual specified this as the default username.

4 honeypots in this experiment iteration, the device type and services available on the honeypots needed to be kept constant.

- **Device Type Advertised**

All honeypots were given device names similar to the default hostname assigned to Cowrie. Thus the Cowrie honeypots were named *srv01*, *srv02*, *srv03*, *srv04* and *srv05*, where *srv04* was the *control* honeypot.

- **Services Advertised**

The simplest solution to keeping the services advertised by the honeypots constant was to use the default Cowrie container configuration, where the emulated SSH and telnet services were the only services that would be detected by an attacker performing a port scan on these honeypots.

- **Login Credentials Accepted**

Different accepted login credentials were configured on each of the Cowrie honeypots, with the exception of the control honeypot *srv04*. These were as follows:

1. *srv01* accepted username *root* with any password;
2. *srv02* accepted username *admin* with any password;
3. *srv03* accepted any credentials provided after a random number of attempts, which can be configured using the Cowrie's *AuthRandom* option<sup>7</sup> in the *cowrie.cfg* file;
4. *srv05* used the username-password combinations captured from one of the four attackers in the first exposure of Cowrie to attack<sup>8</sup>, explained in *Section 5.1.2.3*;

---

<sup>7</sup>By setting this option in the *cowrie.cfg* file, access will be granted to an attacker after *randint(minimum, maximum, cache\_combinations)* login attempts. In this case the parameters specified were 3, 5 and 10 respectively.

<sup>8</sup>All of the brute-force login attempts from this bot used the username *admin*, along with a total of 13 different passwords.

### **6.2.2.3 Findings of the Experiment**

Once the system was exposed to the public internet on ports 22/TCP (SSH) and 23/TCP (telnet), there were immediately plenty of connection attempts captured by the router container over both SSH and telnet. However, it was soon observed based on the event timestamps that most connections were terminated immediately after connecting.

It was concluded upon reflection that this behaviour could be due to detection of the system as a honeypot environment: Since attackers were being given passwordless root access to the router container, it would be reasonable to expect this to raise suspicions about the legitimacy of the system. This is not ideal, since the intention behind allowing passwordless access was to enable attackers to easily gain access to the system: The addition of password authentication, which allows only 1 password per system user, would restrict the number of attackers that could successfully access the system.

The experiment iteration was concluded after 2 days. By this time, there were no attacks logged by any of the Cowrie honeypots in the honeynet, since no attackers had progressed beyond the router container. It was decided on this basis that the experiment iteration should be repeated under the same conditions, with the exception that passwordless access to the router container should no longer be provided.

## **6.2.3 Experiment 1, Attempt 2**

Based on the failure of the system to capture any substantial attack data in the first attempt of experiment iteration 1, the system was re-deployed with the same configuration. This time however, passwordless authentication to the router container would not be permitted.

### **6.2.3.1 Reconfiguring the Router Credentials**

As an alternative to passwordless authentication, it was decided that the password for each existing user account would be configured to be identical to the username for sim-

plicity: For instance, for user *admin* the password would also be *admin*. This seemed a reasonable choice for permitted login credentials for each user account on the router container, since identical username-password combinations are commonly used in brute-force authentication attacks.

Adding this involved some simple edits to the router container’s entrypoint script.

#### 6.2.3.2 Findings of the Experiment

It was discovered after less than 24 hours of running this experiment that the EC2 honeypot server instance had completely crashed. Upon investigation, this crash appeared to be due to an unusual *fork bomb*<sup>9</sup> behaviour by CRON: The *syslogs* showed that it had recursively restarted itself until there were too many processes for the EC2 host to handle, eventually crashing the instance entirely.

After checking the logs generated by the router container, there were a number of successful connection sessions logged for both SSH and telnet: However, after examining the *.bash\_history* file none of the commands executed looked like they could have caused this fork bomb behaviour to occur intentionally. It was thus concluded that this had occurred due to a resource overload on the system

Though the system had been offline for quite some time before the failure was discovered, in the few hours that the experiment had been running there had been a number of successful login attempts to the router container. However, there were also no attacks that had reached the Cowrie honeynet.

The contents of the *.bash\_history* file showed only 6 commands had been executed during the time that the experiment was running. Three similar sequences of commands had been executed as follows:

```
cat /proc/mounts; (/bin/busybox LWQTL —— :)
```

---

<sup>9</sup>A fork bomb is an unintended behaviour of a system, where processes are continually made to replicate themselves until all available CPU resources are depleted, resulting in a crash failure of the system. It is often used as a DOS attack.

```

cd /dev/shm; cat .s —— cp /bin/echo .s; (/bin/busybox LWQTL —— :)
cat /proc/mounts; (/bin/busybox GQZJN —— :)
cd /dev/shm; cat .s —— cp /bin/echo .s; (/bin/busybox GQZJN —— :)
cat /proc/mounts; (/bin/busybox WEGOD —— :)
cd /dev/shm; cat .s —— cp /bin/echo .s; (/bin/busybox WEGOD —— :)
```

Of note was the fact that these commands attempted to invoke the *BusyBox* shell, which had not been installed in the router container as an oversight in the implementation process. In their analysis of the behaviour of the Hajime IoT botnet, Edwards *et al.* noted almost identical invocations of the *BusyBox* shell as part of what they deduced was a fingerprinting mechanism to determine the properties of the victim system. [54] A legitimate system on which BusyBox had been installed would return the response '*CHDGL: applet not found*'. Also noted by these researchers is the purpose of the *cat /proc/mounts* command sequence which "checks the system mounts for a writeable location in the target filesystem", and the sequence *cd /var; cat .s —— cp /bin/echo .s; /bin/busybox ECCHI*, which among other things "picks the first writeable path that is not /proc, /sys, or / and uses that as its working path". The fact that these commands were observed may indicate the presence of the Hajime botnet.

Overall, the results of this experiment iteration were interesting but not complete, since no attacker progressed beyond the initial fingerprinting stage because of the absence of the *BusyBox* utility. It would be necessary to run this experiment iteration a third time with the utility installed.

#### 6.2.4 Experiment 1, Attempt 3

The same experiment that was set up in the previous two iterations was re-deployed for a third time after adding the *BusyBox* utility to the router container Dockerfile. There was no additional configuration of this utility required, and it was simply installed upon rebuilding the container image.

#### 6.2.4.1 Findings of the Experiment

As planned, the experiment iteration was run for two days before the system was taken offline and the results examined. This time, the network events logged by *syslog* showed that several more attackers had managed to successfully authenticate with the router container over both SSH and telnet.

The environment was examined, and the findings were as follows:

- Even after repeating this experiment for a third time, no attacker had proceeded from the router container to the Cowrie honeynet. This was evident from the fact that there were no Cowrie visualisations generated on the Kibana dashboard hosted by the management server. There were also no commands observed in the *.bash\_history* file that would indicate an attacker having probed the *dmz* network.
- Like the previous iteration, commands were observed in the *.bash\_history* file that could be attributed to bot attackers. However, only one sequence similar to those highlighted in *Section 6.2.3.2* was observed. Instead, another sequence *enable; system; shell; sh* was repeatedly observed. This was another sequence discussed by Edwards *et al.*, which they explain "are sent in a blind attempt to navigate whatever vendor-specific command-line interface (CLI) the Telnet server implements.... If any command fails, it will fail". [54] Upon manually attempting to execute the same sequence inside the router container, it was found that the *system* and *shell* commands did not execute inside the container, explaining the fact that the attack sequence ended at this point.
- It appeared that a human attacker had gained access to the system in this experiment<sup>10</sup>. The *.bash\_history*, when examined, showed that the attacker had down-

---

<sup>10</sup>This was inferred by the commands that were found to have been executed after inspecting the *.bash\_history* file, in particular the *clear* command which is used to clear the output displayed in a Linux console window. This is a command that would be useless to an automated bot attacker, and was a tell-tale sign of a human attacker.

loaded a Python script from a web domain<sup>11</sup> after installing the *wget* utility. Upon visiting this domain, the script was found to contain open-source licensing information, and was traced back to an open-source Github project called *speedtest-cli*. [96] This is a tool used to test the network bandwidth available to a device, something that is undoubtedly of interest to an attacker intending to conduct a volumetric network attack such as DDoS. Other activities within the same session included the creation of a new user *huawei*, checking the uptime of the system using the *uptime* command, and checking the system specifications using the *lscpu* command.



Figure 6.1: DDoS-for-Hire: The Web Domain of a Human Attacker

The webpage shown was linked to an attack session in the router container honeypot, where a Python script was downloaded from this site and used to measure the network bandwidth available to the system. The homepage shown in this image advertises DDoS-for-hire services, where it costs €15 to buy a 2GB/second attack and a 4GB/second attack costs €20.

### 6.2.5 Concluding the Honeypot Experiments

Something that became very clear from the experiments conducted was the unpredictability of conducting honeypot experiments. Given that 4 weeks had been allocated to complete all 12 iterations of experiments planned, it was crucial that there were minimal setbacks to the progression of the experiments within this time.

Over a week after setting up the first of these 12 iterations, the first experiment iteration had been deployed a total of three times due to the issues outlined in *Section*

---

<sup>11</sup>The URL from which this script was downloaded is <http://cybernetik.000webhostapp.com/speedtestvps.py>. At the time of writing, the site hosted at this address had been taken down: However, a screenshot of the domain landing page captured after discovering this activity can be seen in figure 6.1.

6.2. It was concluded that conducting the remainder of the planned experiments was infeasible, given that less than three weeks remained to conduct the remaining iterations. Thus, the conducting of the planned experiments was discontinued.

## 6.3 Discussion

Both the achievements and the limitations of the research are evaluated and discussed in this section.

### 6.3.1 Honeypot-Driven Cyber Incident Monitoring

The first objective of this research was to develop a honeypot-driven cyber incident monitor that would be feasible to deploy in modern IT infrastructures, making it feasible for organisations to implement active network defence.

#### 6.3.1.1 Effectiveness of the Containerised Honeynet

As one of the novel features of the implemented incident monitoring system, the effectiveness of the honeynet deployment proposed by this research forms an important component of the evaluation. A useful set of criteria outlined by Chin *et al.* [97] and used by Piscarcik *et al.* [61] to evaluate their honeynet deployments specifies a number of key criteria that should be fulfilled by an effective honeynet. These are discussed below in relation to the system implemented in this research.

##### 1. Scalability

The proposed solution is scalable insofar as it has been tested. The largest honeynet deployment trialled during the research consisted of 11 containerised honeypots<sup>12</sup>, which was hosted continuously for 2 days without any component failing. However,

---

<sup>12</sup>10 Cowrie honeypot containers and the router honeypot container.

during this period there was very little interaction with the system, limiting the load on system resources that would be present in a production environment.

In theory, the honeynet is capable of supporting a large number of honeypot containers. The ability to add more honeypots is largely dependent on the system resources available: The host system must have sufficient resources to support the number of containers deployed. This is however certain to be much less than that required for an equivalent number of virtual machines on the same host.

## 2. Flexibility

The implemented honeynet solution is flexible in its operability, configurability and maintainability.

- Control over the honeynet from an administrative perspective is straight-forward: Containers can be managed through the host even whilst running, meaning that reconfiguration of the networking and administrative access to container resources can be done on-the-fly as required.
- The use of the Cowrie honeypot also contributes to the flexibility of the honeynet. As a highly configurable and customisable open-source honeypot, tweaking the configuration of the honeypots in response to new threat insights is very achievable.
- It would also be relatively straight-forward to use a different honeypot in place of Cowrie in this system: Without having to make any changes to the network configuration, new Dockerfiles can be defined for alternative honeypots if required, integrated to the system with simple edits to the deployment script.

## 3. Attack Containment

Attack containment refers to the restriction of propagation of attacks if a honeypot is compromised. Though attack containment is never completely assured, the

measures taken to provide attack containment described in *Section 4.2.4* mitigate against undesirable attack propagation outside the honeynet.

- The isolation of the host from the *dmz* network on which the honeynet is hosted restricts the ability of an attacker to launch a network-based attack from within the honeynet.
- The carefully considered restrictions placed on an attacker inside the Cowrie honeypots mean that it is highly improbable that an attacker would be able to launch an attack from these honeypots in the first place. The fact that the Cowrie honeypot is an emulated environment in particular restricts an attackers activities, since they are not interacting with a real system with a networking stack, OS and powerful utilities.
- The high-interaction router container is the only component of the system where attack containment may not be fully assured, since this container interfaces directly with a network to which the host is connected. This is a potential area for improvement, where given additional time investigating the use of alternative networks that don't interface with the host system could yield an improved level of isolation.

#### 4. Stealth

Stealth refers to the ability of the honeynet to deceive an attacker and convince them that it is a legitimate system. It is difficult to provide a definitive conclusion regarding this property of the deployed honeynet, given that the interactions of attackers with the system were limited at best.

There is certainly scope for improvement of the stealth of the honeynet based on the little data that was obtained through the experiments outlined in *Section 6.2*, since in each iteration it was found that IoT bot attackers did not progress their attacks beyond their known initial phases. [20] [54] To reach any definitive conclusions

about this property would require substantially more time and experimentation.

However, the properties of the containerised honeynet lend themselves to deceiving attackers about the legitimacy of the environment, since all honeypots are hosted in containers with real OS images and networking stacks. For instance, Chin *et al.* propose that stealth can be achieved through measures such as falsifying high network latency. [97] An attacker testing the legitimacy of this honeynet based on such properties will receive legitimately delayed responses by virtue of the use of containers in the system.

## 5. Resource Management

This criterion refers to the presence of simple mechanisms for allocating resources in the honeynet, which is largely provided through the use of Docker in this implementation.

- The nature of containers is that the resources allocated to the running of an application are highly controlled and specified entirely by their image definition. In order to allocate additional resources, a simple update to this image definition is all that is required.
- The host system resources consumed by containers are controllable primarily through the specification of arguments when creating containers. As stated in the Docker documentation, "by default, a container has no resource constraints ... Docker provides ways to control how much memory, CPU, or block IO a container can use, setting runtime configuration flags of the docker run command". [98] Though these resource restrictions were not explicitly addressed as part of the development of the containerised honeynet, it would be straightforward to add these resource restrictions to the deployment script as part of the *docker create* commands.

## 6. Ease of Deployment

Chin *et al.* state that for a honeynet to be considered easy to deploy, "users are not burdened by complicated configurations and procedures in order to set up their own honeypots".

In the proposed system, the use of containers and the automation of network configuration through scripting means that a fully configured honeynet of connected containers can be removed and re-deployed on a Linux host in approximately 5 minutes<sup>13</sup>. As the initial motivation behind using containers in the proposed system, this criterion has certainly been met by the system.

Ease of usability through an interface to the system is also highlighted by Chin *et al.* as being of importance to the ease of deployment of a honeynet. [97] The usability aids implemented in the *do\_honeypot.sh* deployment script remove additional complexity from the deployment process, providing simple console messages to guide in the deployment and configuration of the system.

Thus, it is concluded that although there is room for improvement of various aspects of the implemented honeynet, the requirements outlined in this classification have largely been delivered by this system.

### 6.3.1.2 Architecture of the Monitoring System

A limitation that has been identified upon evaluation of the system architecture is that of the centralisation of log processing in the incident monitor. Although the centralised aggregation of honeypot data for visualisation in the system enables the correlation of attack data to provide a holistic view of the entire honeynet, it also means that the system is architected in such a way that it has a single point of failure.

The current system architecture is non-ideal from the perspective of system reliability, since if the EC2 management server instance was to become overloaded and crash, the incident monitoring system would no longer be able to provide its intended service. A

---

<sup>13</sup>This value is based on empirical timing measurement of the time taken to deploy the system from scratch on a clean EC2 host.

more distributed approach to processing and visualising data generated by the honeynet would improve the reliability of the proposed system. This will be discussed in *Section 7.2.5* in the context of future work for this project.

However, there are some features of the architecture of the implemented system which make it more scalable: In particular, that the honeynet in this deployment is hosted locally with Docker rather than as a distributed network. In the evaluation of the *TraCING* incident monitor by Vasilomanolakis *et al.* [43], they note that their distributed architecture creates a bottleneck with regards to resource competition "when the number of new sensors is massively increased", since multiple distributed honeypots individually provide their results to the centralised incident monitoring system. In the system developed in this research, the data generated by the honeypots is aggregated locally through the use of Docker volumes and CRON rather than being transmitted by each honeypot, reducing the network traffic to the centralised management instance.

#### **6.3.1.3 Impact of Visualisation on the Usability of Honeypots**

The use of visualisation in this project was motivated by the need to make honeypot-driven solutions more usable, providing a way for administrators to easily obtain a holistic view of the state of active network defence in their systems.

Though there was no experimental data captured by the Cowrie honeypots whose logs could be processed and visualised on the Kibana web interface, simulated attacks were conducted to allow visualisations to be produced for evaluation. Figures 6.2, 6.3 and 6.4 illustrate the capacity of the ELK log management stack to communicate complex log data concisely and effectively.

What can be noted about these visualisations is the effectiveness with which information is imparted to the user: Rather than sifting through reams of honeypot log data, correlated information regarding attack behaviours and patterns is represented succinctly in the visualisations. The fact that the logs used to produce these are shipped from

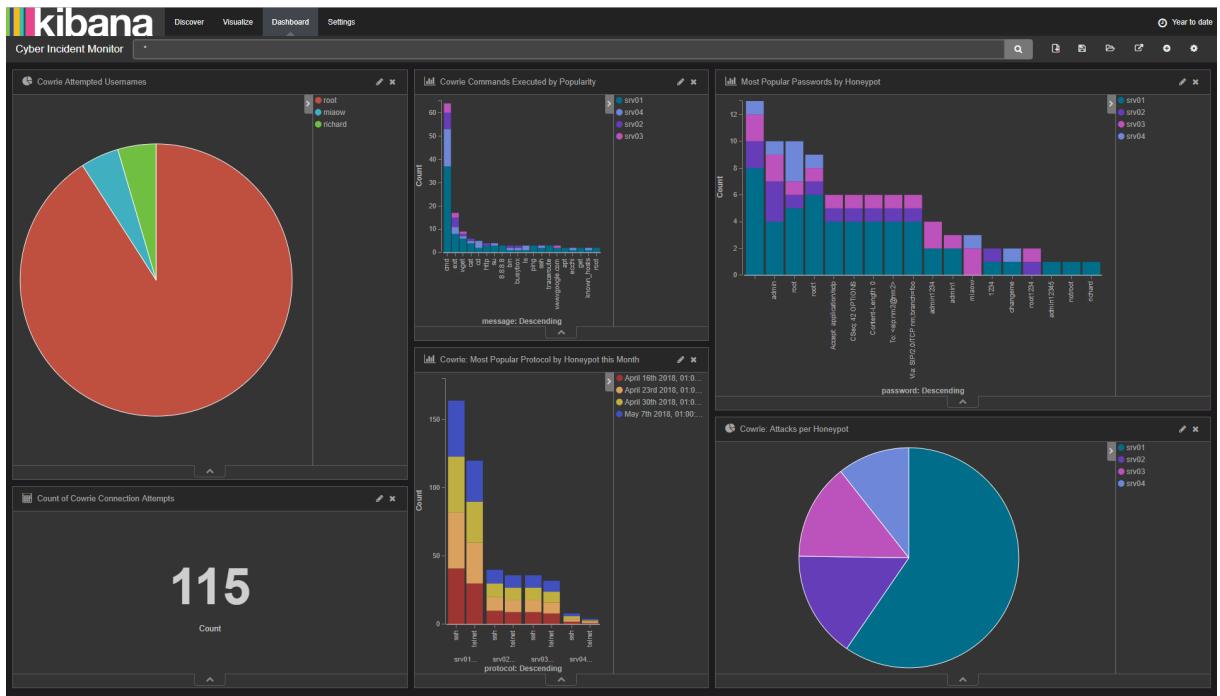


Figure 6.2: Cyber Incident Monitor on Kibana

This image shows the Kibana visualisation dashboard that was created as part of the incident monitoring solution. A number of charts are shown, measuring everything from the popularity of passwords per honeypot to the proportion of telnet and SSH attacks received by each Cowrie honeypot. The dashboard provides a concise description of the threat state of the honeynet.

the honeypot server approximately every 60 seconds<sup>14</sup> means that an administrator is facilitated in understanding the current threats to their system rapidly.

### 6.3.2 The Design of Adaptive Honeypots

As such, it would appear that unpredictability is the nature of research where honeypots are involved. It had been hoped that a quantitative evaluation would be provided in this section: However, due to the issues encountered with completing all planned iterations of experiments, the evaluation of the contribution of this research towards the design of adaptive honeypots is restricted by the incomplete results obtained in *Section 6.2*.

<sup>14</sup>As explained in *Section 5.4.2.1*, the honeypot logs were made available to Filebeat every 60 seconds before being shipped to the EC2 management instance. Any additional delay in delivery is likely to be due to network conditions.

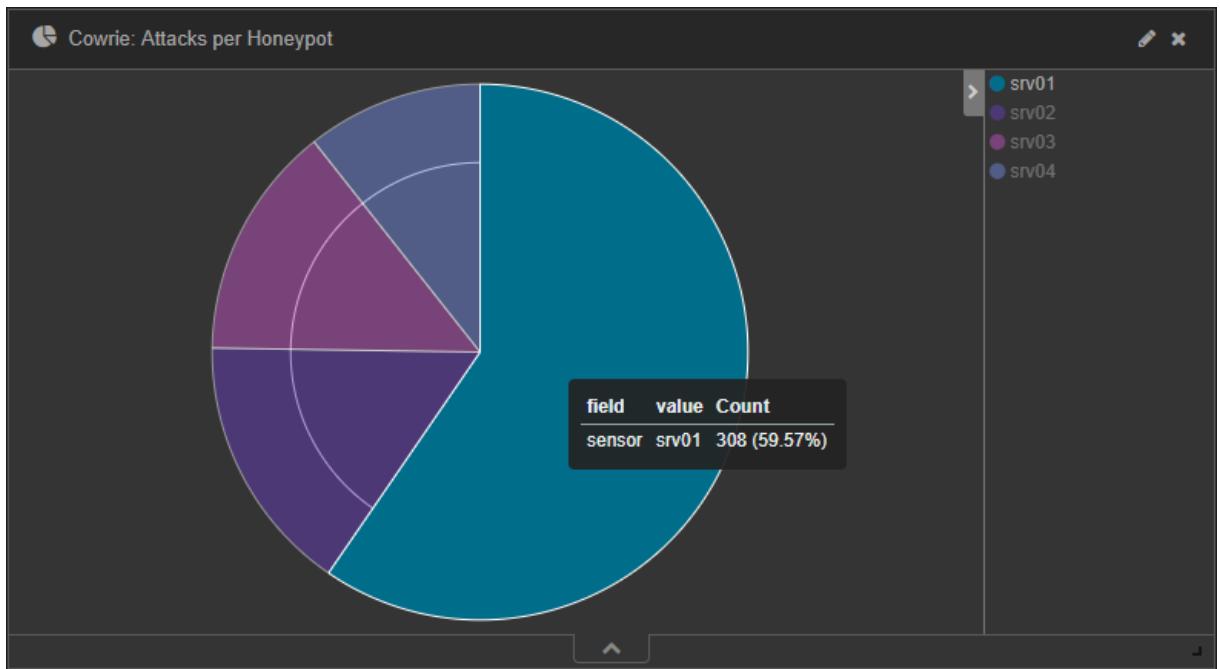


Figure 6.3: Visualisation: Attacks per Honeypot

This image shows a single visualisation created in Kibana as part of the cyber-incident monitor. This pie-chart shows the proportion of attacks each Cowrie honeypot received within a month.

#### 6.3.2.1 Unpredictability of Honeypot Experiments

It has been found that the process of designing honeypots to be adaptive to dynamic threats is a time-intensive one. Unforeseen issues such as the requirement to implement a high-interaction container honeypot, and the further significant challenges which were faced in implementing this effectively, are illustrative of the unpredictability of designing adaptive honeypots.

The fact that the contributions of just 3 characteristics of honeypots were considered in the experiments outlined in *Section 6.1* is explained by the limited remaining time within which the experiments could be conducted. Had more time been available, a set of comprehensive honeypot experiments could be conducted accounting for several other honeypot characteristics.

When compared to the time spent conducting similar research experiments in the closely related projects discussed in *Section 2.4*, the findings were as follows:

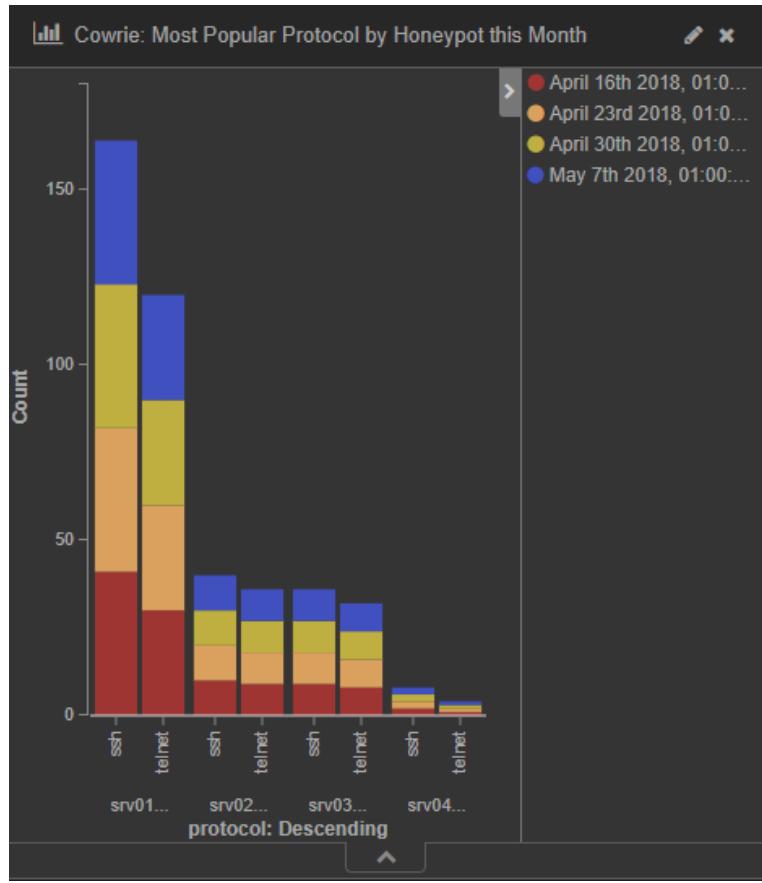


Figure 6.4: Visualisation: Most Popular Protocols per Honeypot by Month

This image shows another visualisation created in Kibana as part of the cyber-incident monitor. This chart shows the proportion of SSH and Telnet attacks received by each of the honeypots for each week over the last month.

- The researchers involved in the *IoTPot* project conducted experiments to evaluate the effectiveness of their design in 2 separate phases: a 144-day trial period from 2014/11/07 to 2015/03/31 which was used to "understand the attackers' behavior and (discuss) the proper setting of the honeypots", followed by a 28-day stable period from 2015/04/01 to 2015/05/09 in which more structured experiments were conducted. [58]
- In their evaluation of honeypot performance in the *TraCING* incident monitor, Vasilomanolakis *et al.* explain that a five-month deployment of their honeypot-driven incident monitor allowed them to obtain meaningful results. [43]

These findings further illustrate the unpredictability of evaluating the effectiveness of

honeypots. It is perhaps an oversight that the time that would be required to conduct such experiments was not realised at an earlier stage.

In recognition of the fact that none of the experiments conducted could provide *conclusive* evidence regarding the design of adaptive honeypots, there was a need to consider what was (i) necessary, (ii) sufficient and (iii) additional towards the achievement of the original objective.

### **1. Necessary**

It is necessary that knowledge is gained about the contribution of a honeypot's design to its effectiveness in attracting attacks through experiments.

### **2. Sufficient**

It is sufficient that the knowledge gained is not conclusive but is based on experimental data.

### **3. Additional**

It is additional that the knowledge gained is definitive and conclusive based on experimental data.

It is clear that obtaining conclusive results regarding effective design for adaptive honeypots became infeasible because of time constraints, ultimately resulting in the early termination of the experiment phase. However, the data obtained from the three experiment attempts, although not definitive, did provide some insights into the nature of designing adaptive honeypots.

- The presence of expected utilities and toolkits in a honeypot environment is clearly crucial to its success in encouraging attacks. This was evident from the data captured in attempts 2 and 3 of the experiment iteration performed, where bot attackers did not proceed with compromise of the system when certain utilities were not present in the honeypot environment.

- Support for protocols and services which are commonly exploited by attackers is an important feature of effective honeypots: this is evident by virtue of the sheer volume of brute-force authentication attacks which were received
- Basing honeypot design on current knowledge of attacks is a useful starting point in enabling attacks to be studied. In the design of the high-interaction router honeypot, it is likely that the experiments would have been more successful if more emphasis had been placed on configuring the environment to provide responses and utilities sought after by active IoT botnets, knowledge of which already exists in research. [20] [54]
- Honeypots must be designed to combat both automated and human attackers. It was not expected based on the findings of Barron *et al.* that attacks by human attackers would be likely to be encountered in the honeypot environment. [49] The fact an attack session by a human attacker was captured in the limited period for which experiments were conducted illustrates that honeypots should be designed to adapt to threats from human attackers, and not solely the more predictable attack patterns of bots.

### **6.3.2.2 Ability to Capture Attack Data**

As evidenced during the conducting of experiments in *Section 6.2*, the fact that keylogging was not available on the router container honeypot severely limits the conclusions that can be drawn regarding any experimental results that were obtained. For instance, although patterns of command execution were observed in the *.bash\_history*, the lack of timestamping and session IDs for these events meant that inferences were made regarding these sequences being from a single attacker.

There is plenty of room for improvement regarding the capturing of attack data, and given more time, it would almost certainly be possible to develop an effective and reliable keylogging mechanism for the router container.

### **6.3.2.3 Hosting Honeypot Experiments on a Cloud Platform**

It is quite probable that some of the attack patterns observed in *Section 6.2* may have been different had a different hosting solution been used to host the system. As noted by Vasilomanolakis *et al.* in their implementation of a honeypot incident monitor, [43] many cloud service providers publish their IP ranges on the public internet: AWS is one such provider. [99] In their paper examining the difference between human and automated attacks, Barron *et al.* also find that "location and host matters... if someone is operating on a constrained budget and wants to maximize the number of brute-forcing IP addresses collected, AWS appears to be the infrastructure that will facilitate this". [49]

As AWS are a very well-known and popular cloud hosting provider, they are likely an attractive target for attackers who have knowledge of their public IP ranges. Thus, there is potential for some bias in results obtained in a deployment such as that implemented in this research, where all honeypots are hosted in a single geographical region by a single hosting provider.

### **6.3.2.4 Fingerprinting Container Environments**

An interesting observation regarding the fingerprinting of honeypot environments is that there are many ways in which an attacker can detect that they are inside a container, if the honeypot is indeed the container itself. This was noted during the implementation of the system, when the command *cat /proc/1/cgroup* was executed, leading to the output shown in figure 6.5. This is consistent with the findings of Kedrowitsch *et al.* when evaluating the susceptibility of containers to being fingerprinted. [62]

This may not come as a surprise to an attacker: As organisations increasingly include containers as part of their infrastructure, this information would likely not identify the container as being a honeypot. However, it is certainly worth considering that if containerised honeypots begin to be used more widely, fingerprinting of these systems as honeypot environments may also become more common.

```

root@65960460d0e1:/# cat /proc/1/cgroup
11:pid:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
10:cpuset:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
9:freezer:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
8:net_cls,net_prio:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
7:memory:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
6:devices:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
5:blkio:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
4:perf_event:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
3:hugelb:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
2:cpu,cpuacct:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
1:name=systemd:/docker/65960460d0e1fe25f78ee764528b590e0d94f109bc99826ad02cf46506130a6c
root@65960460d0e1:/# █

```

Figure 6.5: Fingerprinting of Docker Environments

This console output was generated as a result of executing the command `cat /proc/1/cgroup` inside the router container. It clearly includes several references to 'docker', which any attacker aware of Docker would immediately recognise as being a container environment.

## 6.4 Summary

It is clear that the unpredictability of conducting honeypot experiments was a barrier to obtaining measurable data and providing definitive conclusions regarding the adaptive design of honeypots. This was largely an oversight in the design phase of the research, where had more time been available, there would have been a greater opportunity to achieve this objective as part of this research.

However inconclusive, useful insights have been obtained from the experiments conducted. This is an encouraging indication of the potential for a real contribution to be made through experiments of this nature, and provides a strong basis for completing the remaining iterations of the project as future work.

# Chapter 7

## Conclusions and Future Work

The objectives set out for this research in *Section 3.2* of this document, although not achieved fully, yielded enhancements to the deployability of active network defence through incident monitoring as well as valuable insights into the way in which such systems should be designed to improve their threat detection capabilities.

Section 7.1 draws some conclusions regarding the outcomes of the research, focusing on the original elements of the research objectives.

Section 7.2 explores a number of potential avenues for expanding upon the research conducted in this project. These include potential improvements to the proposed approach as well as opportunities to conduct additional work building on the foundations laid by this research.

Finally, *Section 7.3* includes some final remarks and recommendations based on the research conducted.

### 7.1 Conclusions

Though it is clear that this research would have benefited significantly from additional time within which to meet its objectives, the substantiality of knowledge gained from the exploration of challenges in the field of security today was unprecedented.

The major achievements and conclusions drawn from this research are described in the following subsections.

### **7.1.1 Cyber Incident Monitoring in Critical Infrastructures**

The major contribution of this work has been the development of a novel containerised honeynet-driven incident monitor. The value that can be provided by the succinct description of attack data through visualisation is evident: These incident monitors allow system administrators to represent threat data in a meaningful way, enabling a holistic view of their systems.

There is tangible evidence that the incident monitor developed in this research provides greater usability of honeypot data than that which would have resulted from just using honeypots alone:

- Visualisation of the honeypot data means that reams of logs don't have to be searched through in order to understand the most relevant threat information;
- The aggregation of data from multiple honeypots by a centralised logging system, allowing trends to be easily identified;
- The ability to receive instantaneous threat intelligence alerts via email, pin-pointing the perpetrator;
- The fact that installation and configuration of the honeypots is automated, as well as the networking required to connect them.

As such, this development has effectively addressed many of the major barriers to employing active network defence in infrastructures today: A valuable contribution to the field of security.

### **7.1.2 The Potential of Containers for Security Applications**

The use of containers has added substantial value to the honeynet solution implemented in this project. It serves as a proof-of-concept with regards to usability of active network defence mechanisms for network administrators by providing the ability to automatically install and configure multiple replicable environments on-the-fly.

As evidenced by this research, the work being done to containerise security applications is something that presents many challenges: There are no manuals or existing resources to refer to in the development of such systems, and so there is a lot of room for research and evaluation of different approaches to doing so.

### **7.1.3 The Importance of Honeypots in Active Network Defence**

As has been observed during the course of this research, working with honeypots can be highly unpredictable, making conclusive research difficult to achieve under time constraints. However, it is clear that the ability of honeypots to provide active network defence through deceiving and monitoring attackers make them an attractive option for combating the dynamic, evolving threats that are increasingly impacting individuals and organisations across the world. Critical service infrastructures are in desperate need of the *adaptive* threat detection capabilities of honeypots to protect their systems and those who depend on them.

### **7.1.4 The Future of IoT Security**

It is clear even from the limited experiments that were conducted in this research that IoT botnets are extremely active: Every "thing" with internet connectivity is a means of exploiting communications and interaction. People are increasingly dependent on interconnected services and applications, and as a result are vulnerable to cyber threats beyond the reach of their own devices.

Critical service architectures will continue to be targeted by severe cyber attacks well

into the future as nation states continue to engage in cyber-combat. There will need to be huge innovation in the area of IoT security before the gargantuan challenges being faced by these infrastructures can even be remotely resolved: It is only when a system is designed with security in mind that a system can be considered even relatively secure. Applications are continuing to have connectivity integrated into them with very little consideration given to their security, and this is something that is not likely to end any time soon. It is likely that until there is a truly market-driven incentive for manufacturers to implement decent security mechanisms in their products, this is unlikely to change.

## 7.2 Future Work

Whilst an end-to-end cyber incident monitoring system was successfully developed in this research, the full intended evaluation of the system was not possible as discussed in *Section 5.3.3*. However, this gives rise to opportunities for additional developments and enhancements based on the foundations laid by this research, a number of which are discussed below.

### 7.2.1 Extension of the Cowrie Honeypot

As explained in the *Section 5.3.3*, the Cowrie honeypot by design does not allow outbound network connections. This research would have benefited greatly from the extension of the Cowrie honeypot to include this functionality, but time constraints meant that such a development was not within the scope of the project.

Though an in-depth inspection of how such an extension could be added to the Cowrie source code was not conducted, it is clear that it would involve relatively significant development effort to implement an additional networking client as part of the Cowrie application in Python. The Cowrie application already includes Python modules for SSH and telnet traffic forwarding which would form a useful starting point in this implementation.

Such a networking client would need to be able to have a view of the underlying net-

working stack of the host, such that outbound connections to other Cowrie honeypots can be facilitated and a view of hosts nearby can be obtained so that an attacker is not expected to probe IP address ranges blindly. However, the addition of outbound networking capabilities in the Cowrie honeypot would bring it closer to becoming a high-interaction honeypot, which creates a number of new issues for the use of Cowrie. Circumventing the risks associated with high-interaction honeypots is challenging, as was found during this project.

- The networking client would need to implement some destination authentication mechanism such that outbound connections from Cowrie are only permitted to other Cowrie honeypots. This would minimise the risk of a Cowrie honeypot propagating an attack to another Cowrie honeypot;
- The interface between the networking client and the network stack of the host system would need to be very carefully implemented and audited for it to be shipped as part of the Cowrie honeypot package, since it should be very difficult for an attacker to exploit such an interface to the host system.

The addition of this feature as an optional function of the Cowrie honeypot would facilitate the future deployment of honeynets entirely based on the Cowrie honeypot, leveraging all of its configuration and logging capabilities. This would be a very useful contribution to the Cowrie development effort, and would be an excellent starting point for a continuation of the work conducted in this research.

### **7.2.2 An Improved High-Interaction Containerised Honeypot**

It is clear from the challenges encountered in both the implementation of the incident monitor and the experiments to determine an effective design for adaptive honeypots, that the router container honeypot left much to be desired in terms of its ability to facilitate attacks. This is largely due to the fact that this honeypot was not included as an original component of the proposed system, and so did not benefit from the substantially greater

consideration of related literature that preceded the design of other components of the system.

There is much opportunity for improvement of the router container honeypot. There were no existing container-based high interaction honeypots found during the course of this research, a gap which many would benefit from being filled.

Some potential areas for improvement of the existing router honeypot that have been identified include the following:

- Further isolation from the host system through investigation of alternative Docker networking approaches;
- The implementation of reliable, detailed keylogging inside the router container that could be used to visualise the commands executed by attackers;
- Extension of the image definition of the router honeypot to target requirements of specific IoT botnets, rather than solely improving the design by incrementally altering the environment on an experiment-by-experiment basis.

Given these improvements and sufficient time, it would be strongly recommended that the experiment iterations or an extension of these should be resumed in a bid to improve upon the current state-of-the-art in adaptive honeypots. It is however strongly recommended that any future work conducted into experiment-based design of honeypots should consider the unpredictability factor associated with such work during the design phase.

### **7.2.3 Machine Learning Approaches to Adaptive Honeypot Design**

There is huge potential for the implementation of machine learning-based honeypot solutions, similar to that developed by the researchers behind IoT Candy Jar. [60] The use of such techniques enables honeypots to self-improve by learning what makes them most

effective in enticing and maintaining the interest of attackers, making for ultimately more effective active defence in networks.

In the context of the work that has been done in this project towards proposing design approaches for adaptive honeypots, a machine learning back-end component similar to the *IoTLearner* module implemented by the IoT Candy Jar researchers would be an excellent enhancement to the system that has been developed. [60] Such a module could potentially perform many of the functions currently performed by the router honeypot, providing responses that would encourage the attacker to progress their attack to the stage that they would look to propagate it to nearby hosts. To enable this propagation would either require:

- A traffic routing module similar to that proposed in *Section 7.2.1* to route SSH and telnet attack traffic to the Cowrie honeynet; or
- An architecture similar to that proposed by the IoT Pot researchers, where the *IoTLearner* machine learning module would act as the *front-end responder* in the IoT Pot architecture and forward connection requests to a high-interaction backend to be handled. [58] [60]

In summary, the use of machine learning approaches in the implementation of dynamic, reactive honeypots coupled with the novel architecture and deployability of the incident monitoring solution developed in this research would be a very worthwhile investment as a future project.

#### 7.2.4 Alternative Threat Notification Systems

The alert generation system that was used in this project was the commonly used port scan detection tool PSAD. It was found to effectively detect probing of the EC2 honeypot instance on which the containerised honeypot was hosted, generating email alerts for any probes detected almost instantaneously. However, it was also relatively prone to generating false positives because of scenarios such as that described in *Section 5.5*.

There is certainly potential for other alerting mechanisms to be used in this system which may provide more accurate and effective communication of threat intelligence. Of note is the fact that the Cowrie project supports integration with Slack, an instant messaging and collaboration service that is used in many organisations for workplace communication. As a system that can provide instant messages to a distributed set of mobile users, there is potential for the use of Slack and similar messaging channels as a means of providing threat intelligence alerts generated by the Cowrie honeynet.

### 7.2.5 Alternative System Architectures

As identified in *Section 6.3.1.2*, the fact that there is only a single management server supporting the processing and visualisation of all of the honeypot log data means that there is a single point of failure in the system. Should the management instance crash due to any system failure, the entire incident monitoring system would not be available to provide analysis and visualisation of attack data.

More distributed approaches such as passive replication could provide greater reliability for the system: By passively replicating the management instance, there would always be a node capable of processing honeypot logs and providing visualisations to system administrators. This would incur significant initial configuration overhead: An initial estimation of effort involved here for a single replica would include:

- The provision of a new EC2 instance capable of hosting the ELK log processing stack;
- Replicated configuration of the ELK stack tools and Nginx on the new EC2 instance;
- The generation and distribution of an additional SSL certificate to the EC2 honeypot instance;
- Updating the configuration of Filebeat to ship honeypot logs to both server instances;

- The addition of a monitoring communication between the two management servers, such that the passive replica can take over if the primary fails.

In addition, ongoing maintenance overheads must be considered by virtue of the addition of another node to the system. However, adding replication would significantly increase the reliability of what is a time-sensitive threat mitigation service.

### 7.3 Closing Remarks

This research has seen the development of a containerised cyber-incident monitor: A proposal to address the need for active defence mechanisms in systems of critical services. Though there is significant work remaining to make this system production-ready, there are many elements of the proposed system which address the current issues being faced in these infrastructures by providing a feasible, flexible active defence solution.

Active network defences with their ability to adapt to detect new threats will only become more important into the future as the gap between threats and the mitigations against them continues to widen. There is a fundamental shift in outlook required by those managing the security of IT systems, such that preventative security measures are not the sole mechanisms protecting them from increasingly sophisticated threats.

The crucial takeaway from this research is that it is of critical importance in the development of any software system that a threat model is included in the design from the very beginning. If this is widely included as part of software design and implementation, it is likely that more risks will be mitigated and that fewer vulnerabilities will arise. Though it is only theoretically possible to evaluate and eradicate *all* vulnerabilities in a system, a security-oriented approach to designing software systems would go a long way towards reducing the implications of serious attacks on their users.

# Appendix A

## Abbreviations

Abbreviation	Expanded Term
ACK	Acknowledgement
API	Application Programming Interface
AWS	Amazon Web Services
C+C	Command and Control
CLI	Command Line Interface
CRON	Command Run-ON
DoS	Denial of Service
DDoS	Distributed Denial of Service
DMZ	De-Militarised Zone
DNS	Domain Name System
DPI	Deep-Packet Inspection
DVR	Digital Video Recorder
EC2	Elastic Compute Cloud (AWS)
ELK	Elasticsearch-Logstash-Kibana Stack
FTP	File Transfer Protocol

<b>Abbreviation</b>	<b>Expanded Term</b>
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
LPWAN	Low-Powered Wide-Area Network
LXC	Linux Container
MHN	Modern Honey Network
NAT	Network Address Translation
NHS	National Health Service, UK
NTP	Network Time Protocol
OS	Operating System
PaaS	Platform as a Service
PSAD	Port Scan Attack Detection
RAM	Random Access Memory
RCE	Remote Code Execution
RSA	Rivest, Shamir, and Adelman
SAN	SubjectAltName field of SSL certificate
SANS	System Administration, Networking, and Security Institute
SCADA	Supervisory Control and Data Acquisition
SCP	Secure Copy

<b>Abbreviation</b>	<b>Expanded Term</b>
SHA	Secure Hash Algorithm
SMB	Server Message Block
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSH	Secure SHell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
VM	Virtual Machine
WSN	Wireless Sensor Network
YAML	Yet Another Markup Language

# Bibliography

- [1] National Cyber Security Centre UK, “Russian Military ‘Almost Certainly’ Responsible for Destructive 2017 Cyber Attack - NCSC Site.” <https://www.ncsc.gov.uk/news/russian-military-almost-certainly-responsible-destructive-2017-cyber-attack>, 2018. Accessed: 16th May 2018.
- [2] United States Department of Justice, “Memorandum for Heads of Department Components: Cyber-Digital Task Force.” <https://www.justice.gov/opa/press-release/file/1035457/download>, February 2018.
- [3] B. Krebs, “U.K. Hospitals Hit in Widespread Ransomware Attack.” <https://krebsonsecurity.com/2017/05/u-k-hospitals-hit-in-widespread-ransomware-attack/>, 2017. Accessed: 8th May 2018.
- [4] Oracle Corporation, “Anatomy of a Cyber Attack: The Lifecycle of a Security Breach,” tech. rep., 2017.
- [5] A. Khalimonenko, O. Kupreev, and K. Ilganaev, “DDoS attacks in Q4 2017 - Securelist.” <https://securelist.com/ddos-attacks-in-q4-2017/83729/>, 2018. Accessed: 2nd April 2018.
- [6] M. T. Bastos and D. Mercea, “The Brexit Botnet and User-Generated Hyperpartisan News,” *Social Science Computer Review*, vol. 0, no. 0, p. 0894439317734157, 0.
- [7] Synopsys Inc., “Heartbleed Bug.” <http://heartbleed.com/>, 2014. Accessed: 25th April 2018.

- [8] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, “Security, Privacy and Trust in Internet of Things: The Road Ahead.,” *Computer Networks*, vol. 76, pp. 146–164, 2015.
- [9] Statista GmbH, “IoT: Number of Connected Devices Worldwide 2012-2025 — Statista.” <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>, 2018. Accessed: 22nd April 2018.
- [10] United Nations, Department of Economic and Social Affairs Population Division, “World Population Prospects: The 2017 Revision, Key Findings and Advance Tables,” 2017. ESA/P/WP/248.
- [11] T. Lackorzynski and S. Koepsell, “”Hello Barbie” - Hacker Toys in a World of Linked Devices,” in *Broadband Coverage in Germany; 11. ITG-Symposium*, pp. 1–7, March 2017.
- [12] Insecam.org, “Insecam - World biggest online cameras directory.” <https://www.insecam.org/>, 2017. Accessed: 17th May 2018.
- [13] S. Farrell, “LPWAN Overview,” Internet-Draft draft-ietf-lpwan-overview-10, Internet Engineering Task Force, Feb. 2018. Work in Progress.
- [14] S. Dowling, M. Schukat, and H. Melvin, “A ZigBee HoneyPot to Assess IoT Cyber-attack Behaviour,” in *Proc. 28th Irish Signals and Systems Conf. (ISSC)*, pp. 1–6, June 2017.
- [15] “Telnet Protocol Specification.” RFC 854, May 1983.
- [16] Shodan, “default password - Shodan.” <https://www.shodan.io/search?query=default+password>, 2018. Accessed: 14th May 2018.
- [17] B. Schneier, “The Internet of Things Is Wildly Insecure - And Often Unpatchable.” <https://www.wired.com/2014/01/theres-no-good-way-to-patch-the-internet-of-things-and-thats-a-huge-problem/>, 2017. Accessed: 26th March 2018.

- [18] C. M. Lonvick and T. Ylonen, “The Secure Shell (SSH) Authentication Protocol.” RFC 4252, Jan. 2006.
- [19] Brian Krebs, “Krebs on Security - In-depth Security News and Investigation.” <https://krebsonsecurity.com/>, 2018. Accessed: 22nd April 2018.
- [20] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, “Understanding the Mirai Botnet,” in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1093–1110, USENIX Association, 2017.
- [21] B. Schneier, “Botnet of Things.” [https://www.schneier.com/essays/archives/2017/03/botnets\\_of\\_2017.html](https://www.schneier.com/essays/archives/2017/03/botnets_of_2017.html). Accessed: 26th March 2018.
- [22] B. Krebs, “Who is Anna Senpai, the Mirai Worm Author?” <https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/>, 2017. Accessed: 26th March 2018.
- [23] Dan Goodin, “BrickerBot, the permanent denial-of-service botnet, is back with a vengeance — Ars Technica.” <https://arstechnica.com/information-technology/2017/04/brickerbot-the-permanent-denial-of-service-botnet-is-back-with-a-vengeance/>. Accessed: 12th May 2018.
- [24] The Janit0r, “Internet Chemotherapy.” <https://archive.is/PQAnU#selection-13.5156-13.5200>. Accessed: 12th May 2018.
- [25] United States Computer Emergency Readiness Team, “HIDDEN COBRA - North Korea’s DDoS Botnet Infrastructure.” <https://www.uscert.gov/ncas/alerts/TA17-164A>, 2017. Accessed: 17th December 2017.
- [26] P. L. Beyza Unal, “Cybersecurity of Nuclear Weapons Systems: Threats, Vulnerabilities and Consequences,” Tech. Rep. MSU-CSE-06-2, International Security Depart-

ment, The Royal Institute of International Affairs, Chatham House, 10 St James's Square, London SW1Y 4LE, January 2018.

- [27] J. Brenner, *Keeping America Safe: Towards more Secure Networks for Critical Sectors*. 2017.
- [28] N. Scaife, P. Traynor, and K. Butler, “Making Sense of the Ransomware Mess (and Planning a Sensible Path Forward),” *IEEE Potentials*, vol. 36, pp. 28–31, Nov 2017.
- [29] Sir Amyas Morse, “Investigation: WannaCry Cyber Attack and the NHS,” 2017.
- [30] W. Smart, “Lessons learned: Review of the WannaCry Ransomware Cyber Attack,” February 2018.
- [31] A. Voronkov, L. H. Iwaya, L. A. Martucci, and S. Lindskog, “Systematic Literature Review on Usability of Firewall Configuration,” *ACM Comput. Surv.*, vol. 50, pp. 87:1–87:35, Dec. 2017.
- [32] O. Santos, *End-to-End Network Security: Defense-in-Depth*. Cisco Press, 2007.
- [33] S. M. Bellovin, *Thinking Security: Stopping Next Year’s Hackers*. Library of Congress Cataloging-in-Publication Data, 2016.
- [34] F. Schneider, “Blueprint for a Science of Cybersecurity,” tech. rep., Cornell University, May 2011.
- [35] C. Wang and Z. Lu, “Cyber Deception: Overview and the Road Ahead,” *IEEE Security & Privacy*, vol. 16, pp. 80–85, March/April 2018.
- [36] I. Mokube and M. Adams, “Honeypots: Concepts, Approaches, and Challenges,” in *Proceedings of the 45th Annual Southeast Regional Conference*, ACM-SE 45, (New York, NY, USA), pp. 321–326, ACM, 2007.
- [37] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder, “A Survey on Honeypot Software and Data Analysis,” *CoRR*, vol. abs/1608.06249, 2016.

- [38] L. Spitzner, “Honeypots: Catching the Insider Threat,” in *Proceedings of the 19th Annual Computer Security Applications Conference*, ACSAC ’03, (Washington, DC, USA), pp. 170–, IEEE Computer Society, 2003.
- [39] Thinkst Applied Research, “Canary - Know when it Matters.” <https://canary.tools/>, 2018. Accessed: 10th May 2018.
- [40] B. McFarland, “Ethical Deception and Preemptive Deterrence in Network Security,” tech. rep., SANS Institute, April 2005. Accessed: 17th May 2018.
- [41] L. Spitzner, “Honeypots: Are They Illegal? — Symantec Connect.” <https://www.symantec.com/connect/articles/honeypots-are-they-illegal>, 2003. Accessed: 17th May 2018.
- [42] M. A. Sasse and M. Smith, “The Security-Usability Tradeoff Myth [Guest editors’ introduction],” *IEEE Security & Privacy*, vol. 14, pp. 11–13, Sept.-Oct. 2016.
- [43] E. Vasilomanolakis, S. Karuppiah, P. Kikiras, and M. Mühlhäuser, “A Honeypot-driven Cyber Incident Monitor: Lessons Learned and Steps Ahead,” in *Proceedings of the 8th International Conference on Security of Information and Networks*, SIN ’15, (New York, NY, USA), pp. 158–164, ACM, 2015.
- [44] Supriyo Biswas, “An Introduction to the Docker Ecosystem - Boolean World.” <https://www.booleanworld.com/introduction-docker-ecosystem/>, 2017. Accessed: 13th May 2018.
- [45] A. Leung, A. Spyker, and T. Bozarth, “Titus: Introducing Containers to the Netflix Cloud,” *Commun. ACM*, vol. 61, pp. 38–45, Jan. 2018.
- [46] T. Combe, A. Martin, and R. D. Pietro, “To Docker or Not to Docker: A Security Perspective,” *IEEE Cloud Computing*, vol. 3, pp. 54–62, Sept 2016.
- [47] U. Tamminen, “desaster/kippo: Kippo - SSH Honeypot.” <https://github.com/desaster/kippo>. Accessed: 16th May 2018.

- [48] Cuckoo Sandbox, “GitHub - cuckoosandbox/cuckoo: Cuckoo Sandbox is an Automated Dynamic Malware Analysis System.” <https://github.com/cuckoosandbox/cuckoo>. Accessed: 24th March 2018.
- [49] T. Barron and N. Nikiforakis, “Picky attackers: Quantifying the role of system properties on intruder behavior,” in *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, (New York, NY, USA), pp. 387–398, ACM, 2017.
- [50] DTAG Community Honeypot Project, “T-Pot 17.10 - Multi-Honeypot Platform rEvolution.” <http://dtag-dev-sec.github.io/mediator/feature/2017/11/07/t-pot-17.10.html>, 2018. Accessed: 22nd December 2017.
- [51] J. Trost, “Modern Honey Network — Anomali.” <https://www.anomali.com/blog/mhn-modern-honey-network>, 2014. Accessed: 25th March 2018.
- [52] T. C. Project, “GitHub - cowrie/cowrie: Cowrie SSH Honeypot (based on kippo) with SCP, SFTP, exec command, direct-tcpip support and many other features.” <https://github.com/cowrie/cowrie>. Accessed: 10th May 2018.
- [53] M. Oosterhof, “Cowrie Honeypot - Security Intelligence.” <http://www.micheloosterhof.com/cowrie/>. Accessed: 10th May 2018.
- [54] S. Edwards and I. Profetis, “Hajime: Analysis of a decentralized internet worm for IoT devices,” tech. rep., Rapidity Networks Security Research Group, October 2016.
- [55] The Cowrie Project, “GitHub - cowrie/docker-cowrie: Cowrie Docker GitHub repository.” <https://github.com/cowrie/docker-cowrie>, 2018. Accessed: 8th May 2018.
- [56] Niels Provos, “Honeyd General Information.” <http://www.honeyd.org/general.php>, 2004. Accessed: 11th May 2018.

- [57] N. Provos, “Honeyd: A Virtual Honeypot Daemon,” 01 2003.
- [58] Y. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, “IoT-POT: A Novel Honeypot for Revealing Current IoT Threats,” *Journal of Information Processing*, vol. 24, no. 3, pp. 522–533, 2016.
- [59] C. Leita, M. Dacier, and G. Wicherksi, “SGNET: A Distributed Infrastructure to handle Zero-Day Exploits,” tech. rep., Eurecom, February 2007. Accessed: 16th May 2018.
- [60] T. Luo, Z. Xu, X. Jin, Y. Jia, and X. Ouyang, “IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices,” in *Black Hat*, (Las Vegas, NV, USA), 2017.
- [61] P. Pisarcík and P. Sokol, “Framework for Distributed Virtual Honeynets,” in *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN ’14, (New York, NY, USA), pp. 324:324–324:329, ACM, 2014.
- [62] A. Kedrowitsch, D. D. Yao, G. Wang, and K. Cameron, “A First Look: Using Linux Containers for Deceptive Honeypots,” in *Proceedings of the 2017 Workshop on Automated Decision Making for Active Cyber Defense, SafeConfig@CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pp. 15–22, 2017.
- [63] N. Miramirkhani, M. P. Appini, N. Nikiforakis, and M. Polychronakis, “Spotless Sandboxes: Evading Malware Analysis Systems Using Wear-and-Tear Artifacts,” in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 1009–1024, May 2017.
- [64] DTAG Community Honeypot Project, “T-Pot 16.10 - Multi-Honeypot Platform Redefined.” <http://dtag-dev-sec.github.io/mediator/feature/2016/10/31/t-pot-16.10.html>, 2016. Accessed: 22nd December 2017.
- [65] S. Mansfield-Devine, “The Right Response: How Organisations should React to Security Incidents,” *Network Security*, vol. 2017, no. 12, pp. 16 – 19, 2017.

- [66] J. Cito, V. Ferme, and H. C. Gall, “Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research,” in *Web Engineering* (Bozzon, Alessandro and Cudre-Maroux, Philippe and Pautasso, Cesare, ed.), pp. 609–612, Springer International Publishing, 2016.
- [67] Amazon Web Services, Inc., “What is AWS? - Amazon Web Services.” <https://aws.amazon.com/what-is-aws/>, 2018. Accessed: 25th March 2018.
- [68] A. A. Mohallel, J. M. Bass, and A. Dehghantaha, “Experimenting with Docker: Linux Container and Base OS Attack Surfaces,” in *2016 International Conference on Information Society (i-Society)*, pp. 17–21, Oct 2016.
- [69] desaster, “FAQ - desaster/kippo Wiki.” <https://github.com/desaster/kippo/wiki/FAQ>. Accessed: 10th May 2018.
- [70] Elastic Co., “Secure communication with Logstash by using SSL — Filebeat Reference [6.2] — Elastic.” <https://www.elastic.co/guide/en/beats/filebeat/current/configuring-ssl-logstash.html>. Accessed: 12th May 2018.
- [71] E. Co., “Introduction — X-Pack for the Elastic Stack [6.2] — Elastic.” <https://www.elastic.co/guide/en/x-pack/current/xpack-introduction.html>, 2018. Accessed: 17th May 2018.
- [72] Docker Inc., “Docker Security — Docker Documentation.” <https://docs.docker.com/engine/security/security/>, 2018. Accessed: 1st April 2018.
- [73] J. Chelladurai, P. R. Chelliah, and S. A. Kumar, “Securing Docker Containers from Denial of Service (DoS) Attacks,” in *2016 IEEE International Conference on Services Computing (SCC)*, pp. 856–859, June 2016.

- [74] Docker Inc., “Docker Hub.” <https://hub.docker.com/>, 2018. Accessed: 16th May 2018.
- [75] Denis Vlasenko, “BusyBox - The Swiss Army Knife of Embedded Linux.” <https://www.busybox.net/downloads/BusyBox.html>. Accessed: 12th May 2018.
- [76] M. Oosterhof, “cowrie/INSTALL.md at master - cowrie/cowrie.” <https://github.com/cowrie/cowrie/blob/master/INSTALL.md>, 2017. Accessed: 12th May 2018.
- [77] K. Reitz, “Pipenv and Virtual Environments - The Hitchhiker’s Guide to Python.” <http://docs.python-guide.org/en/latest/dev/virtualenvs/>, 2018. Accessed: 16th May 2018.
- [78] Cisco Systems, Inc., “TP-Link TL-SC3171 IP Cameras Remote Command Injection Vulnerability.” <https://tools.cisco.com/security/center/viewAlert.x?alertId=32038>, 2013. Accessed: 13th May 2018.
- [79] MaxMind, Inc., “GeoIP2 Database Demo — MaxMind.” <https://www.maxmind.com/en/geoip-demo>, 2018. Accessed: 16th May 2018.
- [80] Docker Inc., “Dockerfile Reference.” <https://docs.docker.com/engine/reference/builder/>, 2018. Accessed: 13th May 2018.
- [81] V. Ferme, “Using Docker Containers to Improve Reproducibility in Software and Web Engineering.” <https://www.slideshare.net/vincenzoferme/using-docker-containers-to-improve-reproducibility-in-software-and-web-engineering>, 2013. Accessed: 13th May 2018.
- [82] Docker Inc., “Use bridge networks.” <https://docs.docker.com/network/bridge/>, 2018. Accessed: 13th May 2018.
- [83] Docker Inc., “Use overlay networks.” <https://docs.docker.com/network/overlay/>, 2018. Accessed: 13th May 2018.

- [84] Amazon Web Services, Inc., “Regions and Availability Zones - Amazon Elastic Compute Cloud, year = 2018, url = <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html>, howpublished = <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/regions-availability-zones.html>, note = Accessed: 13th May 2018.”
- [85] Elasticsearch BV, “Heap: Sizing and Swapping — Elasticsearch: The Definitive Guide [2.x] — Elastic.” <https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html>, 2018. Accessed: 13th May 2018.
- [86] Mitchell Anicas, “How To Install Elasticsearch, Logstash, and Kibana (ELK Stack) on Ubuntu 14.04 — DigitalOcean.” <https://www.digitalocean.com/community/tutorials/how-to-install-elasticsearch-logstash-and-kibana-elk-stack-on-ubuntu-14-04>, 2017. Accessed: 10th May 2018.
- [87] R. G. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, “Address Allocation for Private Internets.” RFC 1918, Feb. 1996.
- [88] R. Gerhards, “The Syslog Protocol.” RFC 5424, Mar. 2009.
- [89] kernc, “kernc/logkeys: A GNU/Linux keylogger that worked!” <https://github.com/kernc/logkeys>, 2018. Accessed: 16th May 2018.
- [90] “dumpkeys(1) - Linux manual page.” <http://man7.org/linux/man-pages/man1/dumpkeys.1.html>, 1993. Accessed: 11th May 2018.
- [91] “command line - How do I log all input and output in a terminal session? - Ask Ubuntu.” <https://askubuntu.com/questions/161935/how-do-i-log-all-input-and-output-in-a-terminal-session>, 2016. Accessed: 14th May 2018.
- [92] NGINX Inc., “Welcome to NGINX Wiki! — NGINX.” <https://www.nginx.com/resources/wiki/>, 2018. Accessed: 16th May 2018.

- [93] Fernando Dominiguez, “Logging Cowrie logs to the ELK stack.” <http://blog.fernandodominguez.me/logging-cowrie-logs-to-the-elk-stack/>, 2016. Accessed: 17th January 2018.
- [94] Huawei Technologies Co., Ltd, “Security Notice - Statement on Remote Code Execution Vulnerability in Huawei HG532 Product.” <http://www.huawei.com/en/psirt/security-notices/huawei-sn-20171130-01-hg532-en>, 2017. Accessed: 14th May 2018.
- [95] Check Point Software Technologies Ltd., “CPAI-2017-1016 — Check Point Software.” <https://www.checkpoint.com/defense/advisories/public/2017/cpai-2017-1016.html>, 2017. Accessed: 14th May 2018.
- [96] Matt Martz - sivel, “sivel/speedtest-cli: Command line interface for testing internet bandwidth using speedtest.net.” <https://github.com/sivel/speedtest-cli>, 2018. Accessed: 14th May 2018.
- [97] W. Y. Chin, E. P. Markatos, S. Antonatos, and S. Ioannidis, “HoneyLab: Large-Scale Honeypot Deployment and Resource Sharing,” in *2009 Third International Conference on Network and System Security*, pp. 381–388, Oct 2009.
- [98] Docker Inc., “Limit a container’s resources — Docker Documentation.” [https://docs.docker.com/config/containers/resource\\_constraints/](https://docs.docker.com/config/containers/resource_constraints/), 2018. Accessed: 16th May 2018.
- [99] Amazon Web Services, Inc., “AWS IP Address Ranges - Amazon Web Services.” <https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html>, 2018. Accessed: 10th May 2018.