NATIONAL UNIVERSITY OF IRELAND GALWAY

DOCTORAL THESIS

A New Framework for Adaptive and Agile Honeypots

Author:
Seamus DOWLING

Supervisors:
Dr. Michael SCHUKAT
Dr. Enda BARRETT

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

School of Computer Science and OSNA
College of Engineering and Informatics

January 30, 2020

Declaration of Authorship

I, Seamus DOWLING, declare that this thesis titled, "A New Framework for Adaptive and Agile Honeypots" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Rouse him, and learn the principle of his activity or inactivity. Force him to reveal himself, so as to find out his vulnerable spots."

Sun Tzu, The Art of War

NATIONAL UNIVERSITY OF IRELAND GALWAY

Abstract

College of Engineering and Informatics
School of Computer Science

Doctor of Philosophy

A New Framework for Adaptive and Agile Honeypots

by Seamus DOWLING

As new technological concepts appear and evolve, cyberattack surfaces and vectors are exploited. Every Internet facing device or service is vulnerable from the untrusted external Internet. Previously standalone devices are accessible through new hardware and software attack vectors. Internet of things significantly increases the attack surface available to malware developers. Malware methods of propagation and compromise are highly automated and highly repetitious. To react to new changes in malware evolution, cybersecurity measures have to evolve also. One such tool traditionally used for retrospective analysis is a honeypot. Honeypots facilitate attack interaction with scripted responses to attack command streams. Global honeynets capture large scale datasets which are useful for longitudinal analysis of malware methods. Standard honeypots are deployed for long periods and capture datasets comprising of automated and repetitive attacks. If the honeypot encounters an attack command it cannot process then the attack terminates. A **H**oneypot for **A**utomated and **R**epetitive **M**alware (HARM) can use reinforcement learning, to learn the best responses when interacting with attack sequences. The actual characteristics of malware, automation and repetition, can be exploited using embedded reinforcement learning within the honeypot. This adaptive ability allows honeypots to prolong interaction, realise attack sequences faster and conceal it's functionality from dedicated honeypot detection tools and methods. The agility of HARM's functionality can be further enhanced by periodically evaluating its performance so as to optimise further deployments targeting immediate threats. The cyclic method of development, deployment and optimisation improves honeypot operations and requires a *new framework for adaptive and agile honeypots*

Acknowledgements

There are a number of people to whom I need to say a big thank you, for getting this thesis to this point.

My supervisors Dr. Michael Schukat and Dr. Enda Barrett for their guidance and support. They both gave of their time and knowledge to direct me from beginning to end. They made important suggestions at key points and their comments were always encouraging. Be it a series of emails, a Skype call or over a cup of tea/coffee it was always a pleasure to talk with them. A mention also for Dr. Hugh Melvin who along with Michael, was a co-supervisor for the first three years. His comments in those initial years helped bring order to the early chaos.

Invariably my research seeped into my conversations at the workplace. Colleagues listened to my thoughts and theories, often voluntarily. Those unfortunate enough to engage were asked for their thoughts. Thanks to Andrew for kicking things around with me and Pearse for his Linux nous.

Supervisors and work colleagues engaged with me when required. This is as it should be. But family have had to live with me during the entire process. They give me the inspiration and motivation to do and to be. It's the reason why I started this and the reason why I got to this point. Yvonne and Leah Mai, thank you, thank you, thank you. Love you lots and always. It's all good.

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	**T**hird **G**eneration **P**artnerships Projects
APT	**A**dvanced **P**ersistent **T**hreats
AI	**A**rtificial **I**ntelligence
AWS	**A**mazon **W**eb **S**ervices
BAN	**B**ody **A**rea **N**etwork
BMS	**B**uilding **M**anagement **S**ystem
C&C	**C**ommand & **C**ontrol
CASSH	**C**ase **A**daptive **SSH**
CBR	**C**ased **B**ased **R**easoning
CERT	**C**omputer **E**mergency **R**esponse **T**eam
CPS	**C**yber **P**hysical **S**ystems
CPU	**C**entral **P**rocessing **U**nit
CWMP	**C**PE **W**AN **M**anagement **P**rotocol
DDoS	**D**istributed **D**enial **o**f **S**ervice
DHCP	**D**ynamic **H**ost **C**ontrol **P**rotocol
DMZ	**D**e**M**ilitarised **Z**one
DoS	**D**enial **o**f **S**ervice
FFD	**F**ull **F**unction **D**evice
FTP	**F**ile **T**ransfer **P**rotocol
GSM	**G**lobal **S**ystems for **M**obile Communications
HARM	**H**oneypot **A**utomated, **R**epetitive **M**alware
HiHP	**H**igh **i**nteraction **HoneyP**ot
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure
HVAC	**H**eating **V**entilation **A**ir **C**onditioning
IDS	**I**ntrusion **D**etection **S**ystems
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**engineering **T**ask **F**orce
IIoT	**I**ndustrial **I**nternet **o**f **T**hings
IoE	**I**nternet **o**f **E**verything
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol
IPS	**I**ntrusion **P**revention **S**ystems
IRC	**I**nternet **R**elay **C**hat
ISDN	**I**ntegrated **S**ervices **D**igital **N**etwork

LiHP	**L**ow **i**nteraction **HoneyP**ot
M2M	**M**achine to **M**achine
MDP	**M**arkov **D**ecision **P**rocess
MiHP	**M**edium **i**nteraction **HoneyP**ot
ML	**M**achine **L**earning
MLP	**M**ulti**L**ayered **P**eceptron
NAT	**N**etwork **A**ddress **T**ranslation
NFC	**N**ear **F**ield **C**ommunication
NIDS	**N**etwork **I**ntrusion **D**etection **S**ystems
NIST	**N**ational **I**nstitute of **S**tandards and **T**echnology
PAN	**P**ersonal **A**rea **N**etwork
PART	**PA**rtial **D**ecision **T**rees
PKI	**P**ublic **K**ey **I**nfrastructure
POMDP	**P**artially **O**bservable **MDP**
RASSH	**R**einforced **A**daptive **SSH**
RFD	**R**educed **F**unction **D**evice
RFID	**R**adio **F**requency **ID**entification
RL	**R**einforcement **L**earning
SARSA	**S**tate **A**ction **R**eward **S**tate **A**ction
SDN	**S**oftware **D**efined **N**etworking
SEP	**S**mart **E**nergy **P**rofile
SQL	**S**tructured **Q**uery **L**anguage
SSH	**S**ecure **SH**ell
SVM	**S**upport **V**ector **M**achines
SVR	**S**upport **V**ector **R**egresson
TCP	**T**ransport **C**ontrol **P**rotocol
UML	**U**ser **M**ode **L**inux
UMTS	**U**universal **M**obile for **T**elecommunications System
URL	**U**niform **R**esource **L**ocator
URN	**U**niform **R**esource **N**ame
VM	**V**irtual **M**achine
WINS	**W**indows **I**nternet **N**ame **S**ervice
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WPAN	**W**ireless **P**ersonal **A**rea **N**etwork
WSN	**W**ireless **S**ensor **N**etworks
WWW	**W**orld **W**ide **W**eb

Chapter 1

Introduction

1.1 State of the Art

THe war against malware is constantly evolving. Technology is evolving. Societal and industrial use of technology is evolving. Malware is evolving. Cybersecurity measures are evolving. There is an acceptance within the Internet community that security is required to counter existing malware. There is no silver bullet solution. Prevention, detection and remedial measures are all required to mitigate against the effects of malware. Updates, upgrades, patches, fixes, new definitions and versions, whitelisting, blacklisting, defense hardening are some of the wholly accepted measures used every day. The demilitarised zone (DMZ) for an organisation contains hardware and software tools to separate the trusted internal network from the untrusted external Internet. When 'Packets found on the Internet' was published in 1993, Bellovin [1] had discovered a covert attempt to compromise an available Internet service. New terminology for attacking code has evolved. Malicious software such as virus, worm, rootkit, trojan became common parlance in the formative years of the Internet. Since then Denial-of-Service (DoS), distributed DoS (DDoS), bots, ransomware, spyware, adware have added to the malware that society and industry has to mitigate against. New and emerging technologies and concepts change how society and industry interacts. Physical things can become Internet facing providing for remote access and control. An Internet-of-things (IoT) requires relevant security measures on all connected devices. The world wide web (WWW) has had a number of iterations. Whereas Web 1.0 was static and read only, version 2.0 became read and write. Social media allows users to connect and interact with others. Web 3.0, the Semantic web, provides connections between machines and people. It is "an attempt to represent knowledge in a way that allows computers to automatically come to conclusions and make decisions as a result of a certain type of reasoning" [2]. Web 4.0 and beyond creates an intelligent and telepathic web where machines will make decisions based on data that it has or is receiving. The concept of a smart city demonstrates how society, technology and industry can contribute to create better living conditions. IoT and industrial IoT (IIoT) will collect, transmit, collate, process, analyse and store vast amounts of data. Data sources can be very diverse as sampling population densities at a particular place and time through mobile

phone usage to automated control of cyber-physical systems (CPS). The criticality of the data can be as varied as communicating with a light bulb, to controlling critical IIoT infrastructure. Amid the heterogeneous data sources, communications protocols, channels and networks is the potential to manipulate the data to perform some malicious task. It is hoped that the security measures that are implemented globally will protect against malware propagation and compromise. But the truth is very different and daily occurrences of cyberattacks are commonplace.

As new technological concepts appear and evolve, cyberattack surfaces and vectors are exploited. Every Internet facing device or service is vulnerable from the untrusted external Internet. Previously independent devices are accessible through new software protocols and physical wired and wireless vectors. IoT significantly increases the attack surface available to malware developers. In 2016 Mirai [3] compromised IoT devices and contributed to large scale DDoS attacks. When the Mirai code was made available, it spawned new variants using the same methods on different attack vectors. Unfortunately, existing measures of patches, fixes and updates will always be required. To react to new changes in malware evolution, cybersecurity measures have to evolve also. Global cybersecurity groups such as Computer Emergency Response Team (CERT) and Kapersky capture, sandbox and publish methods to mitigate against attacks. Zero day attack discovery is sought whereby newly deployed malware is detected before getting established. If established, malware uses very automated and repetitive methods to propagate globally, infecting and compromising hosts and forming large scale botnets. These botnets can be used to launch enormous DDoS attacks or can further propagate and morph into new variants. Kapersky states that the number of malware modifications targeting IoT devices in the first half of 2018 was greater than all of 2017. Fig. 1.1 from Kapersky labs [4] also shows the growth of IoT malware in the last 3 years:

To understand malware and their variants, Spitzner [5] in 2003 suggested capturing attack data so as to understand the motives of their developers and the behaviour of the tools. He proposed a 'honeypot' as a "security resources whose value lies in being probed, attacked or compromised". Honeypots play a role in the uncovering attack behaviour. They are associated with longitudinal deployments capturing large datasets for retrospective analysis. Early versions of honeypots had low interaction capabilities. These were simple devices simulating Internet services and detected the presence of an attacker. Mid and high interaction honeypots provided for more interaction with an attacker. To adapt to new malware methods, honeypots have evolved to utilize new technologies. For example, Provos [6] in 2003, proposed Honeyd, a virtual honeypot to abstract deployment from the physical infrastructure. More recently, IoTPot [7] captured and analysed malware targeting IoT devices.

Honeypots as a dynamic real time analytical tool has imitations. Their design, configuration and operations also require careful consideration. A compromised honeypot could itself inadvertently participate in further attacks therefore honeypot

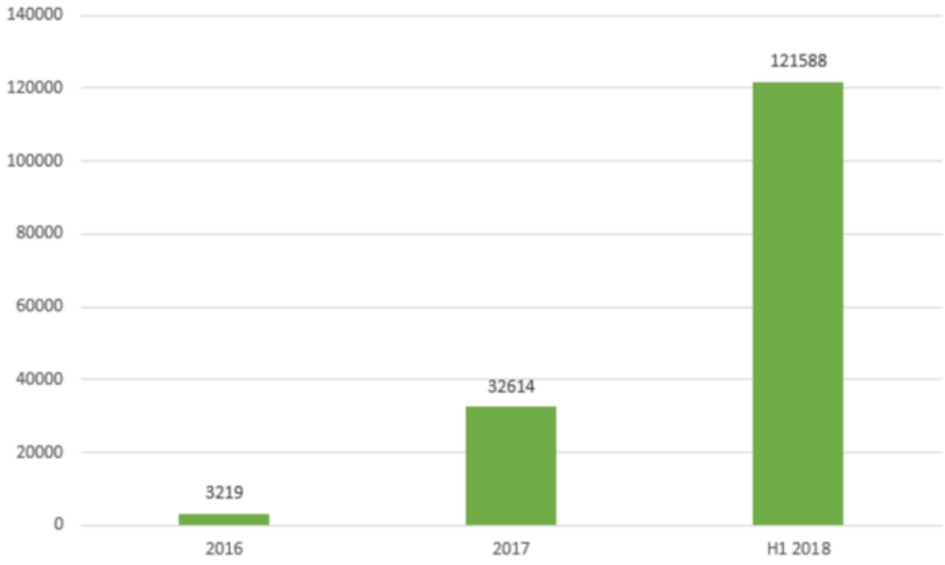


FIGURE 1.1: Number of malware samples for IoT devices collected 2016-2018 [4].

operations require constant monitoring. It is important that accessing the honeypot is configured correctly. Too hard and the honeypot captures very little attack traffic; too easy and the honeypot is flooded with irrelevant malware. Honeypots facilitate attack interaction with scripted responses to attack command streams. If the honeypot encounters an attack command it cannot process then the attack terminates. The characteristics of malware are automation and repetition. A standard deployed honeypot will therefore capture a dataset with a truncated collection of automated repetitive attacks. When honeypot deployments became popular, new versions simulated Internet services available on various attack vectors. To counter this, dedicated anti-honeypot software was released. Tools such as Honeypot Hunter [8] scanned the Internet for potential honeypot deployments. False services were created and connected to by the anti-honeypot tool. Honeypots are predominately designed to capture attacker interaction and therefore pretend to facilitate the creation and execution of these false services. This immediately tags the system as a honeypot. Automated malware also employs honeypot detecting mechanisms within its code. Once honeypot functionality has been exposed, malware such as botnets will cease the attempted compromise. Subsequent malware variants employ similar techniques to evade detection by known honeypots. This reduces the potential size of a captured dataset and subsequent analysis. With the growth of Web 4.0/5.0 concepts and the opportunities for new attack surfaces and vectors for malware developers, cybersecurity measures such as honeypots will need to adapt dynamically to new threats. New methods of honeypot design are required to overcome their limitations in this cyber war. Honeypots will be required to be adaptive and agile to have stealthier operations and provide faster forensics.

1.2 Problem Statement

The utilisation of technology by society and industry is rapidly evolving. IoT and IIoT produces a vast array of heterogeneous devices with varied services and capabilities. Smart 'things' will have differing levels of embedded security dependent on being full functions devices (FFDs) or reduced function devices (RFDs). Everyday objects can become Internet facing allowing for remote access and control. This rapid expansion of available web-enabled devices provides a fertile ground for malware developers [4]. Dedicated IoT malware has already exploited this space with the Mirai bot [3]. The resultant botnet generated gigabytes of flood data, targeting victims with DDoS attacks. Such was its success, it spawned multiple variants when the source code and methods were made available. Cyber security mechanisms designed to mitigate against these diverse and evolving variants need to be adaptive and dynamic. Defence mechanisms need to be more than just filters and warning systems. They need to dynamically adapt to harden their defences to protect against evolving external attacks. Cyber security groups need to create adaptive and dynamic tools, designed to detect and quickly analyse threats such as zero day attacks. One such tool traditionally used for retrospective analysis is a honeypot. Global networks of honeynets capture large scale datasets which are useful for longitudinal analysis of malware methods. Honeypots played an active role in capturing and analysing the Mirai bot [9]. The propagation and compromise methods used by malware, IoT or otherwise, are predominately automated and repetitive. There are some human interactions between botmaster and the command and control (C&C), and end user naivety. But traffic capture on honeypots demonstrates automation and repetition [10]. In the evolving war against malware, cybersecurity tools such as honeypots will need to *adapt* to new attack methods. This will require them to have more engagement with the attack sequence to capture more meaningful datasets. Incorporating machine learning techniques such as reinforcement learning can enable the honeypot to learn from automated and repetitive malware. This allows honeypots to adapt its functionality to malware interactions creating a honeypot for automated and repetitive malware (HARM). However, by contributing to better cyber forensics in this way, a honeypot can leave itself vulnerable to being compromised or being discovered. Dedicated tools scan the Internet for bogus services indicating honeypot operations. Malware also integrates detection measures within its code to determine if the device it has compromised is a real physical device or a fake or virtual services. If discovered or compromised, a honeypot can inadvertently terminate an attack or participate in propagating the malware. Therefore they need to be more covert in their operations. HARM's deployments need to be *agile* to target immediate threats. Periodic evaluation of reinforcement learning algorithms and policies optimises performance for further deployments. **The cyclic method of adaptive development, agile deployment and optimisation improves honeypot operations and requires a *new framework for adaptive and agile honeypots***

Honeypot development and deployment is governed by Seifert's taxonomy [11]. Published in 2006 it proposes how honeypots should appear to an attacker, how it interacts and collects information from the attack. However, this taxonomy could be considered out-dated with the evolving nature of malware. A new framework is required incorporating adaptivity and agility into honeypot functionality and operations.

1.2.1 Research Questions and Hypotheses

To contribute meaningfully to the rapidly changing world of cyber security and forensics, this thesis proposes a number of *enhancements* which advance the effectiveness of honeypot operations. A unique state action space formalism is proposed as a primary contribution (subsection 3.3.1.3). By integrating reinforcement learning into honeypot development, the thesis examines the following research questions:

1. Using reinforcement learning, can an IoT honeypot exploit the automated and repetitive characteristics of malware, learn the best responses to attack commands and converge towards the optimal learning policy?

2. By learning from automated and repetitive malware, can the adaptive IoT honeypot overcome inadvertent attack termination, realise entire attack sequences and increase the volume of data captured?

3. Using the captured dataset as an input stream to evaluate reinforcement learning algorithms and policies, can subsequent adaptive IoT honeypot deployments be optimised for immediate threats?

Given these questions the following hypotheses can be proposed:

1. A honeypot using a unique state action space formalism can learn from automated and repetitive malware to prolong interaction. An adaptive honeypot for automated and repetitive malware (HARM) exploits the actual characteristics of malware, namely automation and repetition, to determine the best responses to attack commands.

2. Integrating reinforcement learning with state action space formalism for automated and repetitive malware, honeypots can covertly overcome detection and termination methods, realise entire attack sequences and capture larger datasets.

3. Reinforcement learning algorithms and policies can be evaluated using current captured datasets in a controlled environment. This allows for the optimisation of subsequent deployments for immediate malware threats.

Fig. 1.2 graphically represents the cyclic process of improving honeypot operations espoused by this thesis. It identifies the practical experiments used the explore each hypothesis.

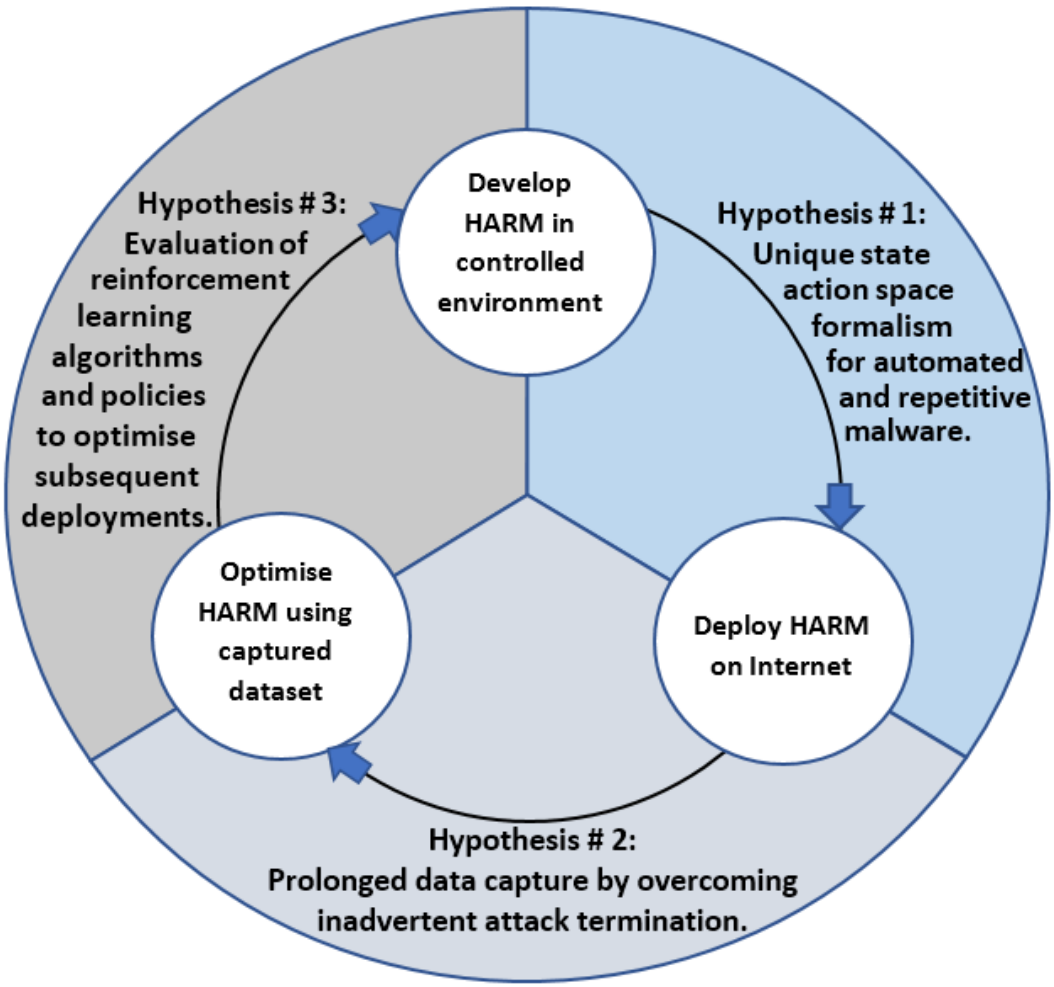


FIGURE 1.2: Practical Exploration of Hypotheses.

1.3 Thesis Structure

Chapter 2 links three evolving paradigms. Societal and industrial use of technology is linked with the emergence and variety of malware which gives rise to new methods of honeypot design. This chapter will examine the attack surfaces associated with new technological concepts. Malware developers exploit these creating new methods of propagation and compromise. Since 1993 [1] researchers have attempted to capture these methods. Honeypots emerged as a viable option and have evolved in conjunction with new technologies and malware. More recently honeypots have used machine learning techniques to analyse captured datasets retrospectively. Proactively, reinforcement learning has been used to interact with human attackers by using delay and insults to prolong interaction.

Chapter 3 sets out the methodologies used for this thesis. These methodologies detail the creation and deployment of two honeypots: an IoT honeypot and an adaptive honeypot. The former captures a dataset that demonstrated automation and repetition. This dataset informs the development of the latter. A **H**oneypot for **A**utomated and **R**epetitive **M**alware (HARM) uses reinforcement learning to learn the best responses when interacting with automated and repetitive attack sequences. This adaptive honeypot uses a novel state action space formalism designed to reward the learning agent for prolonging interaction with automated and repetitive malware.

Chapters 4, 5 and **6** present the results from hypotheses 1, 2 and 3 respectively:

1. HARM demonstrates improved adaptive abilities for automation and repetition, over previous research. It demonstrates this by clearly identifying the best actions whilst interacting with attack commands and converges towards a near optimal learning policy.

2. Honeypot deployments are vulnerable to being compromised themselves or being discovered as honeypots. Dedicated honeypot detection tools and methods can compromise the effectiveness of honeypots. Standard honeypots can also inadvertently terminate attack sequences with scripted responses. HARM can overcome these detection methods, conceal honeypot functionality and prolong attack interaction. In doing so it can continue to operate uninterrupted and capture larger datasets in an equivalent time period.

3. How big is enough? The captured dataset can be used to optimise the performance of HARM for immediate threats. Reinforcement learning algorithms and policies are assessed for their suitability to the problem domain of automated and repetitive malware. This contribution finds that an adaptive honeypot like HARM should have a time delimited deployment period. Evaluating learning

algorithms and policies in a controlled environment identifies redundant data capture, highlighting agile deployment methods.

Chapter 7 proposes a new framework for adaptive and agile honeypots. It examines the findings from hypotheses 1, 2 & 3 and discusses their implications for future honeypot development and deployment. It analyses the role of Seifert's taxonomy [11] and proposes changes for evolving technologies and malware. The new framework for future development and deployment of adaptive, agile honeypots emerges from these discussions.

Chapter 8 concludes the thesis with a summation of findings, limitations and the potential future work.

Fig. 1.3 graphically represents the layout of the thesis document.

LaTeXdevelopment environment for this document can be viewed here

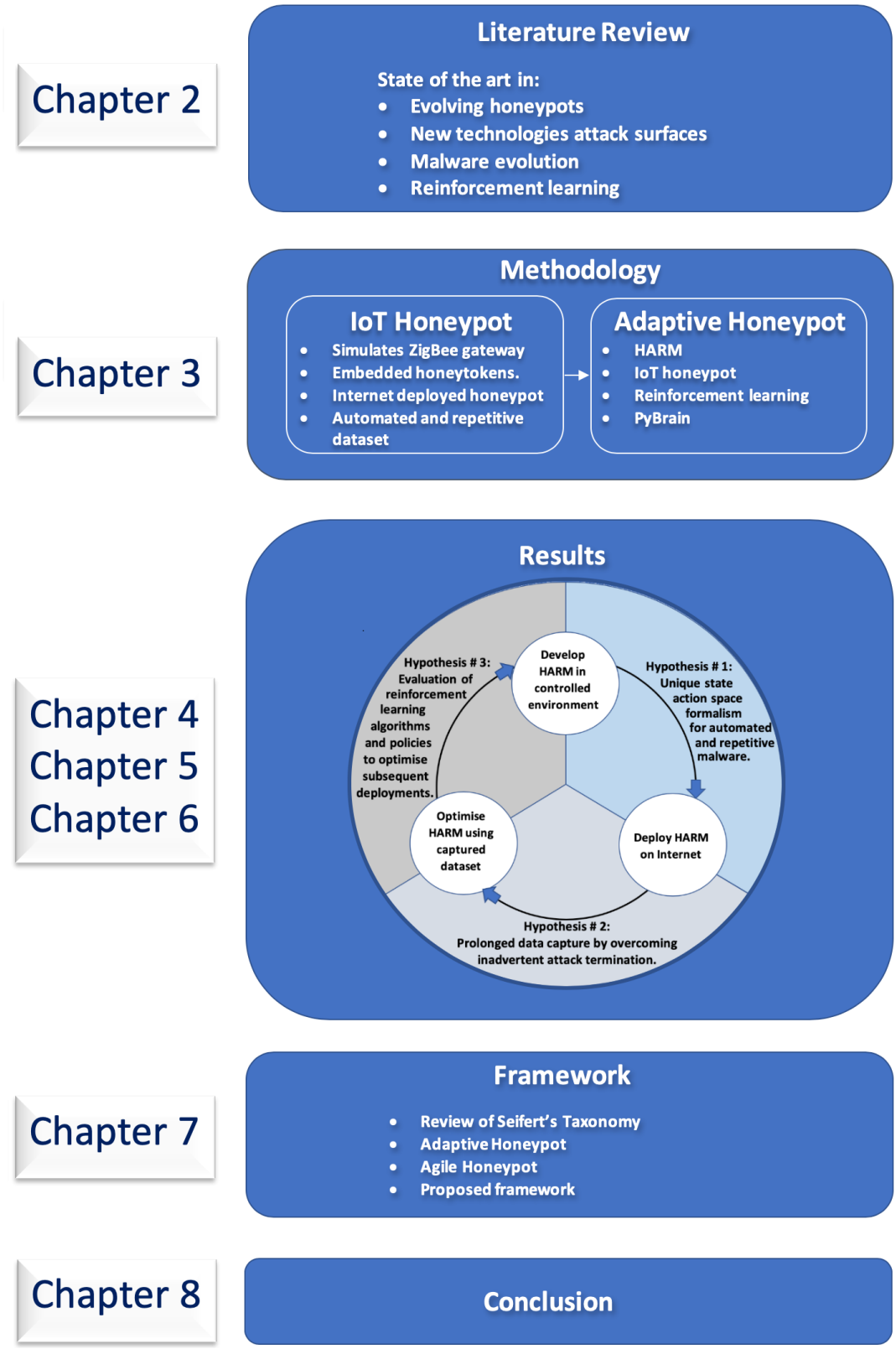


FIGURE 1.3: Structure of Thesis

1.4 Publications

1.4.1 Peer-reviewed journal publications

1. **Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware**, Seamus Dowling, Michael Schukat & Enda Barrett (2018) , Journal of Cyber Security Technology, 2:2, 75-91, DOI: https://doi.org/10.1080/23742917.2018.1495375

2. **Using Reinforcement Learning to Conceal Honeypot Functionality** Dowling, S., Schukat M., Barrett E. (2019) In: Brefeld U. et al. (eds) Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018. Lecture Notes in Computer Science, vol 11053. Springer, Cham. DOI: https://doi.org/10.1007/978-3-030-10997-4_21

3. **Self HARMing Malware: An Adaptive IoT Honeypot for Automated, Repetitive Malware**, Seamus Dowling, Michael Schukat, and Enda Barrett, 2019, In: https://onlinelibrary.wiley.com/journal/15320634 (Under review - See Appendix D.1)

4. **A New Framework for Adaptive and Agile Honeypots**, Seamus Dowling, Michael Schukat, and Enda Barrett, 2019, In: https://onlinelibrary.wiley.com/journal/22337326 (Under review - See Appendix D.2)

1.4.2 Peer-reviewed conference publications

1. **A ZigBee honeypot to assess IoT cyberattack behaviour**, Dowling, S., Schukat, M. and Melvin, H., 2017, In Signals and Systems Conference (ISSC), 28th Irish (pp. 1-6). IEEE.

2. **Using analysis of temporal variances within a honeypot dataset to better predict attack type probability**, Dowling, S., Schukat, M. and Melvin, H., 2017, In Internet Technology and Secured Transactions (ICITST), 12th International Conference for (pp. 349-354). IEEE.

3. **Data-centric framework for adaptive smart city honeynets**, Dowling, S., Schukat, M. and Melvin, H., 2017, In Smart City Symposium Prague (SCSP), 2017 (pp. 1-7). IEEE.

1.5 Talks

1. **Dowling, S.**, *Using open source machine learning tools and repositories*, MAYO-AI, GMIT Mayo Campus, 20/11/2018, mayo-ai.com

2. **Dowling, S.**, *Optimizing IoT Cyber Attack Analytics with Adaptive Honeynets*, NUIGalway UL Alliance, 7th Postgraduate Research Day, NUI Galway, 19/04/2017, nuigulresearchday2017.wordpress.com

3. **Dowling, S.**, *A Zigbee Honeypot to establish baseline data for wireless sensor networks cyberattacks*, GMIT Research Cycle Conference, GMIT, Galway, 01/05/2015, gmit.ie/gmit-research-cycle-conference-2015

4. **Dowling, S.**, *Evaluating cyber attack frequency on ZigBee communications with online Honeynet* Intel Ireland Research Conference, Trinity College, Dublin, 18/11/2014, Intel-Ireland-Research-Conference

1.6 Media

Dowling, S., *AI Conference in GMIT – an insight on AI and the West*, 29/10/2018, irishtechnews.ie/ai-conference-in-gmit-an-insight-on-ai-and-the-west

Chapter 2

Literature Review

2.1 Technology, Society and Industry

Technology is continually finding methods to enhance societal living in meaningful ways. Advances in technology help industry to perform existing tasks more efficiently, directly impacting other industries and ultimately society. The popularity of using personal computers was established during the 1980's. These standalone, fixed terminal devices introduced the mass public to computing concepts such as operating systems, programming, early office applications and gaming. In the late 1960's, *ARPANET* developed protocols for connecting remote computers for military purposes [12]. This involved a handful of nodes connected across existing telecommunications infrastructure. In the late 1980's, the *world wide web* (WWW) became more mainstream with *hypertext markup language* (HTML). Since then web population and means of access has increased [13]. Retrospectively the nomenclature of Web X.0 is used to demark the evolution of the WWW. Web 1.0 was and information portal whereby users could access static pages without being able to post comments of give feedback to the website in any way. If Web 1.0 was 'read-only' then 2.0 was 'read-write' It allowed end users to have more interaction with other users by sharing information on websites and other platforms. It provided for social interaction in new and meaningful ways, giving rise to social media as a new technological paradigm. Web 3.0, a semantic web, explores human to machine and machine to machine interactions. Data collected and analysed facilitates decision making for humans and other machines. This smarter web uses artificial intelligence (AI) to builds self-aware systems capable of making independent decisions. IoT and Internet-of-Everything (IoE) are core concepts for Web 3.0. This WWW evolution is invariably linked with access and connectivity to the Internet. From dial-up modems and *Integrated Services Digital Network* (ISDN) to broadband and mobile services, data and telecommunication advancements have acted as catalysts. Figs. 2.1 and 2.2 demonstrate the growth and demand in fixed line and mobile data subscriptions respectively [14].

It is possible to measure the categorical changes based on region, gender, usage and access [13]. It shows for example that for the first time, people accessing

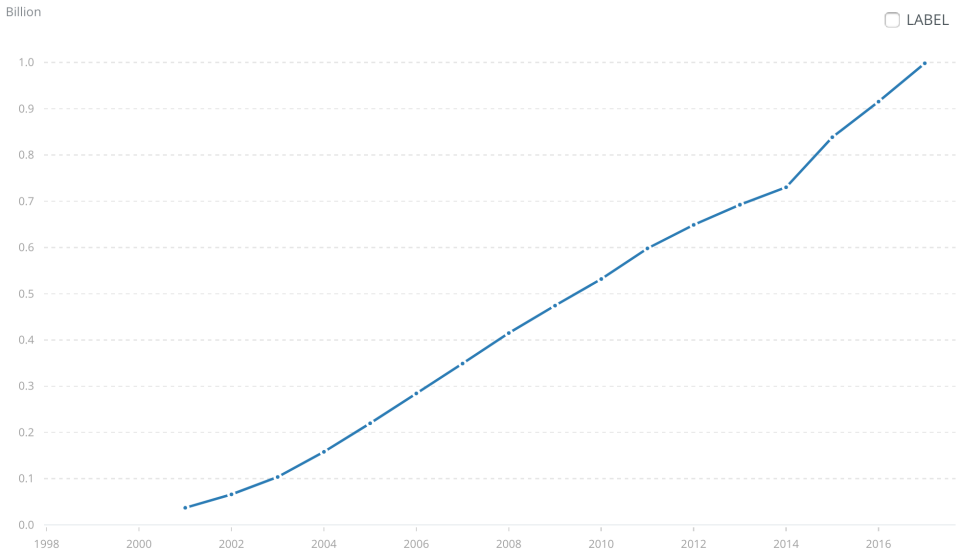


FIGURE 2.1: Global growth in fixed broadband subscriptions 1998-2017.

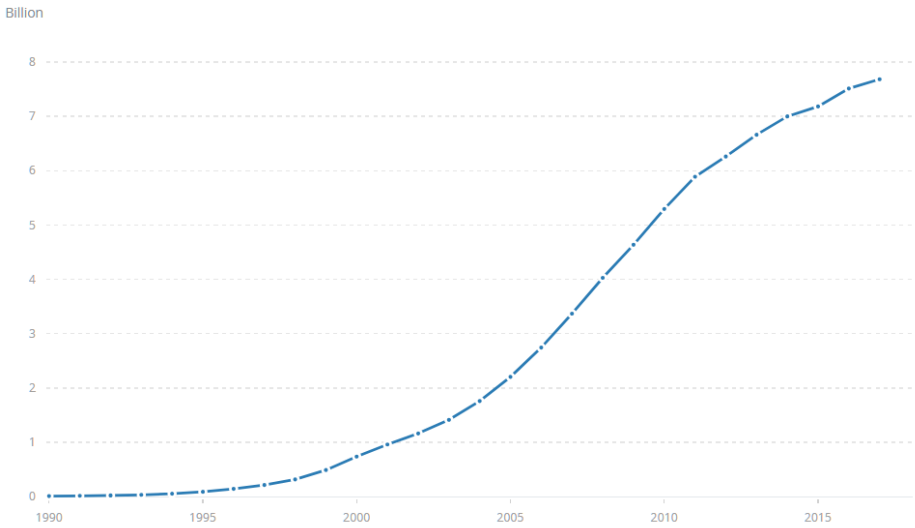


FIGURE 2.2: Global growth in mobile broadband subscriptions 1990-2017.

the Internet on mobile devices is greater than using desktop devices. Mobile Internet services have also evolved since the early 1980s. First generation (1G) offered 2.4kbps rising to 64kbps for second generation (2G). 2G development was overseen by *Global Systems for Mobile Communications* (GSM). GSM heralded the beginning of new projects such as *Universal Mobile Telecommunications Systems* (UMTS), and new partnerships *Third Generation Partnership Projects* (3GPP). 3G offered data services from 144kbps in transit to 2Mbps stationary. 4G is a *Long Term Evolution* (LTE) wireless broadband service developed by 3GPP, offering high speed voice, video and data services of up to 100Mbps download. These projects and partnerships have established the platform for the development and rollout of 5G services in the near future. As the Internet and WWW evolves, society will be connected in more valuable and relevant ways. Today people mostly communicate through devices such as smartphones, personal computers, tablets etc. However Web 3.0 and beyond will proliferate the linkages of society, industry and machines. As new advances in technology emerge, processing power will increase while storage and battery capacity will be available at a relatively low cost. This enables the development of small-scale electronic devices with communication, computing and identification capabilities, which can then be attached to common objects. These devices could then be used to produce a wide range of novel applications and services, which interact with each other. IoT as a concept is relatively simple. It concerns enabling devices and everyday objects to have the ability to connect with each other over the Internet in order to communicate with other connected devices. IoT can be viewed as a distributed system of networks, made up of a growing number of smart objects, interacting with each other and producing information. The ability to interact is made possible through the presence of devices with the capability to sense physical telemetries and translate them into information data.

IoT is still in its formative stages. Yet, in spite of this, there exists a large collection of articles, papers, publications and conference proceedings dedicated to research in the area [15]. Over 75% of this research concentrates on the technology and application of IoT, with the majority of the remaining research examining ethical legal and future elements. Kevin Ashton is credited with coining the term *Internet of Things* [16] at a presentation on supply chain management using *radio frequency identification* (RFID) [17]. RFID is a short-range communications technology using tags to send data to a receiver. The tags are programmed to contain small amounts of data pertaining to the attached 'thing'. RFID is an established technology in manufacturing and industrial settings, tracking products as individual items or as collections, around the globe. *Near field communication* (NFC) [18] builds upon RFID by allowing devices communicate with each other when brought together in close proximity. Credit card transaction and guided tours are some example of everyday uses. Bluetooth [19] is a ubiquitous standard use in mobile phones, car communications and audio. RFID, NFC and Bluetooth devices are already components of IoT providing identification and data transmission. However the component parts cannot perform

further complex operations and hence limit their role. IoT needs things that can do more than just identify an item in a network. It also needs to be able to initiate communications and interact and collaborate with other things. Whitmore et al performed a survey of IoT research and presented IoT applications research as having 4 categories: Smart Infrastructure, Healthcare, Supply Chains/Logistic and Social [15]. Other research presents alternative categorisations [20][21]. *IoT Analytics* [22] uses inputs from Google searches, Twitter, LinkedIn and news items to come up with the top IoT applications based on segment activity and popularity in 2018. Fig. 2.3 indicates the growth in IoT projects globally and regionally. All of the projects in the figure impact on society and industry, collectively and individually.

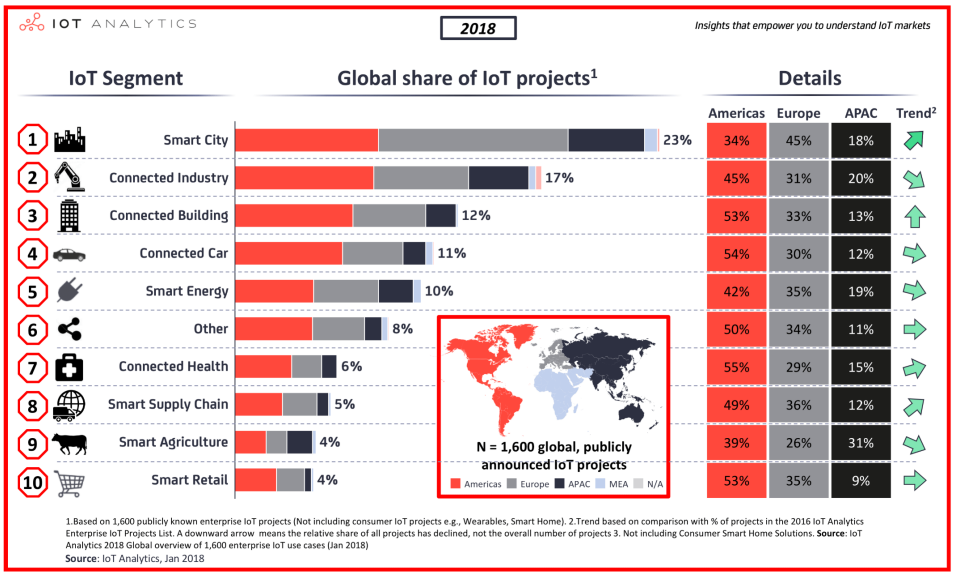


FIGURE 2.3: Global share of IoT projects [22].

A simpler categorisation of IoT is as follows:

- *Smart Cities:* The biggest impact made by IoT applications on society is with the deployment of sensors to monitor city ecosystems. Optimising power grids, controlling traffic flows, monitoring pollution and urban environmental ecosystems are some examples of IoT managing cities assets for the betterment of its population.

- *Connected Industry:* RFID was initially deployed to track large industrial assets and packets. Sensors can be deployed to provide information for *building anagement systems* (BMS). Machine-to-Machine (M2M) and CPS were strongly associated with industry before IoT introduced web connectivity and reachability. Other applications in industry include product shelf life monitoring, structural monitoring, preventative maintenance, smart factories, robotics, augmented reality and much more.

- *Connected Building:* Monitoring utilities, security and environmental variables (CO_2, smoke) are regular features of home IoT. As well as monitoring devices, interactive deployments controlling heating, lighting and appliances are widespread. BMS embed small heterogeneous wireless devices that collaborate with other devices and systems on a home site to benefit occupants in a discrete and transparent manner. Heating, ventilation and air conditioning (HVAC) is an example of BMS being used to control energy costs.

- *Environment:* IoT and sensor deployment coexists very well with environmental monitoring. Conditions such as temperature, light, disease, pollution, wind, rainfall and many others can be measures and relayed to data collection sites. Deployment can be discrete, both for urban and remote locations. Cheap, disposable miniature devices can be deployed in remote locations with a finite power source. At risk locations such as costal or earthquake and volcanic zones can relay vast amounts of data for real-time analysis and act as early warning indicators.

- *Health:* Body area networks can provide an all-in-one biological telemetry measurement suite to relay medical information to a central location. This can aid medical step down facilities or remote patient care. Other examples include medication management and ambient assisted living for vulnerable people. Existing health and fitness real-time monitoring already exists for elite and professional sportspeople.

Directly impacting individuals in society is the concept of a smart home. Energy companies are introducing functionality to control home heating remotely. Other home automation concepts remove implicit human instruction such as use automated heating, lighting, security, on-demand video services and the much maligned smart fridge. However IoT is more ambitious than just smart homes. In conjunction with IIoT, it can be scaled up to smart cities. Through a network of combined devices smart cities can for example, monitor traffic flow, energy usage, waste disposal, emergency services, pollution etc. IoT and IIoT concepts requires the deployment of devices that are configured to collect data, communicate and interact with other devices. This type of inter machine communication is not a new concept. M2M communication originated in the 1970's enabling end devices to captured events as data and communicate to an application that transformed this data into meaningful information. This communication exists in a closed loop, secure from external interference. M2M implies non-IP end-to-end communication all the way to the end device whereas IoT implies some IP communication. CPS is a term often used interchangeably with IoT. However the term implies cyber and physical systems linked together, IoT providing a medium between systems. CPS not only communicates with each other but are also autonomous entities in their own right, and are required to communicate and control one another collaboratively. Sensor networks are dynamic systems composed of large numbers of sensors that produce, consume, process and

communicate information and data [23]. These networks gather telemetry measurements on their environment and provide the mechanism to relay this information back to a collection point or server, creating *wireless sensor networks* (WSN). WSN nodes require electronic equipment capable of monitoring its environment using attached sensors. Data captured by these sensors require a communications stack built onto the electronics, to transmit and receive as part of a network and provide reachability to all connected devices. Data, security and communications all differ for smart objects within these components. IEEE defines two device types that can participate in an 802.15.4 network: a full-function device (FFD) and a reduced-function device (RFD).

> "An FFD is a device that is capable of serving as a personal area network (PAN) coordinator or a coordinator. An RFD is a device that is not capable of serving as either a PAN coordinator or a coordinator. An RFD is intended for applications that are extremely simple, such as a light switch or a passive infrared sensor; it does not have the need to send large amounts of data and only associates with a single FFD at a time. Consequently, the RFD can be implemented using minimal resources and memory capacity" [24].

IoT will develop incrementally by adding to existing technologies rather than creating a new class of system [25]. Table 2.1 presents the system level features that are required to support the evolution of IoT.

The table presents the importance of heterogeneous devices interacting and communicating together in ad-hoc networks. This concept impacts greatly on the physical dimensions of the underlying electronics and power consumption. WSN nodes require electronic equipment capable of monitoring its environment using attached sensors. Data captured by these sensors require a communications stack built onto the electronics, to transmit and receive as part of a network [26]. Heterogeneous nodes require complete protocol stacks to ensure interoperability. On top of this, middleware [27] will ensure that the hardware layers can communicate with application layers created by developers to access and exploit IoT benefits. Gubbi et al outlines the uses of Uniform Resource Name (URN) and Uniform Resource Locators (URLs) in middleware's XML schema for unique addressing and accessibility [20]. Transmission Control Protocol/Internet Protocol (TCP/IP) is the basic protocol or communication language of the Internet and can be implemented for WSNs [28]. Transmission Control Protocol (TCP) which is the higher layer, manages the assembly of files into smaller packets which are transmitted over the Internet and are reassembled upon arrival at its destination. The Internet Protocol (IP) is the lower layer protocol which manages the address of each packet to ensure that it is delivered to the precise destination. Each gateway device on the network will check this address to gain its destination information. Internet Protocol is the standard network layer protocol of the internet allowing communication between different networks.

TABLE 2.1: System level issues for IoT

Feature	Comment
Device Heterogeneity	IoT devices will vary from hardware, computing, architecture and communication. Interoperability is therefore a key issue.
Scalability	Issues such as addressing, data communication and information management are critical as IoT deployments grow
Ubiquitous Data Exchange	Smart objects will need to communicate to create networks and connect to central collection point. The wireless spectrum will require examination.
Energy optimised solutions	Computation and communication will drain the battery life of self-contained nodes. Better solutions are required to ensure longevity.
Localisation and tracking capabilities	It will be necessary to locate and track devices in the physical realm
Self-organising capabilities	Nodes will organise themselves into ad-hoc networks depending on the environment to ensure distributed data communication.
Semantic interoperability	Will enable IoT applications to support automated reasoning
Embedded Security	Security should be considered a key system level property to ensure wide scale adoption of IoT

In order for a network to be accessible from the internet there must be a router that complies with this protocol. Internet Protocol Version 4 has performed relatively well until now. However the introduction of all the extra devices connected to the Internet added to the strain on the addressing capabilities of IPV4. The *Internet Engineering Task Force* (IETF) began working on proposals for the successor of IPv4 in 1993 giving recommendations for the introduction of IPv6 [29]. The protocol stack, battery life and electronic physical dimensions will impact greatly on deciding what services are mandatory or optional. A characteristic of WSN deployments and the realisation of IoT is the implementation of distributed systems architecture. Inherent networking concepts such as routing, flow control and synchronisation are required [21]. These concepts will become issues as WSNs scale into larger entities. Larger volumes of data will require transmission placing a constraint on bandwidth. A single node failure can cause interrupted communication to that node, a group of nodes or an entire network [30]. IoT provides reachability to all connected devices. Sensors and things are web enabled and web facing to facilitate communication and interaction. Security of communications and data, and access and participation are

critical components to IoT deployments. Encryption techniques such as public key infrastructure (PKI) [31] are often too power and resource hungry for IoT devices. Authentication and joining a sensor network requires establishment of connectivity between genuine nodes. In ZigBee for example, a coordinator node has a role as a trust centre to share link and network keys to end devices. Link keys are shared between two communicating devices only [32]. A network key is a global key used by all devices in a *wireless personal area network* (WPAN). Some sharing methods require these keys to be transported across the WPAN over-the-air; sometimes keys are hardcoded in the device.

With such a diversity of smart objects, interoperability will require collaboration and compromise. Communications capabilities for smart objects will range from very basic to highly complex. Security measures implemented will depend on the objects processing capabilities, leaving some objects more vulnerable than others. The value of the data processed or stored also depends on the physical characteristics of the smart object. As IoT projects continue to expand, so too does the diversity of deployed smart objects. Security of communications and data, and access and participation are critical components to IoT deployments. Large volumes of big data will be processed by IoT deployments. Concepts such as smart cities give rise to issues concerning the privacy of their citizens and the use of data [33]. Information stored and communicated about location, social activity and profiling need to be secure to engender trust and promote the use of smart city applications. The diversity of data from objects and citizens will require secure storage and meaningful integration [34]. All smart objects and user end devices will require the capabilities to securely exchange data between each other, coordinating devices, gateways and subsequent storage locations. The diversity of technology required in a smart city system raises the question of standardization. Whereas certain efforts have been made to define IoT architectural reference models [35], standardization is very much in an embryonic phase for Smart Cities. To address this issue, the U.S. Department of Commerce's National Institute of Standards and Technology (NIST) along with international bodies such as the European Union created the *International Technical Working Group on IoT-Enabled Smart City* (IESCity) [36]. Current smart city projects are bespoke and not interoperable or portable across cities. Various consortia are attempting to define architectural frameworks. IES-City group are in the process of producing a consensus framework that will identify common architectural features to facilitate smart city projects.

All objects and communication channels are open to probing and compromise. The US Department of Homeland Security produced a risk assessment for CPS in a smart city [37]. It highlighted potential vulnerabilities for CPS or other related technologies as integral parts of the fabric of smart city infrastructure. The Industrial IoT Consortium published a security framework and an approach to assess cybersecurity in IIoT systems [38]. WSNs can be a target of cyber attacks as security may

not be feasible on constrained nodes. Security of a smart object depends on embedded security, if any and it's ability to defend against cyber attacks. The complexity of communications available on a smart object will depend on the functionality of the device and physical resources available. Constrained and unconstrained IoT devices will implement different protocols depending on these physical resources. This increases the diversity of attack surfaces available to malware developers. The criticality of data collected or transmitted by IoT objects is very diverse. Data communicated to a street light is minimal compared to the data collected and transmitted by a FFD IIoT actuator. Securing this data is essential to engender trust amongst smart city citizens who value their privacy. Individual and collective societal data can be compromised. Mobile phone technology and social media shifts the traditional concept of smart city from industrial concepts to holistic living. Irrespective of individual, societal or industrial IoT concepts, the security and privacy of data is paramount.

2.2 Malware

It is a great irony that a botnet provided a census of connected routers on the Internet. The Carna botnet scanned the IPv4 address space to create an image of fixed line Internet connectivity [39]. This automated and repetitive program globally propagated and compromised devices, predominately routers, to measure the extent of Internet access.

The concept of reproducible computer code originated in 1949 [40]. This theory involved programs self-replicating and passing this code onto the new programs. Fred Cohen coined the term *virus* in 1987, designing a computer program that could infect a computer, make a copy of itself and spread to other machines [41]. In the 1980's the connected world of ARPANET and personal computers (PC) was evolving. The opportunity to create malicious software was also evolving. The self-replicating programs proposed in 1949 now had a means to propagate to other sites. The *Creeper* worm infected ARPANET terminals, posted a message and opened new connections to other terminals [42]. Although no damage was done, this irritant demonstrated that computer code could use automation and repetition to infect connected devices. Using these methods, more malicious software emerged. PC popularity and the MS-DOS operating system were targeted with *Brain* virus and connection to the ARPANET was exploited with the *Morris* worm [43]. HTML facilitated the creation and expansion of the WWW in the early 1990's. This expansion of connected computers facilitated the spread of malicious software. Evolutionary Web X.0 paradigms created new methods and variants of malware. The following is a list of some key milestones in the evolution of the WWW and its resultant malware:

- *1991*:- The Internet public became cognizant of the threat of viruses with the spread of *Michelangelo*. This virus raised the awareness of the dangers of viruses and heralded the introduction of anti-virus software.

- *1999*:- E-mail provides an efficient delivery system for malicious code with *Melissa*, *ILOVEYOU* and *Anna Kournikova* causing widespread infection across the Internet.
- *2003*:- SQL vulnerabilities are exposed as *Slammer* and *Conficker* used to create global DDoS attacks.
- *2005*:- Social Media platforms are used to spread *Koobface*.
- *2007*:- *ZBot* becomes the most successful botnet of all time, infecting Windows machines and designed to steal banking information.
- *2010*:- IIoT and industrial control systems (ICS) are targeted with the high profile *Stuxnet* worm designed to compromise the Iranian nuclear program.
- *2013*:- *Ransomware* emerged as revenue generating malware with *Cryptolocker* encrypting users data on their machines. A ransom, usually in *Bitcoin*, has to be paid before the decryption key is given. In 2017 *WannaCry* propagated globally, affecting end users, organisations and goverment bodies in over 150 countries.
- *2016*:- *Mirai* botnet was created by targeting unsecured IoT devices. The resultant DDoS attack paralysed high profile web services.

2.2.1 Malware Methods

The original *Creeper* worm used automated and repetitive method to propagate and infect terminals. These characteristics are shared with the various types of malware in the above list. The human factor involves designing, coding and launching the malware. Infected machines may communicate with the command and control (C&C) which is also controlled by a human *botmaster*. Human naivety contributes to infection as end users unwittingly enable content. But predominately, global infection uses automation and repetition methods. Botnets provide a mechanism for global propagation of cyber attack infection and control. They are defined as large networks of compromised machines used to carry out further attacks [44]. These botnets are under the control of a single C&C. A typical botnet attack will consist of 2 sets of IP addresses. The first set of IPs are the compromised hosts. These are everyday compromised machines that are inadvertently participating in an attack. The second set of IPs are the C&Cs. These are the sources from which the desired malware is downloaded. There are 3 methods of communications between C&C and compromised host [45]:

- IRC (Internet Relay Chat) based model using push commands from C&C.
- HTTP based model using pull commands from hosts.
- P2P based model where bots use peer-to-peer communications.

Botmasters obfuscate visibility by changing the C&C connection channel. They use Dynamic DNS (DDNS) for botnet communication, allowing them to shut down a C&C server on discovery and start up a new server for uninterrupted attack service. Tracking botnets is possible by examining methods of communication and establishing patterns. C&C communications with botnets exhibit different characteristics from legitimate traffic [46]. Honeypots are valuable tools when analysing the behaviour of automated and repetitive malware. Honeypot deployments are longitudinal and the resultant dataset contains large amounts of repetitive data. This allows researchers to ascertains patterns in the behaviour over a period of time. Global honeypots often operate for long periods with a view to collecting large datasets. Temporal variances and IP blocks provide interesting insight into the propagation methods used by bots and botnets. The diurnal characteristic of end users turning on their machines in the morning, turning them off in the evening can be modelled [47]. Bias for regional online populations can also be established [48][49].

As technology and malware evolves, it is important that modern honeypots are deployed to capture new and relevant datasets. The temporal and regional variances from a 3-month IoT honeypot deployment, indicate that similar methods are used to compromise and propagate [50]. From this dataset, Figs. 2.4 (temporal) and 2.5 (spatial) shows compromised host source activity from distinct global regions of Asia, Americas and Europe. Rotating the graph in fig. 2.4 and overlaying it on fig. 2.5, it is possible to visualise this temporal pattern and associated volume of attacks. Fig. 2.6 therefore gives a global and regional view of end device security.

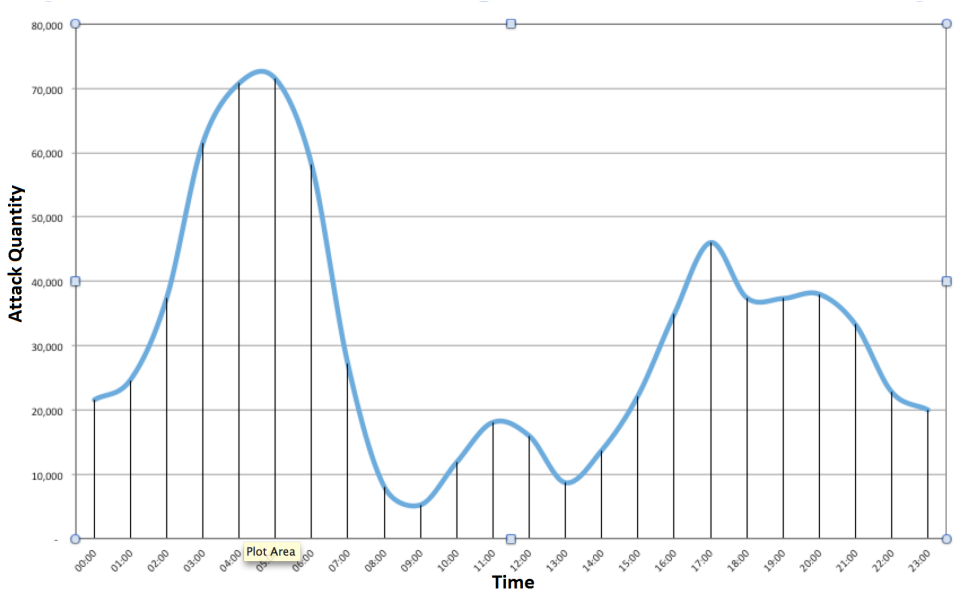


FIGURE 2.4: Temporal attack pattern on IoT honeypot [50].

FIGURE 2.5: Geo-spatial distribution of attacking IPs [50].

FIGURE 2.6: Global view of end device security [50].

2.2.2 IoT Malware Attack Surfaces

Successful malware diverge into new variants using similar methods [9]. The Mirai botnet attacked and compromised routers, firewalls, cameras, DVR and other media devices. The top attacking vectors were CWMP (28.30%), Telnet (26.44%), HTTPS (19.13%), FTP (17.82%) and SSH (8.31%). The attack surface available to malware developers is increasing rapidly with the deployment of new IoT projects. With such a diversity of smart objects, interoperability will require collaboration and compromise. Communications capabilities for smart objects will range from very basic to highly complex. Security measures implemented will depend on the objects processing capabilities, leaving some objects more vulnerable than others. The value of the data processed or stored also depends on the physical characteristics of the smart object. As IoT projects continue to expand, so too does the diversity of deployed smart objects. Security of communications and data, and access and participation are critical components to IoT deployments. Encryption techniques such as PKI are

TABLE 2.2: FFD and RFD Protocols

Data	XML, HTML, EXI
Application and Transport	DDS, CoAP, AMQP, MQTT, XMPP HTTP, REST, TCP, UDP, SSH
Infrastructure	RPL, 6LoWPAN, IPv4, IPv6, 802.15.4, ZigBee, Bluetooth, LoraWAN, 3G, 4G, LTE-A, EPCGlobal, Z-Wave, WiFi, RFID

often too power and resource hungry for IoT devices therefore alternatives need to be considered [51]. Table 2.2 presents an example of protocols used at data, application, transport and infrastructure layers for FFD and RFD devices.

The heterogeneous aspect of smart IoT devices and their protocols, creates a range of potential attack vectors. The data collected and transmitted on these smart devices could be irrelevant or critical. The required security and complexity of communications depend on their application and physical resources. To bring order to this diversity a data-centric view of societal smart devices is created [52]. Three key values for this data-centric view are:

- *Data Complexity*: The complexity of communications available on a smart object will depend on the functionality of the device, it resources and its position in a smart city system. Communication protocols can be separated into Application Protocols, Service Discovery Protocols and Infrastructure Protocols [53]. Constrained and unconstrained smart objects will implement different protocols depending on the physical resources available.

- *Data Security*: The diversity of IoT devices and resources produce security vulnerabilities [54]. These vulnerabilities create tangible weaknesses in IoT deployments. Irrespective of whether the smart object is FFD or RFD, implementing security measures to mitigate all vulnerabilities in a IoT system is not feasible. For many constrained smart objects, it will not be possible to implement robust security even if data value is sensitive and/or data exchange is complex.

- *Data Criticality*: Large scale IoT project such as smart cities, will increase the number of devices that hold or communicate sensitive data. Securing this data is essential to engender trust amongst smart city citizens who value their privacy. Individual and collective societal data can be compromised. Mobile phone technology and social media shifts the traditional concept of smart city from industrial concepts to holistic living. Irrespective of individual, societal or industrial smart city concepts, the value and privacy of data is paramount. However, the data value associated with compromising different smart object will differ greatly. For example, the data value associated with a street light will differ from the data value associated with a BMS gateway.

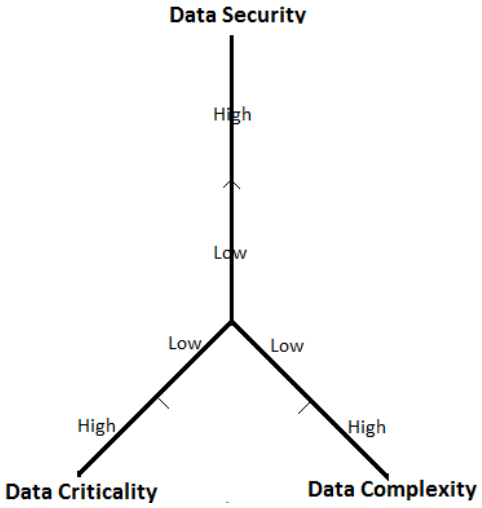
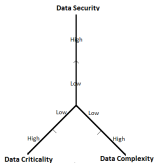
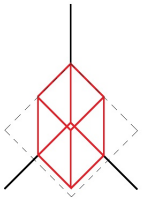
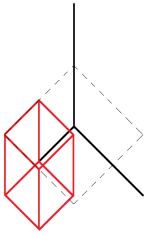
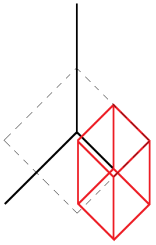
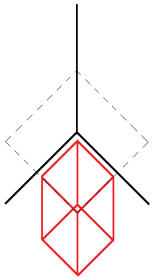
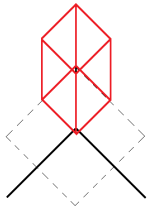
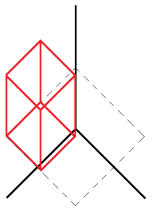
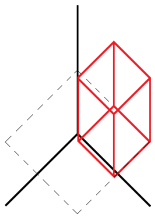
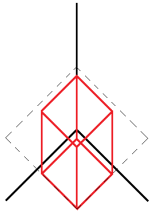


FIGURE 2.7: Cartesian Representation of Data-Centric Model [52].

FFD can be considered to be resource rich to varying levels. They have more CPU, power and memory. Because of this they can implement more robust security, have more communication capabilities and can process, store and transmit complex data streams. RFD are considered resource poor and therefore have reduced capabilities than a FFD. The three values of complexity, security and criticality can exist for each IoT device to varying degrees. To give them a basic value, fig. 2.7 presents a Cartesian coordinate system with *high* and *low* values assigned.

Table 2.3 presents the heterogeneous nature of IoT objects using this data-centric view and can be considered as a contribution to the state of the art. The three key values are given *high* and *low* options. It suggests communication protocols used by sample IoT objects. At the low end of the range a streetlight will possess little complexity or embedded security, reflecting the nature of its functionality. At the upper end of the range, an IIoT Actuator will have be a FFD with complex and secure communications reflecting its purpose. Across the entire range in the table, all IoT devices are susceptible to malware attacks contributing to the growth of the available attack surface.

TABLE 2.3: Data-Centric View of Smart IoT Objects

								
Complexity	low	low	low	low	high	high	high	high
Security	low	low	high	high	low	low	high	high
Criticality	low	high	low	high	low	high	low	high
Attack Vector	IPv6	IPv6	IPv6	IPv6	IPv6	LoraWAN	HTTPS	HTTPS
Examples		UDP	TCP	802.15.4	WiFi	IPv6	CoAP	REST
			SSH	ZigBee		6LoWPAN	ZigBee	MQTT
							IPv4	IPv6/v6
							SSH	SSH
Examples	Street Light	Traffic Light	HVAC Sensor	Medical Sensor	LiFi Light	RFD CPS	ZigBee WSN	IIoT Valve
			Gateway			Actuator	Gateway	Actuator

2.3 Honeypots

2.3.1 Standard Honeypots

Since their introduction in the 1990s, honeypots have evolved to meet the changing landscape of cyber threats. Just as IoT deployments gather pace, so to does the deployment of bots and malicious code targeting IoT end devices [7]. It is not uncommon for Internet facing devices such as cameras, digital recorders and remote controlled toys to partake in distributed denial of service attacks (DDoS). In 1992, USENIX conferences presented work on captured 'crackers' activities [1]. Dummy machines were deployed to lure and monitor the activity of the attackers. In his seminal book, *Honeypots: Tracking Hackers*, Spitzner defined honeypot as being a 'security resource whose value lies in being probed, attacked or compromised' [55]. Scientific research into honeypots and honeynets has increased since then. Provos [6] in 2003 presented *Honeyd*, an easy to deploy, low risk honeypot. It details how to deploy virtual honeypots with different IPs safely. This is important. Honeyd is considered a low interaction honeypot, gathering information on the activities of an attacker in a virtual, confined space. At this point in their evolution, Honeypots could be compromised and inadvertently partake in subsequent attacks [56]. Therefore, a decision has to be made concerning the trade off between gathering relevant attack data and protecting subsequent, inadvertent attacks emanating from the honeypot itself. Terminology such as low-interaction, medium interaction and high-interaction came into honeypot parlance. Low interaction honeypots (LiHP) capture base information such as IP addresses, port numbers and services. They will not permit the installation or execution of downloaded malware and are considered low risk in their implementation. Medium (MiHP) and high interaction honeypots (HiHP) give the attacker more scope for installing malware and exploring the operating system and file structure. This has the advantage of maintaining the interest of the attacker and capturing more information on their behaviour. HiHPs are real systems often mirroring live production systems. The obvious disadvantage of a more interactive honeypot is the potential for compromising the honeypot itself. Unwittingly, this could lead to access to the live production networks, or participation in subsequent attacks. *Honeyd* acted as a catalyst for the development of further low interaction honeypots. Nepenthes [57] and Argos [58] became very popular global honeypot tools. Faster networking and virtual technologies presented honeypot developers with a means of deploying high interaction honeypots and honeynets with low risk [59], isolating attack traffic from connected hardware and networks. A tangent of honeypot development was their use in a variety of security measures, such as DMZ intrusion detection and prevention. Dynamic honeypots are able to learn about the linked networks and integrate with existing communications, within these networks. Dynamic honeypots keep monitoring networks to detect any changes and collectively update, based on the detected change. Honeypots were proposed as intrusion detection systems (IDS), intrusion prevention systems (IPS) [60] and network IDS (NIDS)

[61]. Global projects such as Leurre [48] and Honeynet Project [62] were created and provide longitudinal analytical data. The Honeynet Project is a non-profit organisation dedicated to improving Internet security through developing open source tools, creating awareness of threats and conducting research on captured data. It has *chapters* around the world consisting of like minded security personnel conducting research on Internet threats and security. It presents many white paper on global honeypot projects in their *Know your Enemy* section. Honeypot and honeynet developments mirrors new advances information and communications technology. With each new advance, technology specific threats emerge. Honeypot technology has proven to be a valuable resource combating these threats by understanding their behaviour. The *European Network and Information Security Agency* is a "centre of network and information security expertise for the EU, its Member States, the private sector and Europe's citizens" [63]. In 2012, it produced the "Proactive Detection of Security Incidents" report which tested and evaluated over 30 existing honeypots. The report found difficulties with honeypot usage, documentation, software stability and developer support. It made recommendations for the future of honeypot development. *Surveys* of honeypot technology have appeared in 2003 [64], 2009 [65], 2011 [66], 2012 [67, 68], 2014 [69] and 2016 [70].

More recently, emerging networking technologies have facilitated new directions in honeypot deployment. Real-time visualisation of global attacks are provided by Deutsche Telekom Honeypot Project [71]. This current honeypot development community created *T-Pot*, which is used to collectively capture and visualise attacks on multiple well known honeypots. T-Pot also uses provides operating system level virtualisation for the well known honeypots in the form of *Docker*. Docker uses software containers to isolate honeypot tools. Containers can however communicate with each other through a single operating system kernal, making them much smaller than virtual machines [72]. Other advances in honeypot deployment include the use of *software defined networking* (SDN) to completely abstract bare metal from honeypot deployments. Intelligent solutions have been proposed to design and flexibility deploy next generation honeynets via SDN [73][74].

2.3.1.1 Taxonomies

The popularity of honeypot deployment raised the question of their role. To address this, Zhang [75] introduced a honeypot taxonomy. This taxonomy identifies security as the role or class of a honeypot, and could have *prevention*, *detection*, *reaction* or *research* as values. Deflecting an attacker away from a production network, by luring them into a honeypot, has a prevention value. Unauthorized activity on a honeypot is red flagged immediately providing a detection value. Designing a honeypot to maintain an attackers interest, by offering choices or tokens [76] has a reaction value. Finally the research value provides a crucial value of a honeypot by understanding the behavior and motivation of an attacker. As more devices connected and threats evolved on the Internet, Seifert introduced an updated taxonomy [11]. As can be

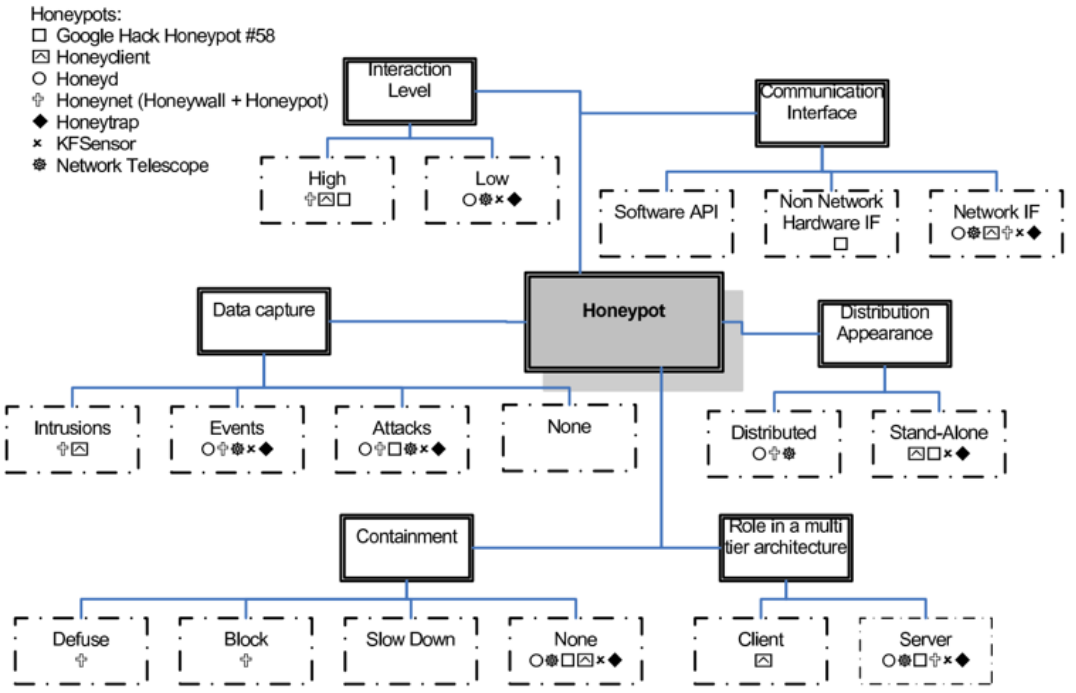


FIGURE 2.8: Seifert's Taxonomy of Honeypots [11].

seen in fig. 2.8, six core classes were proposed, each with their own set of values. Also included in the image is a legend to popular honeypots of that time. Table 2.4 briefly explains the purpose of these classes and values.
Seifert's taxonomy provided the framework for subsequent honeypot development. The six classes inform the development of honeypots:

- *Interaction*: Further honeypots were developed with low interaction levels. Low interaction honeypots emulate the network layer (IP stack). With the development of virtual workspaces, high interaction honeypots could be deployed as real systems [59] isolated from the bare metal and connected networks. *QEMU* and *VMWare* enable honeypot designers present a real environment to the attacker. Medium interaction honeypots emulates available services such as WINS, HTTP and FTP [57].

- *Data Capture*: The quality of the captured data can provide an excellent insight into the behaviour of attacker and attack traffic. Valli et al [77] presents findings on the operation and collection of data, on all taxonomy values, from a SSH honeypot. Demonstration of what can be achieved with honeypot data is thoroughly presented by Thonnard [78].

- *Containment*: Just how much interaction that can be provided can be seen in Wagener's self adaptive honeypot [79]. The author uses game theory to coerce attacker behaviour. Portokalidis [58] designed a honeypot called Argos, to slow down dissemination of new malware, such as worms, viruses, and bug exploits.

TABLE 2.4: Seifert's taxonomy for honeypot development

Class	**Value**	**Note**
Interaction Level	High	High degree of functionality
	Low	Low degree of functionality
Data Capture	Events	Collect data about changes in state
	Attacks	Collect data about malicious activity
	Intrusions	Collect data about security compromises
	None	Do not collect data
Containment	Block	Identify and block malicious activity
	Diffuse	Identify and mitigate against malicious activity
	Slow Down	Identify and hinder malicious activity
	None	No action taken
Distribution Appearance	Distributed	Honeypot is or appears to be composed of multiple systems
	Standalone	Honeypot is or appears to be one system
Communications Interface	Network Interface	Directly communicated with via a NIC
	Non-Network Interface	Directly communicated with via interface other than NIC (USB etc.)
	Software API	Honeypot can be interacted with via a software API (SSH, HTTP etc.)
Role in Multi-tiered Architecture	Client	Honeypot acts as a server.
	Server	Honeypot acts as a client.

- *Distribution*: Standalone Honeypots interact with an attacker and captures the activity. Keeping the attacker interested for longer provides better insight into attack behaviour. Large global distributed Honeynet such as Leurre [48] provided large datasets for analysis.
- *Communication*: Threats can originate from two sources, internal or external. External sources emanate from the Internet and will communicate and interact with a Honeypot via an open service such as HTTP, SSH etc. Internal threats exist 'on-site', have access to the internal network services. Honeypots designed to capture internally sourced threats, act as IDS [60] or NIDS [61].
- *Multi-tiered Model*: Honeypots and Honeynets can be deployed to emulate clients or servers. Servers listen for client requests; clients make requests to servers. This client-server-client communication model is popular in deployments such as Argos (server) and Honeyd (client).

Since then new taxonomies, frameworks and architectures have appeared for related technology concepts such as honeynets [80], SDN deployed honeypots [74] and IoT specific honeypots [81].

2.3.1.2 Visualization

Analysis of a captured dataset can be presented in the form of graphs, charts and plots. These convey the information more effectively. It also presents large datasets in a meaningful way for a target audience. Honeypots have backend databases for capturing attack information, depending on the interaction level. Some honeypots, such as *Kippo* embed their own visualization tools to represents the data with basic graphs and charts [82]. But the main function of honeypots is to capture the data. Retrospectively, this data can be downloaded, taken offsite and analyzed. Valli [77] used two open source tools, *Graphviz* and *Afterflow*, and presented a series of graphs showing the relationship between IP addresses occurring in the dataset. Thonnard [78] used graphs to represent topological relationships among clusters of attacks, based on similarity distances. Koniaris [83] developed and presented Kippo-graph, a visualization tool that can be installed in conjunction with Kippo. It is written in php and accesses Kippo's backend SQL database to represent the data as graphs and charts.

2.3.2 IoT Honeypots

The popularity of IoT deployments has attracted malware targeting end devices. Simple Internet facing objects such as digital recorders and remote controlled toys are involved in distributed denial of service attacks. The diversity of connected devices, coupled with available physical resources and functionality, make IoT deployments vulnerable. Embedded security can depend on bare metal constraints of FFDs and RFDs. Table 2.3 provides sample protocols representing attack vectors that may

be targeted by dedicated IoT malware. The suggested volume of IoT devices deployed in the near future [84] requires adoption of IPv6 into the IoT architectural stack. For example, WSNs form a key element of IoT. They need to be scalable independent networks able to communicate as part of a larger Internet. Common deployments of WSNs include the monitoring and control of military, environmental, home automation, healthcare, structural, industrial, agricultural and surveillance devices [85]. The ZigBee Alliance produces standards and specifications for short-range wireless technologies [86]. The alliance consists of interested members (commercial, educational and governmental) that work together to progress WPAN WSN technology. They create a set of standards and specifications to facilitate the manufacture of products for WPANs. These standards and specifications allow for product creation that can interoperate in M2M WSNs. WSNs can be created for home automation, building management, smart energy, personal healthcare management, military and other domains. In early 2013, ZigBee released the Smart Energy Profile 2.0 (SEP2). It utilises ZigBee IP specification to create an IPv6 enabled Smart Energy IP Stack that can now integrate with the Internet. The specifications provide a framework for developing products that utilize the IEEE 802.15.4 *lower layers*. IEEE 802.15.4 defines the physical layer (PHY) and medium access control sublayer (MAC) specifications for a low rate WPAN (LR-WPAN). The PHY layer details the frequency bands and channels therein, the associated bitrates and access modes. The MAC sublayer provides an interface between the physical layer and the higher layer protocols. The ZigBee specifications define the *upper layers* of ZigBee WSNs. These layers reside on top if IEEE 802.15.4 lower layers and together form the basis of ZigBee WPAN technology. ZigBee provides 3 Device Types: End Device, Router and Coordinator. With these device types, 3 types of topology can be created: Star, Tree and Mesh. A Star is a simple network with one coordinator connecting many end devices. Tree and mesh networks use router types to extend the network size.

IPv6 comfortably provides for potential IP demands and native IP communication of IoT. The increase in IPv6 adoption is increasing slowly. The success of other IP schemes such as NAT (network address translation) and DHCP (dynamic host control protocol) inadvertently act as a barrier to increased IPv6 uptake. Fig. 2.9 shows the organic growth of IPv6 adoption for the last decade. Schemes such as *dual stack* and *tunnelling* allow IPv4 and IPv6 to cohabit. Because of this new IPv6 versions of existing honeypots have started to be developed [87]. The trend is for malware to still predominately leverage IPv4 vlnerabilities. However recent DDoS attacks have targeted IPv6 devices in IPv6 deployments.

The continual evolution of industrial and societal use of technology has seen equivalent opportunities for exploitative malware and subsequent honeypot development. The success of the Mirai botnet in 2016 spawned multiple variants targeting IoT devices through various attack vectors [9]. Honeypots played an active part in capturing and analysing the Mirai structure [3]. Variants were captured on Cowrie, a version of the popular Kippo honeypot [82]. The effectiveness of honeypots for

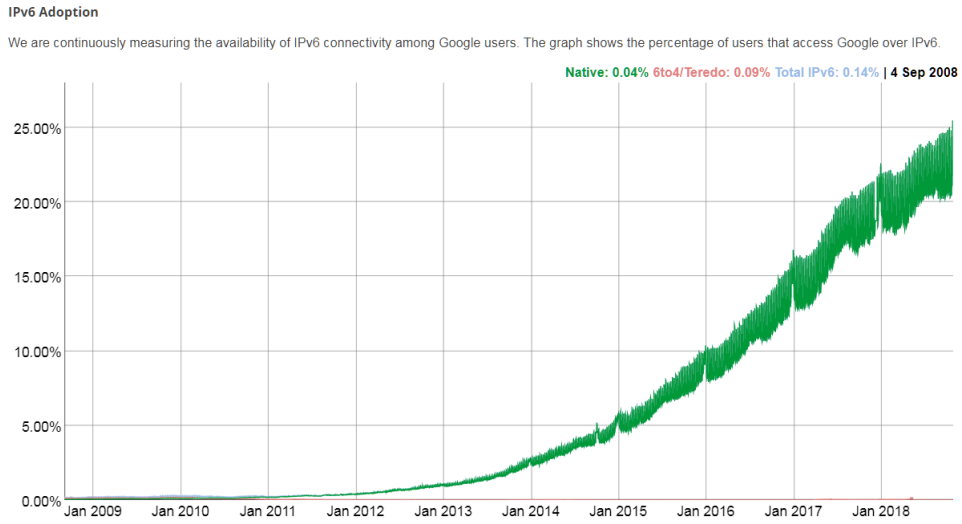


FIGURE 2.9: Global adoption of IPv6 [88].

WSNs is also explored [89]. New IoT versions of honeypots are being designed continuously to capture new IoT malware on different attack vectors. **IoTPot** is a bespoke honeypot designed to analyze malware attacks targeting IoT devices [7]. **ConPot** is a *SCADA* honeypot developed for critical IIoT architectures [90]. The repetitive and automated nature of the malware is visible on a smaller dataset from a IoT honeypot [10]. The captured dataset is freely available [91]. **ThingPot** [92], simulated vulnerable end devices. **IoTCandyJar** presented as a range of industrial, commercial and end devices which could be accessed across multiple attack vectors [81].

2.3.3 Honeypot Functionality

Virtualisation has allowed the security community to deploy high interaction, low risk honeypots [6], negating the inadvertent threat of contributing to subsequent cyber attacks [56]. Reliable standalone Honeypots can be deployed to adapt to attackers behaviour and capture good quality data for retrospective analysis. These can become part of a distributed global project such as *the Honeynet Project* and produce excellent datasets for analysis. Modelling techniques such as stochastic [49] and temporal [47] give valuable insights into cyber attack patterns. The design and deployment of a honeypot will impact on the collection of data [11]. If a Honeypot can engage an attacker for a longer time period, then its behaviour can be better understood. In his survey on *Advances and trends in Honeypot research*, Bringer [68], examined over 80 published papers on honeypots and identifies 5 major research categories:

- New types of honeypots to cope with emerging threats

- Utilizing honeypot output data
- Configuring honeypots to reduce costs and improve accuracy
- Counteracting honeypot detection
- Legal and ethical issues

Bringer's comprehensive review provides summaries of each honeypot paper or journal. It discusses the importance of these five categories for the evolution of honeypots to counter emerging malware. Design, configuration, counteracting detection, legal and ethical honeypot operations need to be considered prior to deployment. These points can be proactively controlled by the honeypot operator. Thereafter, the captured dataset can be effectively utilized for analysis and subsequent legal pursuits if required. Therefore, for the purpose of this thesis, these 5 categories can be further summarised into:

- Proactive functionality
- Retrospective analysis

Proactive functionality. Requires enhancements to existing honeypot operations to enable them to adapt to new malware methods, avoid detection and collect relevant datasets ethically and legally (See subsection 2.4.1).
Retrospective analysis. The effectiveness of malware behavioural analysis from a dataset, depends on the proactive functionality of the honeypot. A honeypot that is compromised or detected, fails to capture complete, relevant data. Equally, a standard honeypot that fails to adapt to new methods, will not capture the attack interactions giving complete insight into malware developers modus operandi (See subsection 2.4.2).
GIGO stands for *garbage in, garbage out*. A standard honeypot will not adapt to new malware and will capture incomplete truncated attack sequences. With no proactive functionality, the honeypot will capture *garbage in*. The resultant dataset for retrospective analysis is *garbage out*.

2.4 Machine Learning in Honeypots

The growth in artificial intelligence (AI) and machine learning (ML) libraries heralded renewed interest in deploying honeypots. Specifically, ML techniques have been used for *proactive functionality* and *retrospective analysis*. The diversity of machine learning algorithms can be applied to both of these categories. *Supervised learning* is ideally suited to retrospective analysis as the algorithm can learn from, or be trained by existing data. This learning is then used to classify new occurrences. Some examples of classifiers are *Linear classifiers*, *Naive Bayes*, *Support Vector Machines* (SVM), *Decision Trees* and *Random Forest*. Similarly *Unsupervised learning* can organise the data in different ways. Algorithms are left to their own devices to

create *clusters* or *associations*. Given a dataset, an unsupervised model can analyse the data to find structures within. *Reinforcement Learning* problems can generally be modelled using *Markov Decision Processes* (MDPs). In fact, reinforcement learning methods facilitate solutions to MDPs in the absence of a complete environmental model. This is particularly useful when dealing with real-world problems such as honeypots, as the model can often be unknown or difficult to approximate. MDPs are a particular mathematical framework suited to modelling decision-making under uncertainty.

2.4.1 Proactive Functionality

The quality of information gathered from Honeypot operations is dependent on the functionality of the honeypot itself. Low interaction Honeypots are emulators only capturing base data such as IP source [6]. High interaction honeypots are real systems often run on virtual platforms [82]. Ramsbrock models the attacker behaviour on a honeypot [93]. Once an attacker has compromised a honeypot, it will attempt to interact in a structured manner. Hardware and software properties are checked, to determine if the compromised host has further potential, or if the host is a virtualised environment [94]. Malware developers use this step to ascertain if the compromised host is actually a honeypot. An attacker may then modify the host system, including passwords. It then attempts to download, install and run malware to complete the compromise. A typical attack therefore will consist of 2 sets of IP addresses. The first set of IPs is the dumb agents. These are everyday compromised machines that are inadvertently participating in an attack. The diurnal pattern is of end users turning on their machines in the morning, turning them off in the evening [47]. The second set of IPs is the command and control centres. These are the sources from which the desired malware is downloaded. These can periodically change as they are discovered. The initial engagement for an attack post compromise, is to examine the hardware and software to determine if progression is relevant. On a live production system, this will return the underlying architecture, CPU, uptime, operating system, user privileges and further relevant information for deciding on continuation. On a honeypot engaging the attack sequence at this point prolongs activity. Capturing attack behaviour is the raison d'être of honeypot design. Attackers will attempt to download and execute code or install malware. In a desire to gather as much information as possible on attacker behaviour, a honeypot could allow the execution of malicious code. This code could itself contribute to further attacks, inadvertently propagating an attack vector [56]. This highlights the need for creating honeypots that prolong attacker interaction for an optimum time period, without breaching legal or ethical responsibilities. Researchers and honeypot developers need to ensure that methods used are legal and ethical [95]. Entrapment could be a mitigating factor when it comes to prosecution of an attacker. Conversely, the security developer could be liable if their honeypot inadvertently becomes involved in further attacks.

Real time decision-making by the honeypot makes it adaptive and able to facilitate longer periods of activity. Chronologically game theory is first to be proposed to define the reactive action of a honeypot towards attacker's behaviour [96]. A hierarchical probabilistic automaton is presented with the purpose of making a honeypot adaptive and autonomous. Data from a high interaction honeypot is streamed through the simulated adaptive honeypot. Using game theory concepts the honeypot lures further information from the attacker. Wagener uses reinforcement learning to extract as much information as possible about the intruder [79]. A honeypot called *Heliza* was developed to use **reinforcement learning** (RL) when engaging an attacker. The honeypot responded to attack interactions with actions such as allowing or blocking commands, substituting messages and insulting the attacker. MDP was used to model the honeypot environment and the RL algorithm SARSA (*state, action, reward, state, action*) rewards the learning agent. To compare its performance, Heliza uses two reward functions: the duration of the attack sequence and the cumulative number of attack transitions. It demonstrates a longer engagement with the attacker which results in uncovering further interactions and subsequent volume of transitions. PyBrain [97] is a ML library, written in Python and was used to facilitate RL functionality for Heliza.

Pauna [98] also presents an adaptive honeypot using similar **reinforcement learning** algorithms as seen in Heliza. *Reinforced adaptive SSH honeypot* (RASSH) also used PyBrain. It augments the responding actions by delaying the interaction response. Because of this, RASSH only compares its functionality with the reward function of cumulative transitions. A delay artificially inflates the duration of attack interactions negating a comparison with Heliza. The cumulative transitions are broadly in line with Heliza. Because Kippo [82] is used as the deployed honeypot, proposed improvements are scalability, localisation and learning capabilities. Pauna also proposes another method used to engage and attacker for longer [99]. **Case based reasoning** (CBR) uses cognitive sciences to model human behaviour. For AI systems, CBR can use prior cases as references points for characterising new scenarios or to suggest solutions for new circumstances [100]. The dataset from RASSH is streamed through a simulated *Case adaptive SSH honeypot* (CASSH) to prove the concept. JColibri is used to provide the Java based CBR functionality [101].

Using a **MDP**, Hayatle et al provides a strategy for the optimal response between for honeypot operators and botmasters [102]. They confirm that botmasters have an advantage over honeypot operators due to legal and ethical constraints. The proposed model prolongs attack activity while minimising legal liability. It allows the honeypot operator to maximise interactions with a botmaster which helps negate their advantage. Some of the elements of the MDP can be unknown. To help mitigate against the unknown elements associated with the botmaster/honeypot model, this research is then extended to **partially observable MDP** (POMDP)

TABLE 2.5: Machine Learning implementations for *proactive functionality* in honeypot technologies

Author	**Title**	Game Theory	RL	MDP	CBR
Wagener (2009) [96]	Self adaptive high interaction honeypots driven by game theory	x			
Wagener (2011) [79]	Heliza: talking dirty to the attackers		x		
Pauna (2013) [99]	Self-adaptive SSH Honeypot Model Capable of Reasoning				x
Hayatle (2013) [102]	Markov Decision Process Model for High Interaction Honeypots			x	
Pauna (2014) [98]	RASSH - Reinforced Adaptive SSH Honeypot		x		

Table 2.5 lists the authors, title, chronology and ML methods used for proactive functionality.

2.4.2 Retrospective Analysis

High interaction Honeypots provide backend databases to collect all activity such as IP addresses, timestamp, attempts, interactions, commands, downloads and executions [77]. Downloaded files can be sandboxed and analysed [103]. Sandboxing involves the reverse engineering of malware binaries, by allowing their execution in a controlled, isolated environment. This analysis is of particular interest to the cyber security industry. From this sandboxing activity, they can generate updates and fixes to add to dynamic rollouts. Longitudinal analysis of honeypot datasets has already been discussed in subsection 2.2.1. Clusters and graphs of botmaster to C&C communication, botnet methods, DDoS, time zones, spatial, IP domains and temporal are some of the characteristics already established with standard data mining methods [44, 45, 46, 47, 48, 49, 50]. Machine learning algorithms have also been explored to analyse honeypot datasets. Various classifications and training models have been proposed.

Pouget [104] used **association rules** and **clustering** techniques to analyse a captured dataset. It provided non trivial information allowing for the identification the various attacks targeting the environment. The Leurre project was the data source [48]. It consisted of a networks group of computers in different locations all running *Honeyd*. A centralised relational database to collect all attack traffic. This project and dataset was also used when **linear regression** was used to analyse attack traffic [105]. It found a global pattern to attack traffic with certain regions responsible for specific attack types. **Graph based clustering** was also used on the dataset to find groups of network traces sharing similar patterns [78]. This research proposed a flexible clustering tool that can be applied to the dataset to identify specific attack characteristics.

Seifert had previously proposed a taxonomy governing honeypot development and deployment [11]. In subsequent research, he uses **static heuristics** to identify malicious web pages [106]. The method classified malicious web pages by inspecting the underlying static attributes of the initial HTTP response and HTML code.

Pantev et al focused on the classification of attacks on web 2.0 services [107]. The web service honeypot was deployed for 9 months. The following machine learning algorithms were then applied: **Support Vector Machines** (SVM), **J48 Decision Tree** and **Partial Decision Tree** (PART). The authors compared the ability of each algorithm to distinguish malicious vulnerability scans. The research questioned whether supervised machine learning methods be used to automatically classify malicious activity and can malicious cyber activities be distinguished using a small number of features. It represented the results as confusion matrices and concluded that the decision tree methods, J48 and PART, performed better than SVM.

Zakaria suggests a review of AI techniques for developing intelligent honeypots [108]. It presents some elements of AI in the form of intelligent deployment or fingerprinting network flows. Only then does it propose **expert systems** and **CBR** to emulate human reasoning when interacting with attack traffic. It does not develop to environment further, but rather suggests tools that could be used for future work.

Ghourabi et al comments on the difficulties of analysing large datasets from honeypots [109]. It proposes the creation of a dedicated web service honeypot which is lacking in the research space. The web service honeypot was deployed for 3 months but did not capture enough malicious traffic for effective analysis. Therefore simulated attacks on the honeypot were carried out to enrich the dataset. The paper then uses 4 ML techniques to analyse the data: **SVM**, **Support Vector Regression** (SVR), **Spectral Clustering** and **K-Means Clustering**. These algorithms are implemented using *WEKA*, a Java based suite of software for machine learning [110]. Even though it augmented its dataset with simulations, it concludes that it has a 100% detection rate for DDoS attacks and identified SQL injection as the most common attack type on the web service honeypot.

Owezarski states that there is no research using **unsupervised learning** for honeypot dataset analysis [111]. The author uses *Unsupervised Anomaly Detection* and examines packets with nine traffic flow characteristics:

- SourceIP
- DestnIP
- 3 source prefixes (/8, /16, /24)
- 3 destination prefixes (/8, /16, /24)
- Traffic per timeslot

The resultant honeypot clusters the traffic flows and also creates clusters anomalies from an existing honeypot *NetFlow Project* [112]. It works in a completely unsupervised manner, automatically builds simple and small signatures which fully characterize attacks. It proposes taking advantage of honeypot data and automatically configuring filtering rules on intermediate Internet hardware.

Nanda et al evaluates the performance of four widely used ML algorithms: **C4.5 decision tree**, **bayesian network** (BayesNet), **decision table** and **Naïve-Bayes** [113]. However their goal is to show the viability of these ML approaches in SDN rather than assessing the four algorithms. Using WEKA, they don't build a honeypot but rather uses a dataset from the *Marist Longtail Project* [114]. It trains the algorithms and finds that BayesNet performs best with over 90% detection accuracy. The authors suggest that this ML approach can have a significant effect on the SDN security, if it even determines a small probability of an attack occurring.

Although Vanhoenshoven doesn't propose any honeypot design, the author does analyse a honeypot dataset using the following ML algorithms: **Naïve Bayes**, **SVM**,

Multilayered Perceptrons (MLP), **Decision Trees**, **Random Forest** and **K-Nearest Neighbours** [115]. The ML techniques were applied to a publicly available dataset of malicious URLS. All methods achieve fairly high prediction accuracy. Random Forest appears to be the most appropriate classification algorithm followed by MLP.

Sadasivam used a similar method to assess **SVM**, **decision trees**, **Naïve Bayes** and **K-Nearest Neighbours** [116]. Using an existing dataset from previous research [117], the author analyses the performance of these machine learning algorithms, and concludes that the decision tree algorithms (J48 and PART) performs best.

Table 2.6 lists the authors, title, chronology and ML methods used for retrospective analysis.

TABLE 2.6: Machine Learning implementations for *retrospective analysis* on honeypot datasets

Author	**Title**	Assoc. Rules	Graph Cluster.	Linear Regr.	Static Heur.	SVM	SVR	Spectral Cluster	K-Means	Unsupervised	Decision Tree	Naïve Bayes	Multilayer Perc.	Random Forest	K-Nearest Neigh	Expert System
Pouget (2004) [104]	Honeypot-based Forensics	x	x													
Dacier (2006) [105]	Collection and analysis of attack data based on honeypots deployed on the Internet			x												
Thonnard (2008) [78]	A framework for attack patterns' discovery in honeynet data		x													
Seifert (2009) [106]	Identification of malicious web pages with static heuristics				x											
Pantev (2012)	Using Multiclass Machine Learning Methods to Classify Malicious Behaviors					x					x					
Zakaria (2012) [108]	Review on Artificial Intelligence Techniques for Developing Intelligent Honeypot								x							x
Ghourabi (2013) [109]	Characterization of attacks collected from the Deployment of Web service honeypot					x	x	x	x							
Owezarski(2014) [111]	Unsupervised Classification and Characterization of Honeypot Attacks									x						
Nanda (2016) [113]	Predicting Network Attack Patterns in SDN using Machine Learning Approach										x	x				
Vanhoenshoven (2016) [115]	Detecting Malicious URLs using Machine Learning Techniques					x							x	x	x	
Sadasivam (2018) [116]	Detection of Severe SSH Attacks Using Honeypot Servers and Machine Learning Techniques					x					x	x			x	

2.5 Chapter Summary

Technology has evolved in a very short time and has had a positive impact on society and industry. For example, smart cities as a concept aims to gather and collate information from industry society and the environment. With this information, real-time decisions involving innovative technologies such as CPS, M2M, WSNs, AI etc, can be made to enhance urban living. The Internet is still very much in an embryonic state. Much has been learned about setting up a global connected entity but security issues still exist. Exploitation of these security vulnerabilities is ironically human nature. These problems are migrating to new and emerging technologies such as IoT and IIoT. Malware is evolving apace to exploit these areas. Security groups are continuously in a reactive mode, capturing new malware and their variants and releasing fixes, patches and updates. Contributing to this reactive process, honeypots are predominately used for longitudinal studies of malware behaviour. They evolve in step and simulate new technologies in order to target new malware. To visualize this evolution, fig. 2.10 presents milestones for technology, malware and honeypot development.

To engender trust in integrated technologies that abstract human cognition, society needs to be confident that both it and its data are not vulnerable to cyber attack. It is important to learn from previous exploitations and apply this learning proactively. Better Internet connectivity coupled with more Internet facing devices will facilitate the creation of innovative malware. Whereas some research has proposed proactive methods for high interaction honeypots (table 2.5), it provides for responses to

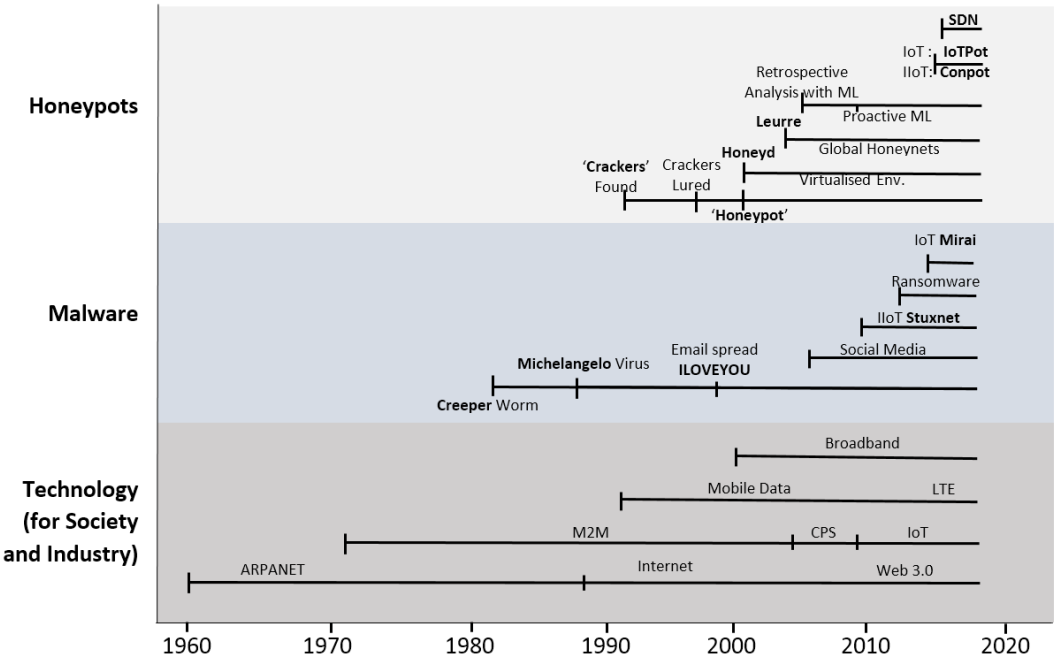


FIGURE 2.10: Evolution of Technology, Malware and Honeypots

human activity on a honeypot. Further research has demonstrated that malware methods are predominately automated and repetitive [10]. They continuously attempt to compromise a host. Therefore captured datasets have the characteristics of attack sequences that are automated and repetitive.

This chapter gives an overview of the evolutionary nature of technology as used by society and industry. The growth and diversity of technological concepts is intrinsically linked with population. For the first time in 2007, urban populations exceeded 50% [118]. Similarly, society is living longer and contributing more and more to climate change. Efforts to deal with issues associated with societal requirements involve technological advances. Mobility, energy, food production and distribution are some of the concepts encapsulated in *smart cities*. The demand to find smart solutions to social issues often create technical vulnerabilities that are invariably exploited. This demand is only going to increase and there isn't a panacea for the problems that will emerge. This chapter also identifies some of the malware advances that have exploits emerging technology vulnerabilities. From the original ARPANET worms to more recent IoT compromises, malware has and will continue to find new attack surfaces. The subsequent exploitation can range from irritating to being financial exploitative.

Finally, this chapter also demonstrates that honeypot development has kept pace by reacting to new malware methods and their variants. Honeypots also use new technologies to capture and analyse malware. Virtualisation and machine learning have significantly contributed to honeypot evolution. Machine learning techniques have been used extensively and effectively to retrospectively analyse captured honeypot datasets. This analysis gives great insight into the methods used by malware developers vis-à-vis infection, propagation and control. Less extensive has been the use of machine learning to proactively interact with an attack in order to prolong interaction and capture bigger datasets or datasets with more meaningful information. Previous research specifically in this area has provisioned human interaction [79, 98]. However, it is contended here that malware is predominately automated and research into adaptive honeypot development should be focused on this. The very characteristics of malware automation and repetition can be used against the malware itself. Machine learning techniques can focus on these characteristics and learn from an interminable supply. Incrementally, honeypot development and deployment can reduce the time it takes to realise a dataset that contains meaningful information. This requires a framework for their adaptive development and agile deployment.

The triumvirate of technology, malware and honeypots governs research involving malware capture and analysis going forward; to wit, beyond 2020 in fig. 2.10. To tilt this balance of power in favour of honeypot development, **chapter 3** proposes a new methodology for honeypot development and **chapters 4, 5 and 6** address the research questions stated in subsection 1.2.1. The chapters detail the methodologies

used to address these questions and presents results and findings supporting the accompanying hypotheses.

Chapter 3

Methodology

3.1 Introduction

Malware employs highly automated methods to propagate and compromise hosts. As discussed in subsection 2.2.1, human cognition in the process can involve initial coding and deployment of the malware, communicating with a C&C if required and coordinating attacks. Network and end devices on the Internet are repeatedly probed by automated malware. Multiple infected sources exacerbates this repetitiveness. Honeypots actively seek to interact with cyber attacks by simulating vulnerable Internet services and devices. It is their raison d'être. This results in honeypot datasets rich in repetitive attacks of automated sequences [10]. The analytical value associated with datasets of this type is longitudinal, providing information on attack patterns such as spatial and temporal [49]. Being deployed on the Internet, honeypot functionality is itself vulnerable. Malware developers became aware of the existence honeypots, capturing and analyzing attack activity. To counter this, dedicated honeypot detection tools were developed and evasion techniques were designed into malware [119]. Tools such as *Honeypot Hunter* performed tests to identify a honeypot [8]. False services were created and connected to by the anti-honeypot tool. Honeypots are predominately designed to prolong attacker interaction and therefore pretend to facilitate the creation and execution of these false services. This immediately tags the system as a honeypot. With the use of virtualization for honeypot deployments [120], anti-detection techniques issued simple kernel commands to identify the presence of virtual infrastructure instead of bare metal [94]. New versions of honeypots are redesigned and redeployed continuously to counter new malware methods. More recently, the Mirai botnet spawned multiple variants targeting IoT devices through various attack vectors [3]. Analysis of the Mirai variants found that Kippo failed initial anti-honeypot tests before honeypot functionality was manually amended [121]. Examining the structure of the Mirai variant [122], when a 'mount' command is issued, the honeypot returns it's standard response at which point the attack ends. It indicates that the variant is implementing anti-detection techniques. Virtual machine (VM) aware botnets, such as *Conficker* and *Spybot* scan for the presence of a virtualized environment and can refuse to continue or modify its methods [123]. As a consequence of this, honeypot effectiveness is compromised.

The quality of the resultant dataset is reduced. Inadvertently terminating interaction truncates the attack sequences captured by the honeypot. In order to mitigate against inadvertent termination, detection and repetitive truncated datasets, it is incumbent on honeypot developers and operators to implement alternative measures. Measures that can provide the honeypot with the ability to learn from the attacks. These intelligent honeypots will ultimately prolong interaction and capture more meaningful data; their raison d'être.

This chapter presents the functional elements of an intelligent honeypot that can adapt to malware. It involves identifying automation and repetition in a honeypot dataset, development of HARM to exploit these characteristics and the assessment of HARMs performance vis-à-vis learning. The first task is to identify malware attack characteristics as being predominately automated and repetitive (section 3.2). Once this is established, HARM development is detailed (section 3.3). Reinforcement learning algorithms are examined to exploit the characteristics of automation and repetition (subsection 3.3.1.1). The unique state action space formalism designed to exploit the characteristics of automated and repetitive malware is detailed (subsections 3.3.1.3 and 3.3.1.4). Section 3.3.2 examines the methodology used to stream a dataset extract (captured from HARMs deployment: see appendix B) into HARM in a controlled environment.

This adaptive honeypot for automated and repetitive malware (HARM) is deployed in live and controlled environments to explore the hypotheses in chapters 4, 5 and 6.

3.2 Automation and Repetition

The dataset associated with fig. 2.4 is from an IoT honeypot deployed over 3 months [10]. The IoT honeypot simulated a ZigBee gateway on a SSH attack vector. Table 2.2 identifies ZigBee as a communications protocol used by FFDs and RFDs. Wireless technologies cannot negate the interception of communications. They can however, ensure the secrecy of the message by providing encryption. The inherent availability of WPANs and *wireless local area networks* (WLANs) transmissions requires robust encryption mechanisms at all layers. Vulnerabilities exist when ZigBee applications do not implement security [124, 125, 126]. Intercepted transmissions can be subjected to brute force attacks and subsequent analysis [127]. Physical, Key, Denial-of-Service, Injection and Relay are examples of well-known attack types that can be perpetrated on ZigBee networks. A network key is a global key used by all devices in a WPAN. Some sharing methods require these keys to be transported across the WPAN; sometimes keys are hard coded in the device. By either physically accessing the device or eavesdropping, keys and key transport transmissions can be captured, for analysis and subsequent key attacks. Replay and injection attacks can have catastrophic consequences. Specific hardware commands can be looped or inserted into a WSN communications channel. A command looped to turn a water valve 1 degree or

delivering incorrect blood pressure measurements from medical *body area networks* (BANs). KillerBee [128] is an attack framework consisting of hardware and software tools that can intercept and analyse 802.15.4 and ZigBee transmissions. Since the ZigBee standard is based on IEEE 802.15.4, attacks on IEEE 802.15.4 MAC layer are harmful to ZigBee-based devices. KillerBee analysis shows that frame security of IEEE 802.5.4 MAC is a critical issue. Frame security is a set of optional services that may be provided by the IEEE 802.15.4 MAC to ZigBee. If an application does not set any security parameters, then no security feature is enabled by default. More recently, incidents have been reported where IoT worms can spread across ZigBee networks [129], drones go 'war driving' to hack a building from outside [130] and **Mirai** is used in a massive DDoS attack [131]. To assess IoT cyberattacks, specific IoT honeypots have been deployed to ascertain vulnerable architecture [81, 7].

As well as defining a *honeypot* as a "security resources whose value lies in being probed, attacked or compromised" [55], Spitzner also discussed the use of a **honeytoken** [76]. Honeytokens lure attackers into a honeypot by appearing to be information pieces that are supposedly left inadvertently available. Interesting file names, folder titles and enticing banners are flagged by an attacker as being of interest. This supposedly means that further interaction is captured on the honeypot as an attacker explores the honeytokens. To simulate this extra layers of attraction are built into the honeypot. These layers will tempt the attacker with ZigBee specific information; the honeytokens pointing to another server. The honeypot was made available on the Internet as an SSH *zigbee-gateway* with embedded fictitious ZigBee WSN medical traffic as honeytokens. Data accessed within this honeypot will point attacker to another IP address, creating a Honeynet to keep an attacker engaged for longer. To create a true simulation of a "zigbee-gateway", it is necessary to modify Python scripts provided by the Kippo distro, and to create new python scripts to display the honeytokens. This included a tcpdump script to manipulate pcap file manipulation. Fig. 3.1 presents the interactive honeypot. The circled elements highlight tcpdump code and further URL honeytokens. Further IP address correspond to Amazon Web Services (AWS) VM instance. This VM will display an error but will also log IP address and time stamps associated with the source.

After the 3 month deployment of the IoT honeypot, the attack types were identified by sandboxing the downloaded files, observing shell interactions and consulting threat advisories. Overall, 99.6% of the traffic encountered on the honeypot was automated [10]. Analysis of 6 million lines of attack code from the dataset produced the following:

- 423228 login attempts
- 413362 unsuccessful logins
- 9866 successful logins
- 5297 distinct IP sources

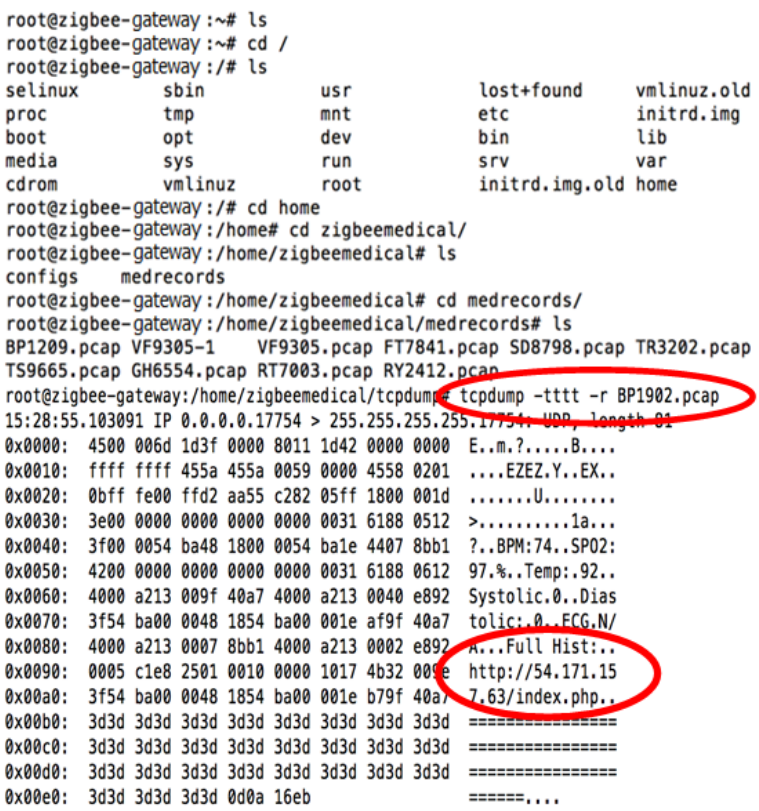


FIGURE 3.1: ZigBee IoT Honeypot with embedded Honeytokens

- 31328 commands
- 5368 downloads

Automated attack types were examined and collated. They were identified as *Dictionary*, *Recon*, *Failed*, *Launch* commands, *XOR DDoS* and *BillGates*. Dictionary attacks continuously probed with username/password combinations. They were eliminated from the dataset as they were pre-compromise. Failed attacks made an initial connection with correct username and password but failed on authentication. The XOR DDoS and BillGates Botnet provided better material for examination. The downloaded files were sandboxed and the scripts were analysed. They demonstrated automated methods to gather information on variables such as compilers, CPU and operating systems. Both treated the honeypot as an SSH device primarily and concentrated on compromising it in that regard. Recon scripts ran ten minutes after XOR and checked for SFTP capabilities. Launch was a command requesting an IP flood to a victim IP. Fig. 3.2 presents an image of the attack types encountered on the IoT honeypot. The cumulative repetition of these attack types can be collated. The repetitious nature of some attacks can be seen in fig. 3.3, which indicates the quantities of each attack within the deployment period. Further analysis revealed in fig. 3.4, shows the constant repetition of two of the more complex automated scripts.

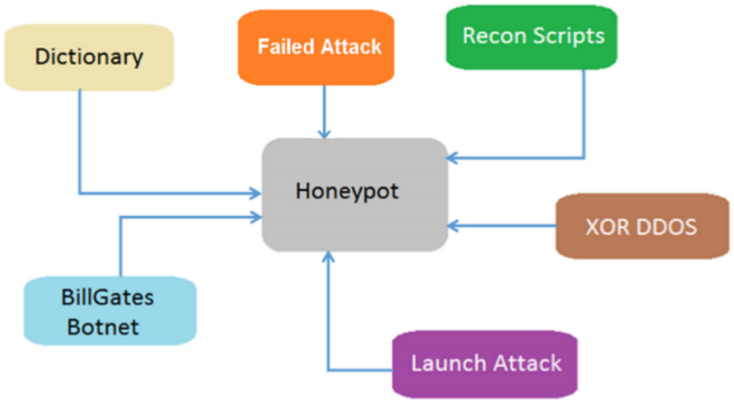


FIGURE 3.2: Attack Types encountered on IoT Honeypot

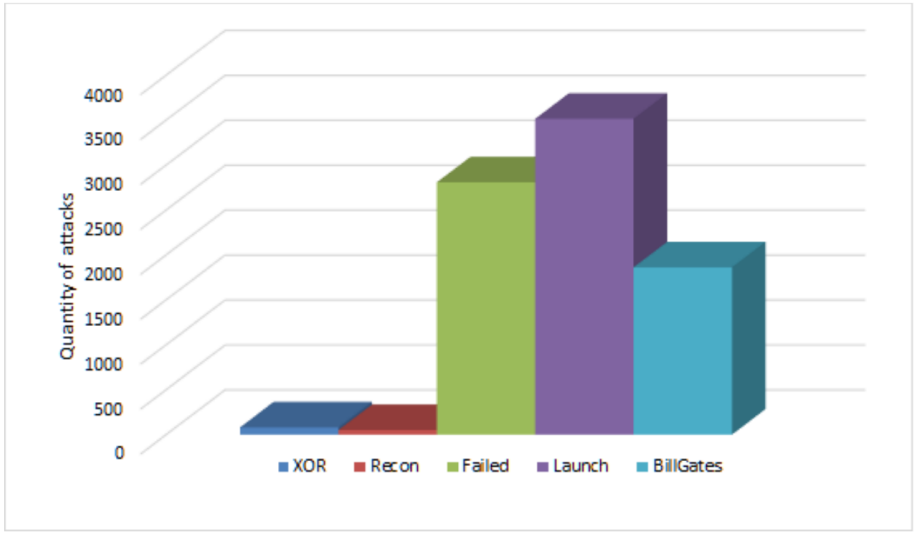


FIGURE 3.3: Number of Attempts by Attack Type

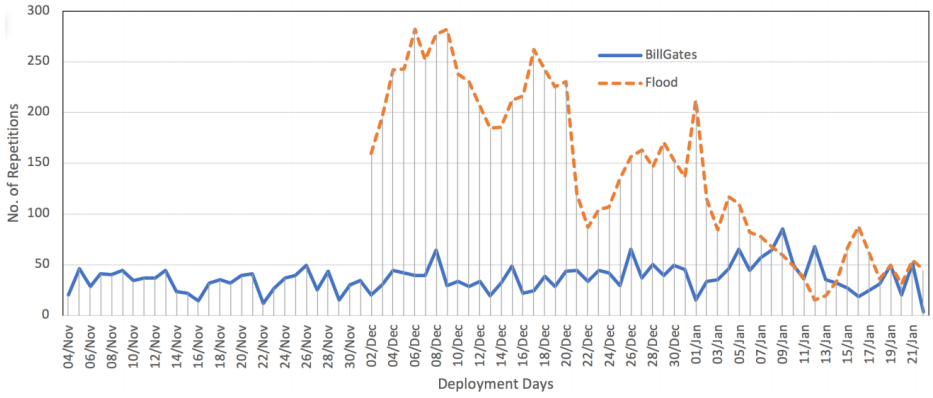


FIGURE 3.4: Repetition of *BillGates* and *Launch Flood* Attack Types

3.3 HARM Development

Section 3.2 identified malware methods using automation and repetition to compromise vulnerable devices and propagate. Honeypot development requires alternative methods to stay relevant as cyber forensic tools. One method is the incorporation of reinforcement learning into honeypot operations to exploit automation and repetition. HARM requires a cyclic process of development -> deployment -> optimisation -> redeployment as proposed in fig. 1.2. The methodology behind this process can be considered to have two stages. Fig. 3.5 graphically presents these two stages, namely:

- **Implementation:** Integrating reinforcement learning into an IoT honeypot
- **Assessment:** Deploying and Assessing HARMs performance

Section 3.3.1 describes the incorporation of reinforcement learning into the honeypot process. Specifically learning algorithms SARSA and Q-learning and the reward function are detailed. The implementation of HARM produces data stores which are explored in chapter 4. Manipulating these datastores (appendix B) produces an attack stream that (when modified) can be used as input into HARM in a controlled environment. Section 3.3.2 explains this process which produces results also presented in chapter 6.

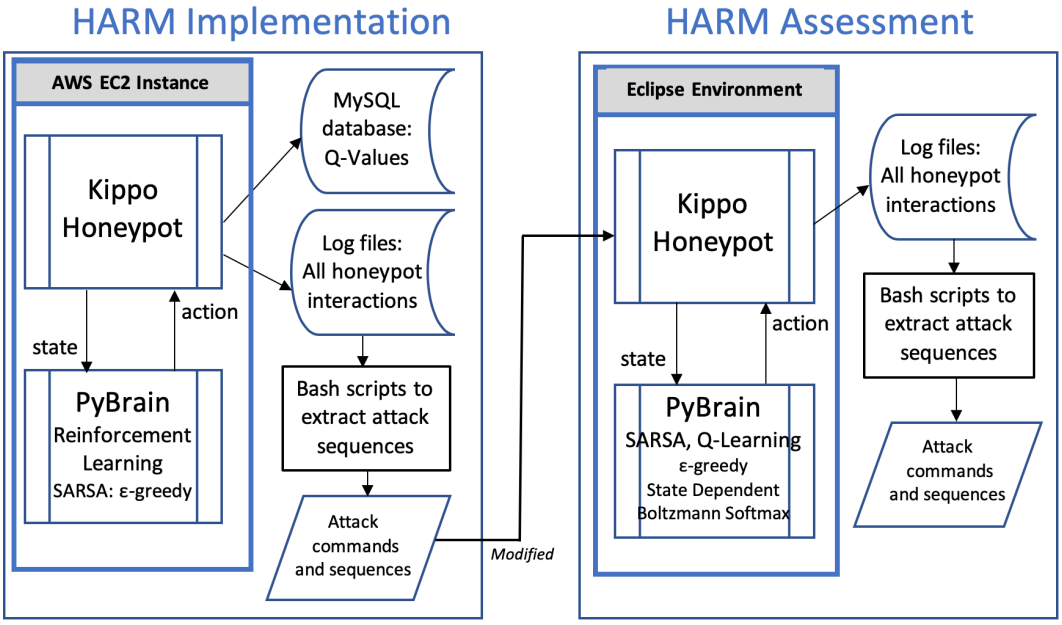


FIGURE 3.5: HARM Implementation and Assessment

3.3.1 HARM Implementation

The incorporation of machine learning into honeypot technology is not new. Tables 2.5 and 2.6 present machine learning algorithms that have been associated with honeypots for proactive functionality and retrospective analysis. The adaptive element of real time, proactive functionality ensures the honeypot learns from attack interactions. The development of HARM reproduces previous research [79, 98], modified for automation and repetition, creating a unique state space action formalism.

3.3.1.1 Reinforcement Learning

Reinforcement learning is a machine learning technique in which a learning agent learns from its environment, through trial and error interactions. Rather than being instructed as to which action it should take given a specific set of inputs, it instead learns based on previous experiences as to which action it should take in the current circumstance.

Markov Decision Processes Reinforcement learning problems can generally be modelled using Markov Decision Processes (MDPs). In fact reinforcement learning methods facilitate solutions to MDPs in the absence of a complete environmental model. This is particularly useful when dealing with real world problems such as honeypots, as the model can often be unknown or difficult to approximate. MDPs are a particular mathematical framework suited to modelling decision making under uncertainty. A MDP can typically be represented as a four tuple consisting of states, actions, transition probabilities and rewards.

- S, represents the environmental state space;
- A, represents the total action space;
- p$(.|s,a)$, defines a probability distribution governing state transitions $s_{t+1} \sim p(.|s_t, a_t)$;
- q$(.|s,a)$, defines a probability distribution governing the rewards received $R(s_t, a_t) \sim q(.|s_t, a_t)$;

S the set of all possible states represents the agent's observable world. At the end of each time period t the agent occupies state $s_t \in S$. The agent must then choose an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all possible actions within state s_t. The execution of the chosen action, results in a state transition to s_{t+1} and an immediate numerical reward $R(s_t, a_t)$. Equation 3.1 represents the reward function, defining the environmental distribution of rewards. The learning agent's objective is to optimize its expected long-term discounted reward.

$$R_a s, s' = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{3.1}$$

The state transition probability $p(s_{t+1}|s_t, a_t)$ governs the likelihood that the agent will transition to state s_{t+1} as a result of choosing a_t in s_t (eq. 3.2).

$$P_a s, s' = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{3.2}$$

The numerical reward received upon arrival at the next state is governed by a probability distribution $q(s_{t+1}|s_t, a_t)$ and is indicative as to the benefit of choosing a_t whilst in s_t. In the specific case where a complete environmental model is known, i.e. (S, A, p, q) are fully observable, the problem reduces to a planning problem and can be solved using traditional dynamic programming techniques such as value iteration. However if there is no complete model available, then reinforcement learning methods have proven efficacy in solving MDPs.

3.3.1.2 Reinforcement Learning Methods

Temporal Difference is a learning methods used in reinforcement learning [132]. As stated, a honeypot modelled as a MDP may have an incomplete environment. Monte Carlo method uses sequence of states, actions and rewards from interactions within the environment. Therefore it learns from experience. Each experience is an episode, at the end of which a return R is given and the value estimates and policies are changed. Temporal Difference (TD) updates are estimated using intermediate rewards as inputs. In equation 3.3, the TD update is the observed reward R_{t+1} and the estimate $V(S_{t+1})$.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1} - V(S_t)] \tag{3.3}$$

There is a requirement for TD to exploit the learned knowledge or explore the environment further. Two approaches are considered for the trade off between exploration and exploitation:

- on-policy
- off-policy

SARSA On-policy During the reinforcement learning process the agent can select an action which exploits its current knowledge or it can decide to use further exploration. Reinforcement learning provides parameters to help the learning environment decide on the reward and exploration values. Fig. 3.6 models a basic reinforcement learning interaction process.

A temporal difference method for on-policy learning uses the transition from one state/action pair to the next state/action pair, to derive the reward. **State**, **Action**, **Reward**, **State**, **Action** also known as **SARSA**, is a common implementation of on-policy reinforcement learning (*eq.* 3.4). The reward policy Q is estimated for a given state s_t and a given action a_t. Actions are chosen based on an action selection strategy, where the ultimate goal is to choose actions which yield the greatest long term

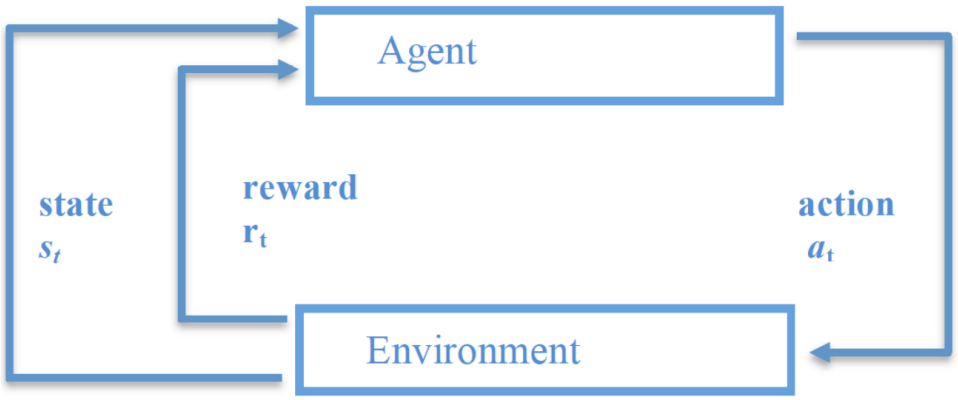


FIGURE 3.6: Reinforcement learning model

reward. The estimated Q value is expanded with a received reward r_{t+1} plus an estimated future reward $Q(s_{t+1}, a_{t+1})$, that is discounted (γ). A learning rate parameter is also applied (α), determining the extent to which new information is merged with older information.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{3.4}$$

Applying this process to honeypot interactions, each attack is considered an episode. Each episode contains a collection of attack commands directed at the honeypot, post compromise. The learning policy is evaluated at the end of each attack command.

Q-Learning Off-policy Watkins proposed Q-Learning as an off-policy TD learning algorithm [133]. The Q values converge to the optimal policy independent of the agents explorer. The goal of Q-learning is to learn the optimal policy, whereby the agent takes the best action in certain circumstances. This will require each state to be visited an infinite number of times in order to converge on the optimum Q-values. The estimated Q-value is expanded with an estimate of optimal future value $\max_a Q(s_{t+1}, a)$ (*eq.* 3.5).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.5}$$

Both Q-learning and SARSA are model free approaches as the environment is partially known.

3.3.1.3 HARMs State Action Space Formalism

Previous contributions demonstrate that there is a relationship between honeypot and malware development and evolution. Botnet traffic is highly automated as it attacks, compromises and reports without any intervention. From a malware developer's perspective, there is very little human interaction, post launch. Standard

high interaction honeypot responses can terminate an attack sequence. Known evasion techniques can uncover honeypot functionality. Therefore adaptive honeypots need to learn from these automated interactions. It is relevant to consider how the reinforcement learning model in fig. 3.6, applies to adaptive honeypot development.

- Throughout its deployment, the honeypot is considered to be an **environment** with integrated reinforcement learning. SSH is an access point, simulating Linux server as a vulnerable environment. SSH is responsible for 62% of all compromise attempts [134].
- Within this environment, the server has **states** that are examined and changed with bash command and complex scripts. Examples are *iptables stop*, *wget <URL>*, *sudo*, *chmod 777* etc.
- The reinforcement learning agent can perform **actions** on these state commands such as to *allow*, *block* or *substitute* the execution of the commands and scripts.
- The environment issues a **reward** to the agent for performing that action. The agent learns from this process as the honeypot is attacked and over time learns the optimum policy π^*, mapping the optimal action to be taken each time, for each state s. The learning process will eventually converge as the honeypot is rewarded for each attack episode.

The honeypot is required to learn and will be rewarded for prolonging interaction. Therefore the reward function is to increase the number of commands from the attack sequence. Attacker commands can be categorised into the following:

- L – Known Linux bash commands
  wget, cd, mount, chmod, etc.
- C – Customised attack commands
  Commands executing downloaded files
- CC - Compound commands
  Multiple commands with bash separators/operators
- NF - known commands not facilitated by honeypot
  The honeypot is configured for 75 of the 'most used' bash commands.
- O - Other commands
  Unhandled keystrokes such as ENTER and unknown commands

3.3.1.4 Reward Function

HARM incorporates reinforcement learning to create an adaptive IoT honeypot. Previous contributions related to this concept used action sets in the reinforcement learning model provisioning human interaction: Wagener (2011) [79] and Pauna

(2013) [98]. Wagener presented a honeypot called *Heliza*. The reward function had two objectives. The first is to collect attacker related information, which includes transitions to attacker tools, system tools and insulting commands. The second is to extend the duration of an attacker's interaction expressed as the delay between attacker commands suggesting human cognitive functionality. Wagener presents results for the two reward functions. For the first reward he selects two states of *sudo* and *wget* as state examples and shows the learning values calculated for each action. The second reward compares the cumulative number of transitions captured on the honeypot, with a standard high interaction honeypot. The honeypot was developed using a User Mode Linux (UML) framework. Pauna also presents an adaptive honeypot using the same reinforced learning algorithms and parameters as Heliza. He proposes a honeypot named *RASSH* that provides improvements to scalability, localisation and learning capabilities. It does this by using newer libraries and Kippo, a high interaction honeypot [77]. RASSH compares its functionality with Heliza demonstrating similar behaviour. Both Heliza and RASSH use PyBrain [97] for their implementation of reinforcement learning and both use actions that are targeting human interaction. Actions such as insulting the attacker and delaying the response were proposed and implemented into prototype honeypots. Two reward functions corresponding to the two goals of the honeypots were implemented:

1. Collect attacker related commands and transitional information (r_t)
2. Extend the duration of an attacker's interaction (r_d)

It has been highlighted that attack traffic is predominately automated and repetitive. Actions such as insult and delay artificially contributes to the duration of an attack episode and provides no further insight into the machinations of automated malware. Therefore, there is no requirement to implement the second reward (r_d). Our implementation pursues the goal of increasing the number of attacker commands entered on our honeypot (r_t). Adaptive honeypots deployed on a SSH attack vector should implement reinforcement learning with a reduced action set aimed at automated and repetitive malware. Due to the fact that *insult* is removed as a realistic response to automated malware attacks, the reward function for the reinforcement learning implementation on the honeypot needs to be modified. RASSH compared itself to Heliza by charting the learning evolution for collecting attacker information (transitions), on state *wget*. The formula used by both is as follows:

$$r_t(s_i, a) = \begin{cases} 1, \text{if i} \in \text{C} \\ \min\limits_{x \in Y} (l_d(\text{i,x})), \text{if i} \in \text{I} \\ 0, \text{otherwise} \end{cases} \tag{3.6}$$

$$\text{where Y} = \text{C} \cup \text{L}$$

The formula for transition reward r_t, based on state/action (s_i,a_j), is as follows:

- If the input string i is a customised attacker command C, then reward $r_t(s_i,a_j)$ = 1
- If the input string i is a Linux bash command L, then reward $r_t(s_i,a_j) = 0$
- If the input string i is determined to be an insult I, then the minimum normalised Levenstein [135] distance l_d relative to Y or ENTER is calculated.

The insult set I is considered to anything other than Y and ENTER. The reward for this is obtained by calculating the Levenstein distance between the attack command and Y, ENTER. As stated, insult is not a relevant action item for automated attacks.

The reduced action set for HARM is *allow*, *block* and *substitute*. The proposed transition reward function means the learning agent is rewarded if a bot command is an input string i, comprised of bash commands (L), customised commands (C) or compound commands (CC). Therefore Y = C ∪ L ∪ CC. For example *iptables stop* is an input string i that transitions the state s of *iptables* on a Linux system to the next command in the attack sequence. Other commands or commands not facilitated by the honeypot are not given any reward. The proposed action set is reduced to a = *allow, block, substitute*. Allow and Block are realistic responses to malware behaviour. Botnets can use complex if-else structures to determine next steps during compromise [10]. Using Substitute to return an alternative response to an attack command potentially increases the number of attack transitions to newer commands. This action set coupled with state set Y, creates a discrete state/action space. The reward function is as follows:

$$r_t(s_i, a) = \begin{cases} 1, \text{if i} \in \text{Y} \\ 0, \text{otherwise} \end{cases} \tag{3.7}$$

$$\text{where Y} = \text{C} \cup \text{L} \cup \text{CC}$$

The formula for transition reward r_t, based on state/action (s,a), is as follows:

- If the input string i is a customised attacker command C or a known bash command L or a compound command CC, then reward $r_t(s_i, a) = 1$
- otherwise the reward $r_t(s_i, a) = 0$

This manifests in Python code as a call to *getCommandReward* to retrive the reward for further processing by HARM and PyBrain. Fig 3.7 displays the Python code section to achieve this. The adaptive learning process for HARM incorporating the reinforcement learning model in fig. 3.6, is shown in fig. 3.8.

```
def getCommandReward(command):
    l = ["wget", "scp", "ftp", "mount", "echo", "rm", "tar", "whoami", "cat", <other states>]
    for c in l:
        if c in command:
            return 1
        if command.startswith("./"):
            return 1
        l2 = ["python", "php", "sh", "bash"]
        for c in l2:
            if command.startswith(c):
                return 1
    return 0
```

FIGURE 3.7: Reward Function within Python

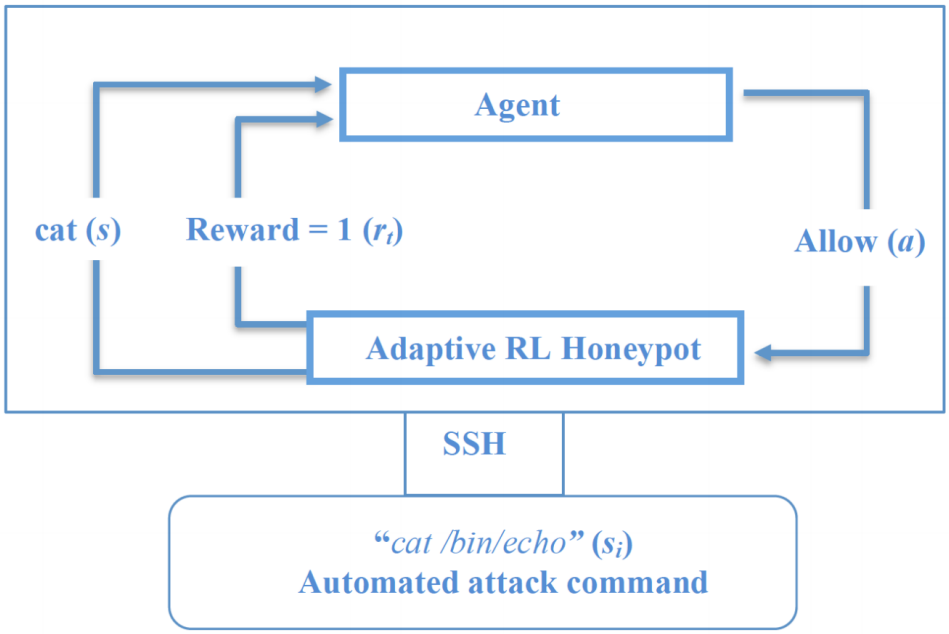


FIGURE 3.8: Adaptive IoT HARM

The adaptive honeypot has the following elements:

- Modified honeypot. Kippo is a widely used honeypot distribution that logs all SSH interactions in a MySQL database. This has been modified to generate the parameters to pass to the learning agent. Depending on the action selected by the learning agent, the honeypot will allow, block or substitute attack commands
- SARSA agent. This module receives the required parameters from the adaptive honeypot and calculates $Q(s_t, a_t)$ as per equation 3.4. It determines the responses chosen by the adaptive honeypot and learns over time which actions yield the greatest amount of reward. The SARSA update rule 3.4 calculates Q values by incorporating the reward achieved from choosing action a, the current value of the state action pair and the discounted value of the next state action pair. Applying a discount factor in this way allows the agent to discount the value of future rewards, as they may not turn out to have that value once you get there. The current attack command is parsed as the current state and passed to the PyBrain SARSA module. PyBrain selects the action that will return the maximal value for a given state. The Q values are returned to the honeypot to be stored in a lookup table.

The honeypot continues by interacting with the attack command depending on the action selected by the SARSA learning agent. SARSA is implemented with the following parameters:

- ϵ-greedy policy
  The honeypot environment is unknown to the learning agent. It learns without prior knowledge and eventually converges. To this end, our explorer is set to ϵ-greedy.
- Discount factor γ=1
  γ is applied when a future reward is estimated. The honeypot has no discount factor applied, as attack episodes are readily defined commands between SSH open and close events. This allows for retrospective reward calculations.
- Step size α=0.5
  a step size parameter is applied for the creation of the state/action space.

3.3.2 HARM Assessment

Fig. 3.5 diagrammatically proposes that the deliverable from *HARM Implementation* can be modified and used in *HARM Assessment*. To realise this the following two elements of this process need to be discussed:

- Reinforcement Learning with PyBrain
- Modified Attack Stream

The attack stream requires a data source representative of real world malware trends and methods. It has to have the ability to interact with the HARM with consistency across the learning environment. The Kippo honeypot and PyBrain are both written in Python allowing for the intuitive modification of learning algorithms, explorers and policies. Together these two elements facilitate an assessment of HARM.

3.3.2.1 Reinforcement Learning with PyBrain

HARM was initially deployed on the Internet using SARSA (*eq.* 3.4). Sutton and Barto [132] presents algorithms for both SARSA and Q-Learning. Algorithm 1 shows that SARSA chooses s_{t+1} and a_{t+1} prior to updating the Q function. Algorithm 2 shows Q-Learning which first updates the Q function and then chooses the next action based on the updated Q function (SARSA being on-policy, Q-Learning is off-policy).

Using Python, HARM can import and use SARSA or Q-Learning (this is coded in fig. 3.11). Within PyBrain, the implementations of algorithms 1 and 2 are facilitated with Python code as seen in fig. 3.9 and fig. 3.10 respectively. For this code to function, the adaptive honeypot stores variables for the current state, action and reward. These are passed to PyBrain to update the Q function and are then updated.

Algorithm 1 Reinforcement Learning SARSA

Initialise $Q(s, a)$ arbitrarily
  repeat (for each episode):
  Initialize s
  Choose a from s using policy derived from Q (e.g., ϵ-greedy)
  repeat
    Take action a, observe r, s_{t+1}
    Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g., ϵ-greedy)
    $Q(s_t, a_t) \leftarrow Q(s_t,a_t)+\alpha[r + \gamma\, Q(s_{t+1},a_{t+1}) - Q(s_t, a_t)]$
    $s \leftarrow s_{t+1}$, $a \leftarrow a_{t+1}$;
  until s is terminal

Algorithm 2 Reinforcement Learning Q-Learning

Initialise $Q(s, a)$ arbitrarily
  repeat (for each episode):
  Initialize s
  repeat
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe r, s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t,a_t)+\alpha[r + \gamma \max_{a+1} Q(s_{t+1},a_{t+1}) - Q(s_t, a_t)]$
    $s \leftarrow s_{t+1}$;
  until s is terminal

```
qvalue = self.module.getValue(self.laststate, self.lastaction)
qnext = self.module.getValue(state, action)
self.module.updateValue(self.laststate, self.lastaction, qvalue + self.alpha
                        * (self.lastreward + self.gamma * qnext - qvalue))

# move state to oldstate
self.laststate = state
self.lastaction = action
self.lastreward = reward
```

FIGURE 3.9: PyBrain SARSA

```
qvalue = self.module.getValue(self.laststate, self.lastaction)
maxnext = self.module.getValue(state, self.module.getMaxAction(state))
self.module.updateValue(self.laststate, self.lastaction, qvalue + self.alpha
                         * (self.lastreward + self.gamma * maxnext - qvalue))

# move state to oldstate
self.laststate = state
self.lastaction = action
self.lastreward = reward
```

FIGURE 3.10: PyBrain Q-Learning

PyBrain also provides for the following policies [97]:

- Epsilon greedy
  A discrete explorer that mostly executes the original policy but sometimes returns a random action.

- Boltzmann Softmax
  A discrete explorer that executes actions with a probability that depends on

their action values.

- State dependent [136]
  A continuous policy that disrupts the resulting action with added, distributed random noise.

These learning policies can be coded as *pybrain.rl.explorers*, as seen in fig. 3.11. This flexibility within the controlled environment allows for experimentation of HARM functionality by modifying variables (SARSA, Q-Learning, ϵ-greedy, ϵ-greedy values, Boltzmann Softmax, State dependent).

```
from pybrain.rl.agents import LearningAgent
from pybrain.rl.learners import Q,SARSA
from pybrain.rl.explorers import EpsilonGreedyExplorer

import threading

class RL:
    def __init__(self):
    learner = SARSA()
    learner._setExplorer(EpsilonGreedyExplorer(0.1))
```

FIGURE 3.11: Adaptive Honeypot Reinforcement Learning Selection

This flexibility within the controlled environment provides for further optimization of HARM. By modifying the parameters (SARSA, Q-Learning, ϵ-greedy, ϵ-greedy values, Boltzmann Softmax, State dependent) within the controlled environment and streaming a captured dataset, an optimum version of HARM can be determined.

3.3.2.2 Modified attack stream

Appendix C.3 demonstrates how command sequences used during attacks are extracted from the captured dataset and exported to a file, *stream.txt*. This text file is used to assess HARMs performance in a controlled environment. However, if streamed as it exists (imported into a terminal window) then HARM will always process each line irrespective of the outcome from the previous line. Therefore a forced intelligence needs to be implemented to reflect malware methods. This intelligence modifies how the lines in *stream.txt* are processed by the honeypot. Whilst it is impossible to cater for all malware types, a standardised method for processing an attack stream ensures consistency across the assessment process. The various combinations of algorithms, policies and explorers will each be subjected to an input stream that behaves in the same way. During HARMs live deployment the dominant malware Mirai variant downloaded a loader program, post bruteforce compromise [9].

> "This loader asynchronously infected vulnerable devices by logging in, determining the underlying system environment and finally downloading and executing architecture-specific malware. After a successful infection, Mirai attempted to conceal its presence by deleting the downloaded binary and obfuscating its process name in a pseudorandom alphanumeric string."

The downloaded malware source code used by the variant was a mixture of **C** and **Go** programming. The commands were also encrypted and obfuscated. The response (action) of HARM to the commands in this loader program, impact on whether the attack sequence proceeds or terminates. HARM is configured with an action set of *allow*, *block* and *substitute*. The rationale for this has already been discussed in subsection 3.3.1.4. During the live deployment, malware (and specifically the Mirai variant) interpreted these actions prior to deciding on the next line in the attack sequence. Therefore an *action daemon* needs to be included on the streaming process to reflect these malware methods. The rules for this daemon are as follows:

- **Allow** - Proceed to the next line in *stream.txt*.
- **Block** - Proceed to the next line in *stream.txt*:
  - If next line action is **Allow** then proceed to the next line in *stream.txt*.
  - If next line action is **Block** then terminate.
  - If next line action is **Substitute** then proceed to the next line in *stream.txt*.
- **Substitute** - If previous 2 actions were **Substitute** then terminate, else proceed to the next line in *stream.txt*.

It is important to note that these specific rules evolved after numerous experiments with HARM and *stream.txt*. Initially, all commands were just executed with no termination. Establishing the rules involved mimicking HARM's live performance without impacting PyBrain's functionality. PyBrain chose the best actions over time to prolong interaction with the downloaded malware. It did this using SARSA and ϵ-greedy = 0.05, allowing the learning agent to use further exploration. The rationale behind these rules are as a result of a quick analysis of the captured dataset from the live deployment. **grep Action kipp*** bash command displays the actions taken for the entire dataset (it uses the 'Action' keyword from the logs). Examining the output, a command is blocked at most 2 times and substituted at most 3 times in a row, in an attack sequence. This bash script can gives immediate insight and informs the creation of the rules in the action daemon. It will not mirror exactly the decision making of live malware on HARM but it will prevent complete execution of the commands in *stream.txt*. The resultant schematic for the action daemon is presented in fig. 3.12. The daemon is incorporated in the assessment process as demonstrated in fig. 3.13. The bash scripting associated with the Action Daemon is available in appendix B.4.

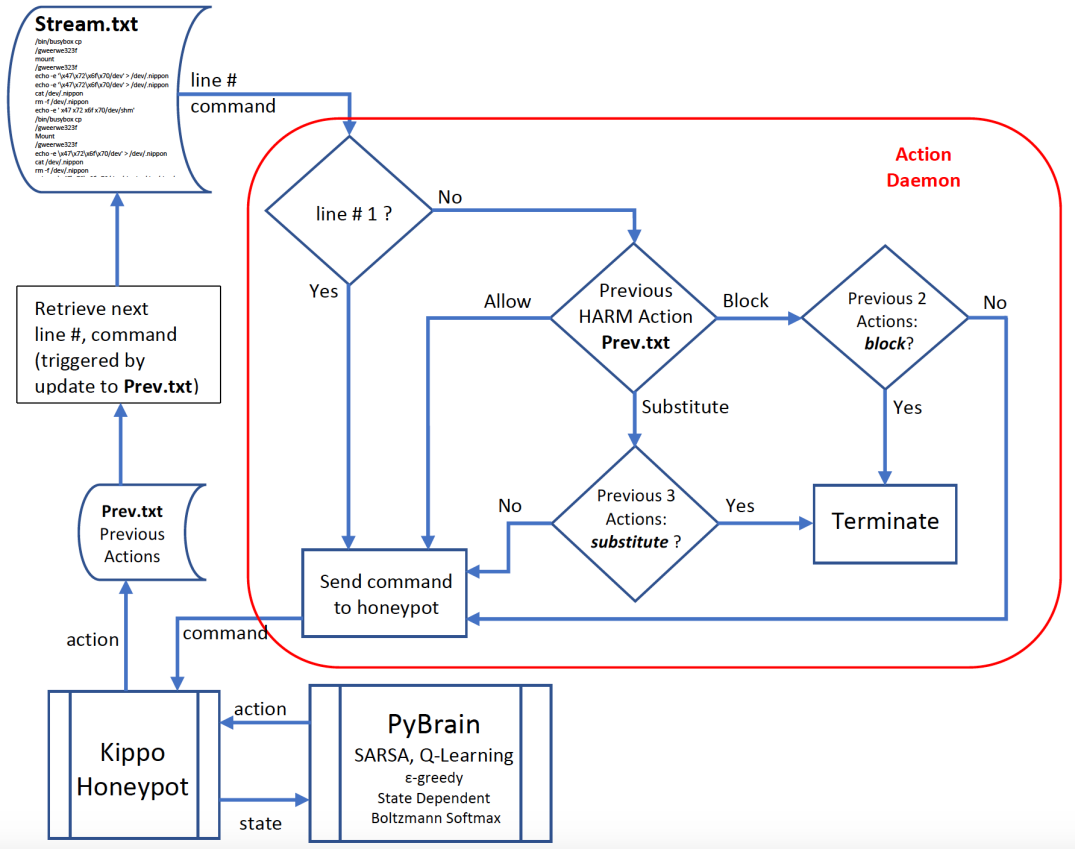


FIGURE 3.12: Action Daemon Functionality

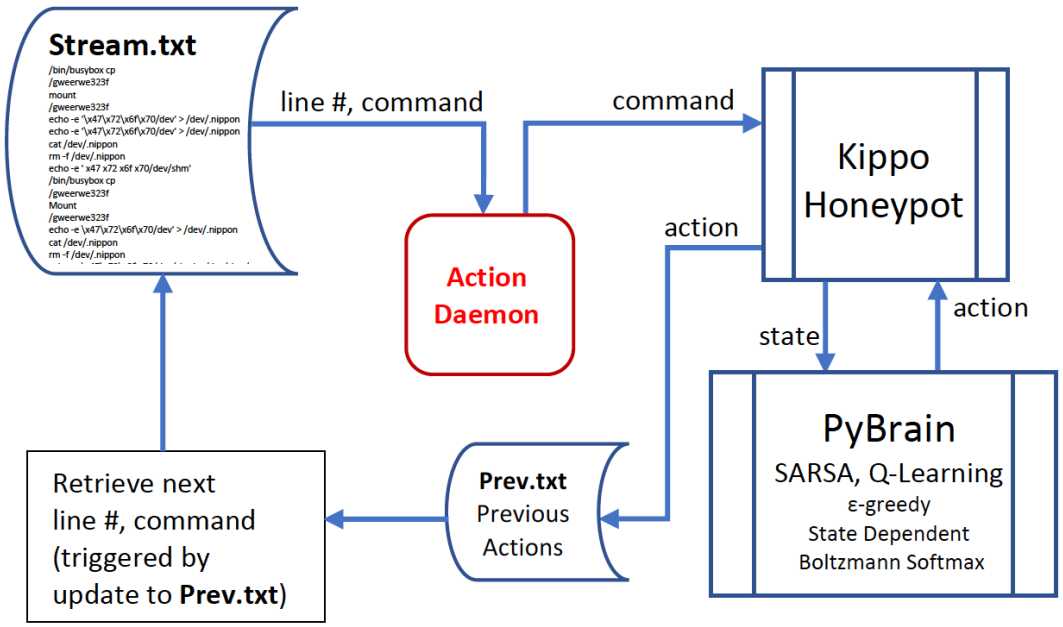


FIGURE 3.13: Incorporating Action Daemon in Assessment Process

3.4 Chapter Summary

The state action space formalism presented in this chapter is differentiated from previous research in that it doesn't provision human responses to human attack interactions. Honeypot activity is predominately automated and concentrating on this aspect of malware provides the best opportunity for analysis. During initial deployment of the IoT honeypot (section 3.2) the honeypot developers stated "*i didn't see human on my kippo in a very long time*" and "*your honeypot is functioning fine*", when queried on GitHub about its functionality. The resultant dataset contained repetitive versions of the trending malware of the time, namely *BillGates Botnet* and *XOR DDoS Flood* traffic.

These characteristics of automation and repetition informed the development of HARM. Removing the *human element* from the reward function of previous research, simplifies the learning process and forms a core contribution. The IoT honeypot outlined in 3.2 was embedded with PyBrain reinforcement learning as seen in 3.3.2.1. Together these functions created an adaptive IoT HARM shown in 3.3.1.3. HARM was installed locally in an **Eclipse**[1] development environment and on **Amazon AWS**[2] web service (Appendix: B.1). The initial installation in the Eclipse environment was to facilitate development and then used to as an optimisation tool for HARM. The AWS installation was a Internet deployment capturing current automated and repetitive malware.

The flexibility of PyBrain regarding Python coding provides for intuitive integration with Kippo and the subsequent modification of HARM for assessment. Learning algorithms, explorers and policies can be easily modified and compiled quickly with new learning variations. Testing these variations requires a data stream representative of real world malware trends and methods. The modified dataset from HARMs live deployment provides this.

It is worth revisiting fig. 1.2. Each research question is explored with a corresponding hypothesis and together they form an approach to improve honeypot functionality. The unique state action space formalism forms a core contribution. Each hypothesis builds upon a previous hypothesis to improve this functionality. The hypotheses themselves are key components in the structure of the proposed framework for adaptive and agile honeypots (chapter 7). The practical elements can also be identified. **Firstly**, The design and testing of HARM is executed in a controlled environment. **Secondly**, HARM is deployed on an AWS instance to facilitate a *live* data capture. **Thirdly**, the captured dataset is analysed further to determine exact events at key milestones in the learning evolution. **Fourthly**, the captured dataset itself is used to examine the effectiveness of reinforcement learning algorithms and their suitability to the current problem domain.

[1] https://www.eclipse.org/
[2] https://aws.amazon.com/

Prior to proceeding to the hypotheses in the next three chapters it is also worth revisiting fig. 3.5. This *concept* diagram is updated and can used as a road map for figures produced by datasets from each hypothesis. The updated concept diagram is displayed in fig. 3.14.

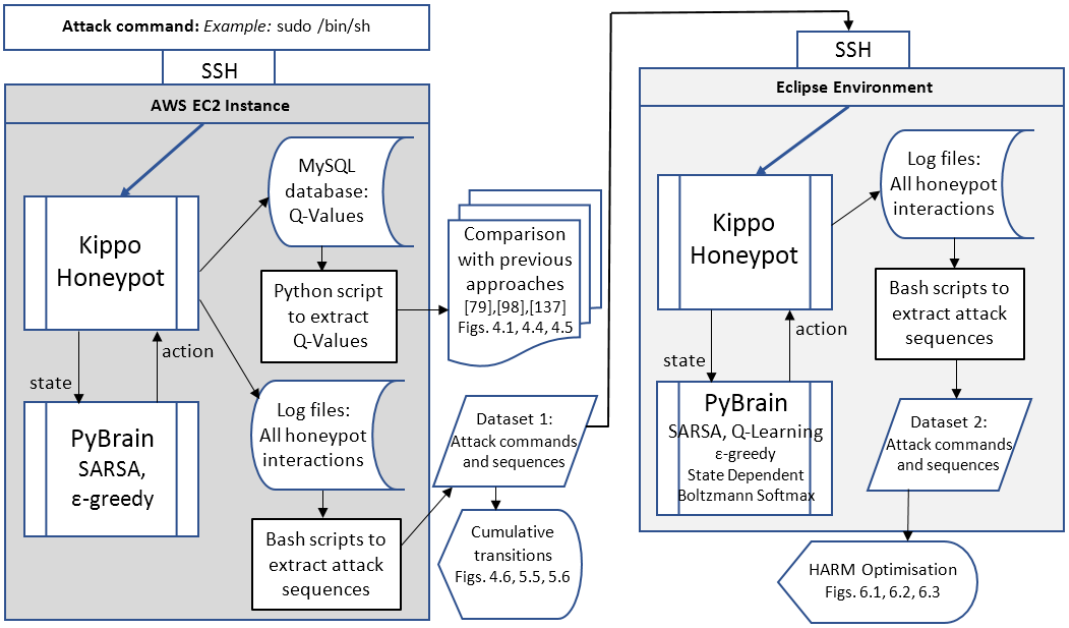


FIGURE 3.14: Datasets and Figures from HARM Deployment and Optimisation

Chapter 4

Hypothesis 1: Improving Adaptive Honeypot Functionality with Reinforcement Learning

A honeypot using a unique state action space formalism, can learn from automated and repetitive malware to prolong interaction. An adaptive honeypot for automated and repetitive malware (HARM) exploits the actual characteristics of malware, namely automation and repetition, to determine the best responses to attack commands.

This hypothesis compares the implementation of HARM's state action space formalism (section 3.3.1.3) to previous research. It evaluates the performance of HARM against previous research using improved reinforcement learning values targeting automated and repetitive malware. It is also explored in journal publication:

- *Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware*, Seamus Dowling, Michael Schukat & Enda Barrett, (2018) **Journal of Cyber Security Technology, 2:2, 75-91**, DOI: https://doi.org/10.1080/23742917.2018.1495375

The goal of this hypothesis is to improve honeypot functionality by increasing attack interactions. This in turn produces a larger dataset with more relevant information. To enhance cyber forensics, data captured on the honeypot should give insight into how malware developers design their attack tools. Prolonging interaction results in the capture of more attack methods and tools. Sandboxing these tools produces the source code designed by the attackers. It is therefore better to maximise both interaction and data capture. This hypothesis explores and modifies previous implementations with reinforcement learning parameters that are more effective for automated and repetitive malware. The reward function is simplified, creating a new state action space formalism (see section 3.3.1.3). Subsection 3.2 empirically proved that malware interactions on honeypots is predominately automated and repetitive. The implementation of this formalism is compared to previous research. Journal publication *Improving adaptive honeypot functionality with efficient reinforcement*

learning parameters for automated malware [137] evaluates the performance of HARM against this previous research.
This simplified reward function is implemented on the adaptive IoT HARM and the following three tasks are pursued:

1. Direct comparison with previous research in a simulated environment. (section 4.1)
2. Deployment of a live HARM targeting automated and repetitive malware. (section 4.2)
3. Comparison of the number of command transitions on a live HARM and standard honeypot, with previous research. (section 4.3)

4.1 Task 1: Direct comparison with previous research in a simulated environment

Two adaptive honeypots using reinforcement learning have been previously proposed. RASSH [98] implemented an improved version of Heliza [79] using Kippo and observed similar behaviour with the dataset. The reinforcement learning algorithm is implemented using PyBrain for both and allowed for comparisons between the two. The RASSH software framework demonstrated similar learning behaviour to Heliza. They both presented an action-value graph for state *wget*, for transition reward 3.6. HARM was created using RASSH as the software framework and the data presented by Heliza. Wagener states that the data ...

> "... is quite valuable for a honeypot operator who does not want to setup Heliza but rather simply wants to install static fake services" [79, p.228].

The same software code with the revised reward formula 3.7 was implemented in a controlled Eclipse environment. In the controlled experiment, the honeypot was attacked 350 times. Each wget attack command populates the Q-Values table as described in subsection 3.3.1. A simple botnet was used to attack the honeypot. This botnet consisted on Linux commands such as *mount*, *ls*, *rm*, etc. representing states in the reinforcement learning environment. Critically it also consisted of a *wget* command allowing for direct comparison with the previous research of RASSH and Heliza. Any forced intelligence espoused in subsection 3.3.2.2 is irrelevant as the focus is on the learning evolution of just *wget* and comparison with the Q-values of previous research. This botnet consisted of Linux bash scripts containing executed commands provided by Heliza.

Development code for the HARM is freely available [138]. However for this task, the code is restricted to learn on one state only, namely *wget*. At the end of the attack sequence, the reward values were extracted from the MySQL database (see appendix C.2). They produced the graph in fig. 4.1. For ease of comparison equivalent graphs from previous research is reproduced in fig. 4.2.

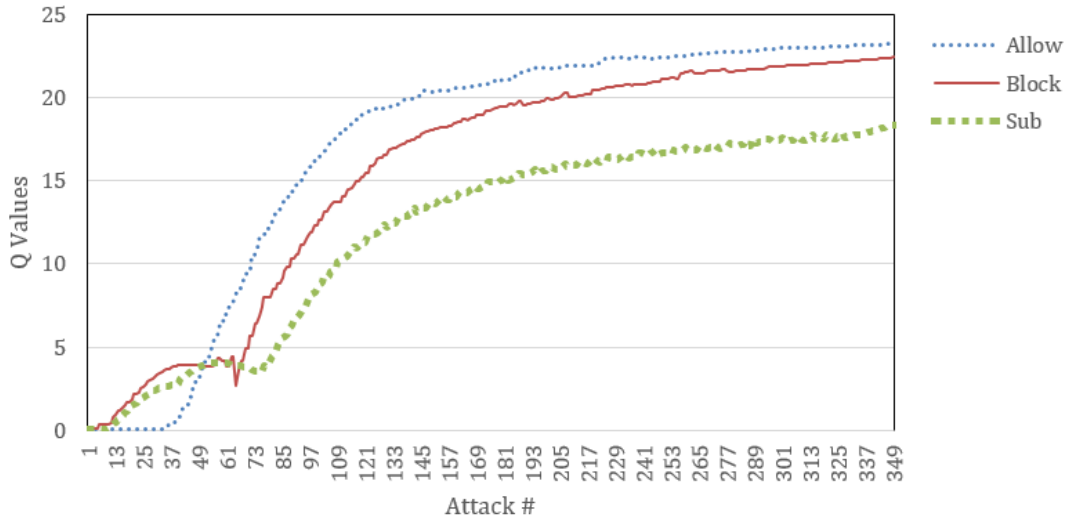
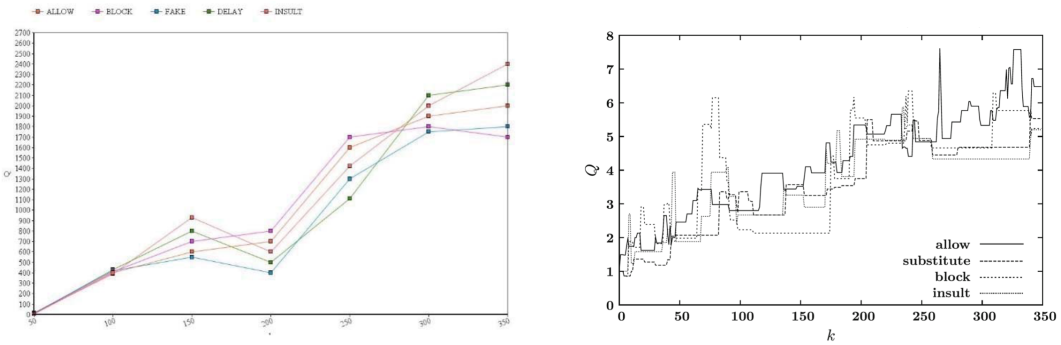


FIGURE 4.1: Learning evolution for *wget* with reward eq. 3.7 in controlled experiment



RASSH evolution for *wget*

Heliza evolution for *wget*

FIGURE 4.2: Learning evolution for *wget* with RASSH [98] and Heliza [79]

The reward evolution in fig. 4.1 is similar to RASSH and Heliza. However two distinct differences exist.

- The learning evolution converges towards the optimum policy for all actions. Both RASSH and Heliza display a linear evolution.
- A clear separation appears at approximately attack 65 as the honeypot determines the best actions to apply to an attack command. RASSH and Heliza do not definitively determine the best action during the attack sequence.

These two distinctions demonstrate a more efficient learning process when compared to previous implementations. Heliza determined best action only at the end of the 350 attack sequences. The results did not demonstrate a move towards convergence but rather a linear evolution. RASSH replicated those results stating that they are "promising" with emphasis on the scalability of the adaptive honeypot. HARM learned very early that allowing an attack returns the best reward value. This began

at attack 35 and returned higher rewards from attack 50 onwards. The honeypot still explored the environment with all actions being selected for the 350 attacks. From approximately attack 120, the honeypot starts to converge towards the optimum policy. It is important to comment on the deployments associated with figs. 4.1 and 4.2.

- Heliza was a deployment on the Internet in 2011 (live).
- RASSH was 'experimental' in 2014 (controlled).
- HARM was from a Eclipse environment in 2018 (controlled).

This would account for the erratic behaviour of Heliza when compared to RASSH or HARM. A live deployment of HARM in task 2 provides for additional comparison.

4.2 Task 2: Deployment of a live HARM targeting automated and repetitive malware

Task 1 deployed HARM in a controlled environment. It examined one state *wget* and presented encouraging results. A live deployment was needed determine if these results are replicated on the Internet and to compare its performance to Heliza specifically. Therefore the HARM from task 1 was modified to generate rewards on 75 most popular Linux states and was deployed on the Internet. AWS EC2 instances are used to facilitate an Internet facing honeypot in the European region. Kippo, PyBrain, MySQL and other dependencies were installed on the instance. EC2 provides for a convenient method of honeypot deployment. It was accessible through SSH and immediately it started to record malware activity. Initially it logged *dictionary* and *bruteforce* attempts. Thereafter it captured other malware traffic including a Mirai-like bot [122]. When the Mirai malware code was made publically available, it spawned a series of variants [3]. One of these variants became the dominant attacking tool over a 30-day period, until over 100 distinct attacks were recorded on the honeypot. The bash scripts within this bot represented states in the learning environment. Table 4.1 provides a sample of the bot scripts. It defines a profile of the scripts as being attacker tools or honeypot states. The attacker tools are relevant for discussion in task 3.

The Mirai-like ssh bot sample in table 4.1 shows that it uses *sudo* and *wget* in its configuration. This facilitates the extraction from the dataset of the rewards given to *wget/action* and *sudo/action*. The SQL code in appendix C.2 was modified to select these actions only (see fig. 4.3). 100 attacks from this bot were collected. Other SSH malware interacted with the honeypot. But these often used different commands (such as *curl*) or were too infrequent to excessively modify the rewards. The graphs produced from these extractions are presented in figs. 4.4 (wget) and 4.5 (sudo).
These figures reflect the findings from fig. 4.1 in task 1. It can be seen that the adaptive honeypot identifies *allow*, as the best action to take at attack 22 (wget) and attack

TABLE 4.1: Mirai Variant

Sequence	Bot Command	Profile
1	/gweerwe323f	Attacker Tool
2	sudo /bin/sh	State
3	/bin/busybox	State
4	/gweerwe323f	Attacker Tool
5	mount	State
6	/gweerwe323f	Attacker Tool
7	echo -e '\x47\x72\x6f\x70/' > //.nippon	State
8	cat //.nippon	State
—-	————————————	————
38	/gweerwe323f	Attacker Tool
39	cat /bin/echo	State
40	cd /	State
41	wget http://Redacted IP/bins/usb_bus.x86 -O - > usb_bus	State
42	chmod 777 usb_bus	State
43	./usb_bus	Attacker Tool
44	/gweerwe323f	Attacker Tool

```
mysql> select * from cases where initial_cmd = "wget";
+-------+-------------+-------------+--------+-----------------------------------+
| id    | initial_cmd | cmd_profile | action | rl_params                         |
+-------+-------------+-------------+--------+-----------------------------------+
| 11178 | wget        |             |      2 | [ 0.1        0.09925    0.0988775] |
| 11179 | wget        |             |      2 | [ 0.1        0.0988775  0.0985675] |
```

FIGURE 4.3: SQL Select statement for *wget/action*

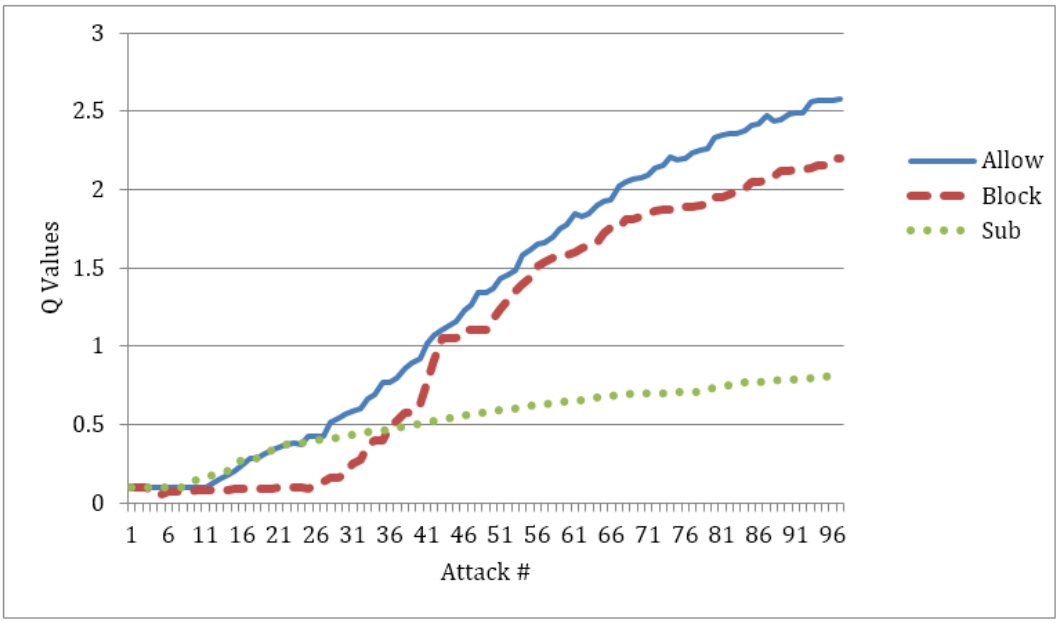


FIGURE 4.4: *wget/action* values for reward *eq.* 3.7 in live Internet deployment

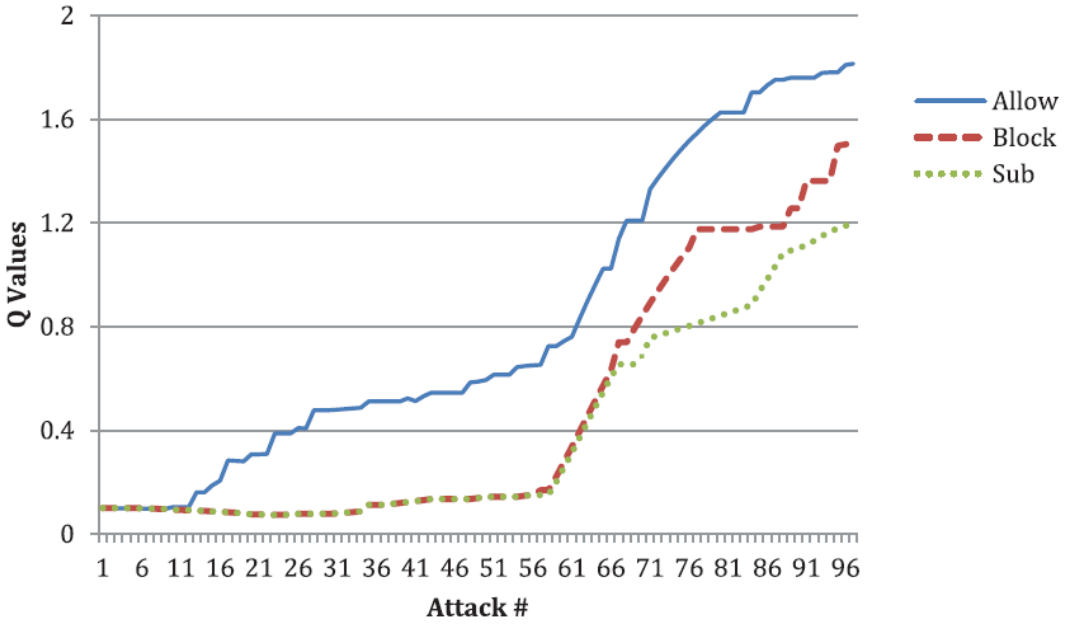


FIGURE 4.5: *sudo/action* values for reward *eq.* 3.7 in live Internet deployment

9 (sudo). Although the honeypot captured 100 Mirai variant attacks, the graphs do present a move towards converging on the optimum learning policy. In both graphs, *allow* and *block* are the actions that produce the best reward. Checking the logs from the honeypot dataset, blocking the commands sometimes resulted in the command being reissued. This immediately contributes to the transition count but more interestingly it gives insight into the operations of the bot, without resorting to sandboxing for code examination. The log files capture both the attack interactions and the reinforcement learning calculated Q-values. A sample of this is shown in fig. 5.4 in chapter 5. This allows for further observation of HARM's decision making at discrete points during interaction with the malware. For example, *sudo* was allowed for the first time at attack number 12, the impact of which is visible in both fig. 4.5 and fig. 5.4

4.3 Task 3: Comparison of the number of command transitions on a live HARM and standard honeypot, with previous research

This task replicates experiments performed by Heliza to quantify the increases in transitions when deployed on the online. Heliza categorises the attack interactions as *attacker tools* and *state commands*. It did this so as to separate all non-state commands from customised attacker tools. These attacker tools gave insight into the linguistic features of the attacker and were used to ascertain human interactive behaviour. HARM has been designed with a simplified state action space formalism for automated and repetitive malware. The very characteristic of automation negates further investigation into human activity for this thesis. However, for comparison purposes the distinction is made between *attacker tools* and *state commands*.

Two Kippo honeypots were deployed on the Internet at the same time; HARM as detailed in task 2 and one standard Kippo deployment. To compare performance, all commands executed on the honeypots were extracted. Kippo stores all interactions in log files as standard. It also stores all downloaded files in a download directory. This makes it possible to accumulate a count of all the commands attempted on the honeypot. These commands all represent interactions post-compromise. Therefore events such as failed attempts, dictionary and bruteforce attacks are excluded as they represent pre-compromise interactions. The dominant interaction came from the Mirai-like SSH bot as seen in table 4.1. Other commands such as *direct-tcpip* and embedded *perl* scripts also appeared infrequently. These were not recognised states on the honeypot and did not contribute to the reward values. The benefit of honeypot datasets to the security community is in the insight gained into the mechanisms used by malware developers. The methods used to gain access to a device, compromise it and then propagate can be captured. As an example we can examine the role honeypots played in the capture and analysis of the Mirai bot [9]. This analysis allowed security groups to detail the methods used to compromise and self propagate. The Asia Pacific IP registration authority, APNIC, produced a detailed graphic form of these methods [139]. The bot infects and kills competing malware, scans for other vulnerable network devices and communicates with the C&C. The data captured as sampled in table 4.1 represents the executable code after a successful bruteforce entry. It can be deconstructed into 3 constituent parts [123]:

1. ascertain information about the host such as underlying architecture and operating system.

2. install software to compromise the host and install hidden daemons to create communication channels to C&C

3. download files from C&C for further compromise, propagation and instruction.

Specifically these 3 parts are identifiable as:

1. *sudo/bin/sh* and */bin/busybox* determines if the host is running 'busybox', a known shell for IoT devices.

2. *echo -e ' \x47 \x72 \x6f \x70/' > //.nippon* creates and executes code. Malware developers use obfuscation to avoid detection [140]. This code has been obfuscated with *\x* handles and decoded from hex to ASCII to create executable code. This code is executed, *> //.nippon*, creating a daemon to communicate with the C&C.

3. *wget http://Redacted IP/bins/usb_bus.x86 -O - > usb_bus* downloads files from the current C&C. This file is executed with the next command in the sequence. In this case the file resembles an existing library but its purpose is to scan for further devices in the address space and cleanup.

The profile in table 4.1 identifies system commands as *states* and other commands as *attacker tools*. The distinction between both is necessary for comparison with Heliza. */gweerwe323f* is a random command sequence designed to generate an error response. This indicates that the bot is interacting with a functioning operating system and is therefore susceptible to compromise. It also ensures that a previous command has been executed prior to continuing. The specific downloaded tool *usb_bus* further compromises the host, scans for other potential victims and masks its' operations. In reality the bot has a sequence of 44 commands [122]. It has 6 commands that can be considered attacker tools. The first 3 of these exist within the first 6 commands in the sequence. The remaining 3 exist in the final 7 commands of the sequence. To assess the performance of HARM with previous research, the quantity of attacker tools were extracted and their cumulative transitions were compared with the high interaction honeypot. The logs from the captured dataset are examined to specifically look at the transition to attacker commands. The results are presented in fig. 4.6. The location of the attacker tools within the sequence produces very linear results. The standard Kippo honeypot only ever encountered three attacker tools throughout its deployment. This is because it only ever interacted with and captured a maximum of 8 commands in the attack sequence. It can be deduced that it failed a check whereby the variant decided that the underlying architecture was virtual or suspicious or not viable, from a command response such as *mount* which appeared early in the attack sequence. Similarly HARM initially had only three attacker tool as it learned from individual attacks and increased its interactions. At attack 42 HARM increased interaction sufficiently to capture more attacker tools and continued this increase for the duration of the deployment. It demonstrates that HARM does perform better than a standard Kippo honeypot. It performs 63% better in capturing attacker related tools. However the performance of HARM overall is considerably better than just 63% (see section 5.3). It actually captures four times more transitions. In this deployment, the dominant malware encountered had its attacker tools located at opposite

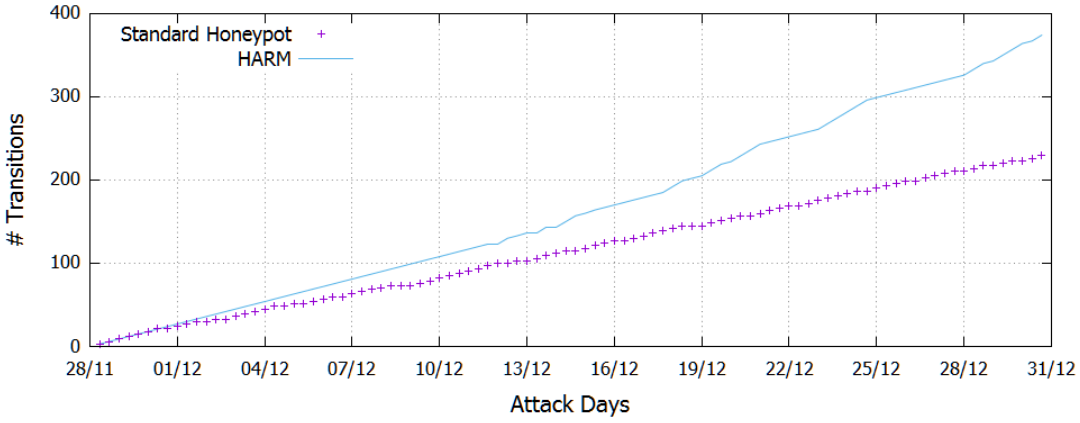


FIGURE 4.6: Cumulative transitions for attacker tools

ends of the attack sequence. Depending on the malware type, this performance will vary as the bespoke attacker tool commands will appear at differing positions in the attack sequence. The difference in time (years) between the deployment of Heliza and HARM make it impossible to make an exact comparison. Live deployments will invariably capture very different malware types

4.4 Discussion

The title and resultant framework involves a cyclic process of development, deployment and optimisation. This hypothesis represents *development* and will be improved upon during the subsequent cyclic processes. It is important that this phase is designed initially to maximise interactions. The learning agent is rewarded using the reward function in equation 3.7. Previous research may have encountered higher volumes of human interactions necessitating their relevant reward function, equation 3.6. It is argued in section 2.1 that a rapid expansion of technological concepts have facilitated the evolution of malware. This is evidenced by the dominance of automation and repetition in captured datasets associated with the honeypot deployed and detailed in section 3.2. And this is critical in the context of both this hypothesis and the thesis as a whole. Longitudinal honeypot deployments will become irrelevant against evolving malware. The deployment and optimisation processes are discussed in hypotheses 2 and 3 respectively. Getting the best start point for this cyclic process is critical and is the focus of this hypothesis. Malware is evolving therefore our state action space formalism has evolved when compared with previous research. In controlled and live environments, HARM demonstrated improvements in learning, action selection and command transitional accumulation.

Noteworthy observations are the deployment period and individual attacks encountered on HARM in the live environment. HARM encountered a dominant malware variant and collected 100 distinct attacks. Repetitive attacks of identical automated malware contributed to HARM's learning evolution. It optimised action

selection resulting in increased attack command transitions. It also obtained the entire attack sequence for the dominant variant after 19 days. This informs hypotheses 2 and 3, the cyclic process of improvement culminating in the proposed framework. Another consideration for this hypothesis is the dates in years between the comparative deployments of Heliza, RASSH and HARM. Each honeypot will have encountered completely different malware types and methods. This specifically impacted on task 3 whereby the categorisation and location of the attacker tools in the attack sequence, impacted the results represented by fig. 4.6. HARM demonstrated better learning and overall data capture. Except in task 3. It is impossible to accurately correlate and compare with previous research due to the evolution of malware in the intervening years. Ironically, it is task 3 that makes the strongest case for agility in honeypot development and deployment.

Chapter 5

Hypothesis 2: Overcoming Inadvertent Attack Termination

Integrating reinforcement learning with a novel state action space formalism for automated and repetitive malware, honeypots can covertly overcome detection and termination methods, realise entire attack sequences and capture larger datasets.

Analysis of the resultant dataset from HARM's deployment (chapter 4) is covered in this hypothesis. Malware trends create dominant attack interactions on honeypots. HARM's performance expedited the realisation of a dominant attack sequence. HARM's interactions concealing the functionality of the underlying honeypot technology is also presented in journal publication:

- *Using Reinforcement Learning to Conceal Honeypot Functionality*, Dowling, S., Schukat M., Barrett E. (2019) In: Brefeld U. et al. (eds), **Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018. Lecture Notes in Computer Science, vol 11053**. Springer, Cham.
  DOI: https://doi.org/10.1007/978-3-030-10997-4_21

Honeypot efficacy depends upon its ability to realistically replicate devices and services. Honeypots are designed to simulate and emulate available services. LiHPs are designed to simulate the basic behaviour of a specific system. If the rules of the system are challenged by malware then the simulation fails. Similarly HiHPs often emulate and behave like real systems. But this might be performed in a different environment such as in virtual honeypots. Again the rules of the emulated system are fixed and the honeypot will respond as the system responds. Malware developers use automated code to assess a compromised system, to decide whether to continue further or to terminate the attack. Complex code structures provide the mechanism for malware to determine this [10]. Post compromise, malware proceeds in a structured manner to determine viability [93]. Responses by a honeypot, either simulated or emulated, can inadvertently terminate further attack interaction. Cloud services makes it easy to deploy honeypots. Step-by-step guides help to setup honeypots with default settings very quickly [141, 142]. Because of this, honeypots give default responses to attack commands. These responses belie the presence of the honeypot

or malware determines there is no worth in continuing. Malware developers became aware of the existence honeypots, capturing and analysing attack activity. To counter this, dedicated honeypot detection tools were developed and evasion techniques we designed into malware [119]. Tools such as Honeypot Hunter performed tests to identify a honeypot [8]. False services were created and connected to by the anti-honeypot tool. Honeypots are predominately designed to prolong attacker interaction and therefore pretend to facilitate the creation and execution of these false services. This immediately tags the system as a honeypot. With the use of virtualisation for honeypot deployments [6], anti-detection techniques issued simple kernel commands to identify the presence of virtual infrastructure instead of bare metal [94]. New versions of honeypots are redesigned and redeployed continuously to counter new malware methods. The Mirai botnet spawned multiple variants targeting IoT devices [3]. Variants were captured on Cowrie, a newer version of Kippo. Analysis of the Mirai variants found that Cowrie failed initial anti-honeypot tests before honeypot functionality was manually amended [121]. Examining the structure of the Mirai variant (table 4.1), when a *mount* command is issued, the honeypot returns a standard response at which point the attack ends.

5.1 Honeypot Data Capture

To counter early termination of attacks, HARM can use the adaptive learning from *Task3* in chapter 4, to learn the best responses to overcome detection and inadvertent termination of the attack. It uses the automated and repetitive characteristics of malware to prolong interaction and realise an entire attack sequence. To demonstrate this, the structure of the data capture first needs to be examined. Kippo stores all activity in folders. All activity on the honeypot such as probes, attempted logins, successful logins and unsuccessful attempts are stored in a *log* directory. On day 1, *kippo.log.1* is created and records all activity until this text based file reaches 1 megabyte. Then *Kippo.log.2* is created and the process continues. For every successful login to the honeypot, the log file records the creation of an individual session file. This session file is stored in binary form in a *tty* directory. It can be executed with the *playlog.py* python script provided by Kippo, showing the attack as it happened in real-time. The log file in a standard honeypot records all connections made to the honeypot. Some of these connections are not successful such as bruteforce or dictionary attacks and do not impact on the reinforcement learning algorithm. Fig. 5.1 demonstrates the data capture format for a standard high interaction Kippo honeypot. It can be seen that date/time, process number (240) and source IP address (61.174.54.228) are recorded for all actions on the honeypot. Further information is then provided on how the honeypot handles the attack commands. Once a command is received, the honeypot opens the aforementioned 'TTY' file. It then attempts to handle each command. In this case, the honeypot catered for the first 8 commands in table 4.1 before losing the connection.

HARM has been designed to record the same information as Kippo but with notable enhancements. It displays the SARSA reward values for the state/space interactions as they are calculated by PyBrain. It also identifies the action performed: 0-Allow, 1-Block and 2-Substitute. If the command is an element of Y in equation 3.7 then *Command found* is displayed. *Command not found* otherwise. Fig.5.2 displays these enhancements from a HARM log file.

```
2017-11-25 07:06:41+0000 [HoneyPotTransport,240,61.174.51.228] NEW KEYS
2017-11-25 07:06:41+0000 [HoneyPotTransport,240,61.174.51.228] starting service ssh-userauth
2017-11-25 07:06:42+0000 [SSHService ssh-userauth on HoneyPotTransport,240,61.174.51.228] root trying auth none
2017-11-25 07:06:43+0000 [SSHService ssh-userauth on HoneyPotTransport,240,61.174.51.228] root trying auth keyboard-interactive
2017-11-25 07:06:44+0000 [SSHService ssh-userauth on HoneyPotTransport,240,61.174.51.228] login attempt [root/admin] succeeded
2017-11-25 07:06:44+0000 [SSHService ssh-userauth on HoneyPotTransport,240,61.174.51.228] root authenticated with keyboard-interactive
2017-11-25 07:06:44+0000 [SSHService ssh-userauth on HoneyPotTransport,240,61.174.51.228] starting service ssh-connection
2017-11-25 07:06:45+0000 [SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] got channel session request
2017-11-25 07:06:45+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] channel open
2017-11-25 07:06:47+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] executing command "/gweerwe323f
2017-11-25 07:06:47+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] exec command: "/gweerwe323f
2017-11-25 07:06:47+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Opening TTY log: log/tty/20171125-070647-3898.log
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Running exec command "/gweerwe323f
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] CMD: /gweerwe323f
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command not found: /gweerwe323f
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command found: sudo /bin/sh
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command not found: /bin/busybox
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command not found: /gweerwe323f
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command found: mount
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command not found: /gweerwe323f
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Reading txtcmd from "/home/kippo/kippo/trunk/txtcmds/bin/mount"
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command found: echo -e '\x47\x72\x6f\x70/' > //.nippon
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Reading txtcmd from "/home/kippo/kippo/trunk/txtcmds/bin/mount"
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] Command found: cat //.nippon
2017-11-25 07:06:48+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] sending close 0
2017-11-25 07:06:49+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,240,61.174.51.228] remote close
2017-11-25 07:06:49+0000 [HoneyPotTransport,240,61.174.51.228] connection lost
```

FIGURE 5.1: Kippo log file at first contact with Mirai variant

```
2017-11-28 15:49:48+0000 [HoneyPotTransport,1,103.41.124.33] NEW KEYS
2017-11-28 15:49:48+0000 [HoneyPotTransport,1,103.41.124.33] starting service ssh-userauth
2017-11-28 15:49:48+0000 [SSHService ssh-userauth on HoneyPotTransport,103.41.124.33] root trying auth none
2017-11-28 15:49:48+0000 [SSHService ssh-userauth on HoneyPotTransport,103.41.124.33] root trying auth keyboard-interactive
2017-11-28 15:49:51+0000 [SSHService ssh-userauth on HoneyPotTransport,103.41.124.33] login attempt [root/123456] succeeded
2017-11-28 15:49:51+0000 [SSHService ssh-userauth on HoneyPotTransport,103.41.124.33] root authenticated with keyboard-interactive
2017-11-28 15:49:51+0000 [SSHService ssh-userauth on HoneyPotTransport,103.41.124.33] starting service ssh-connection
2017-11-28 15:49:51+0000 [SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] got channel session request
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] channel open
2017-11-28 15:49:51+0000 [SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] got global no-more-sessions@openssh.com request
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] pty request: xterm-256color (24, 80, 560, 336)
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Terminal size: 24 80
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] request_env: '\x00\x00\x00\x04LANG\x00\x00\x00\x0ben_IE.UTF-8'
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] getting shell
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Opening TTY log: log/tty/20171128-154951-3127.log
2017-11-28 15:49:51+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] /etc/motd resolved into /etc/motd
2017-11-28 15:49:55+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: /gweerwe323f
2017-11-28 15:49:55+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command not found: /gweerwe323f
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: sudo /bin/sh
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command found: sudo /bin/sh
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] isAllow start
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] case is saving [ 0.1    0.1    0.0995]
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] getAction start sudo
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] get current command -============================== sudo
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] do action ====================================
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Action performed:  [ 0.]
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] ******************** [ 0.1    0.1    0.0995]
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] saving the case ...
2017-11-28 15:49:59+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] case is saving [ 0.1    0.1    0.0995]
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: /bin/busybox cp
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command not found: /bin/busybox
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: /gweerwe323f
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command not found: /gweerwe323f
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: mount
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command found: mount
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] isAllow start
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] case is saving [ 0.1    0.1    0.0995]
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] getAction start mount
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] get current command -============================== mount
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] do action ====================================
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Action performed:  [ 1.]
2017-11-28 15:50:03+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] ******************** [ 0.0995    0.056951    0.0995]
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: /gweerwe323f
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command not found: /gweerwe323f
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: echo -e '\x47\x72\x6f\x70/' > //.nippon
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command found: echo
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] start to get action ..
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] getAction start echo
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] get current command -============================== echo
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] do action ====================================
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Action performed:  [ 1.]
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] ******************** [ 0.0985675    0.05695075    0.09925]
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] saving the case ...
2017-11-28 15:50:06+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] case is saving [ 0.0985675    0.05695075   0.09925]
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] CMD: cat //.nippon
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Command found: cat
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] start to get action ..
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] getAction start cat
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] get current command -============================== cat
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] do action ====================================
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] Action performed:  [ 1.]
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] ******************** [ 0.0985675    0.06925467    0.0985675]
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] saving the case ...
2017-11-28 15:50:09+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,1,103.41.124.33] case is saving [ 0.0985675    0.06925467    0.0985675]
2017-11-28 15:50:37+0000 [HoneyPotTransport,1,103.41.124.33] Got remote error, code 11
    reason: disconnected by user
```

FIGURE 5.2: HARM log file at first contact with Mirai variant

5.2 Realisation of Entire Attack Sequence

This hypotheses examines and compares exact state/space interactions with malware for the two honeypots detailed in section 4.3. As stated, Cowrie and Kippo records all interactions in log files and facilitates the storage of all attempted downloads. In doing so it is possible to examine key points in the dataset so as to examine state/action decisions. Again it is important to revisit the format of the dominant bot encountered on the honeypots (table 4.1). It has a sequence of 44 commands. The high interaction IoT honeypot only ever experienced the first 8 commands in the sequence before the attack terminated. This is evident in fig. 5.1. During 100 captures of the same bot, it never captured more than 8 transitions in the attack sequence. Fig. 5.3 represents the log file from the high interaction Kippo honeypot after nearly 100 separate interactions with the Mirai variant.

```
2017-12-20 17:12:08+0000 [HoneyPotTransport,861,103.107.117.14] NEW KEYS
2017-12-20 17:12:08+0000 [HoneyPotTransport,861,103.107.117.14] starting service ssh-userauth
2017-12-20 17:12:09+0000 [SSHService ssh-userauth on HoneyPotTransport,861,103.107.117.14] root trying auth none
2017-12-20 17:12:10+0000 [SSHService ssh-userauth on HoneyPotTransport,861,103.107.117.14] root trying auth keyboard-interactive
2017-12-20 17:12:11+0000 [SSHService ssh-userauth on HoneyPotTransport,861,103.107.117.14] login attempt [root/admin] succeeded
2017-12-20 17:12:11+0000 [SSHService ssh-userauth on HoneyPotTransport,861,103.107.117.14] root authenticated with keyboard-interactive
2017-12-20 17:12:11+0000 [SSHService ssh-userauth on HoneyPotTransport,861,103.107.117.14] starting service ssh-connection
2017-12-20 17:12:12+0000 [SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] got channel session request
2017-12-20 17:12:12+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] channel open
2017-12-20 17:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] executing command "/gweerwe323f
2017-12-20 17:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] exec command: "/gweerwe323f
2017-12-20 17:12:14+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Opening TTY log: log/tty/20171220-171209-1516.log
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Running exec command "/gweerwe323f
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] CMD: /gweerwe323f
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command not found: /gweerwe323f
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command found: sudo /bin/sh
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command not found: /bin/busybox
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command not found: /gweerwe323f
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command found: mount
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command not found: /gweerwe323f
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Reading txtcmd from "/home/kippo/kippo/trunk/txtcmds/bin/mount"
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command found: echo -e '\x47\x72\x6f\x70' > //.nippon
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Reading txtcmd from "/home/kippo/kippo/trunk/txtcmds/bin/mount"
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] Command found: cat //.nippon
2017-12-20 17:12:15+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] sending close 0
2017-12-20 17:12:16+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,861,103.107.117.14] remote close
2017-12-20 17:12:16+0000 [HoneyPotTransport,861,103.107.117.14] connection lost
```

FIGURE 5.3: Kippo log file after 98 interactions with Mirai variant

The logs identify the state/action pair used by HARM for each attack command. This is identified as *getcurrent command/Action performed* in fig 5.4. At discrete points in all the attacks it is possible to observe HARMs functionality. But this requires close examination of each log and the malware behaviour. For example, HARM initially experienced only the first 8 commands in the attack sequence (fig 5.2). But then the sequence count started to increase as the honeypot learned from its previous interactions. Examining the entire bot structure, a *mount* command appears in the initial command sequence. A standard honeypot has preconfigured parameters, and the standard response of *mount* is a known honeypot response [121]. Therefore the *mount* response values were changed within Kippo. At attack 16, HARM increased the number of commands in the sequence to 12 for the first time. Examining the logs, we find that it substituted a result for *cat* command for the first time. This is displayed in fig. 5.4. This can be found on log 208, process 17164 in the dataset identified in appendix A.3. This requires identifying the attack type, the log file, the time stamp, the commands and associated actions logged by HARM. It is only by close inspection of the log files that attack behaviours begin to appear.

```
2017-12-05 15:27:37+0000 [HoneyPotTransport,17164,122.225.109.120] NEW KEYS
2017-12-05 15:27:37+0000 [HoneyPotTransport,17164,122.225.109.120] starting service ssh-userauth
2017-12-05 15:27:37+0000 [SSHService ssh-userauth on HoneyPotTransport,17164,122.225.109.120] root trying auth none
2017-12-05 15:27:37+0000 [SSHService ssh-userauth on HoneyPotTransport,17164,122.225.109.120] root trying auth keyboard-interactive
2017-12-05 15:27:37+0000 [SSHService ssh-userauth on HoneyPotTransport,17164,122.225.109.120] login attempt [root/123456] succeeded
2017-12-05 15:27:37+0000 [SSHService ssh-userauth on HoneyPotTransport,17164,122.225.109.120] root authenticated with keyboard-interactive
2017-12-05 15:27:37+0000 [SSHService ssh-userauth on HoneyPotTransport,17164,122.225.109.120] starting service ssh-connection
2017-12-05 15:27:37+0000 [SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] got channel session request
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] channel open
2017-12-05 15:27:37+0000 [SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] got global no-more-sessions@openssh.com request
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] pty request: xterm-256color (24, 80, 560, 336)
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Terminal size: 24 80
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] request_env: '\x00\x00\x00\x04LANG\x00\x00\x00\x0ben_IE.UTF-8'
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getting shell
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Opening TTY log: log/tty/20171205-152737-1383.log
2017-12-05 15:27:37+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] /etc/motd resolved into /etc/motd
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command not found: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: sudo /bin/sh
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: sudo /bin/sh
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] isAllow start
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start sudo
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ sudo
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 1.]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: /bin/busybox cp
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command not found: /bin/busybox
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command not found: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: mount
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: mount
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] isAllow start
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488     0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start mount
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ mount
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 0.]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488     0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command not found: /gweerwe323f
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: echo -e '\x47\x72\x6f\x70/' > //.nippon
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: echo
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start echo
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ echo
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 1.]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488    0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488    0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: cat //.nippon
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: cat
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start cat
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ cat
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 0.]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD: rm -f //.nippon
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: rm
2017-12-05 15:27:38+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start rm
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ rm
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 0.]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488     0.11285488    0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488     0.11285488    0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD:   echo -e '\x47\x72\x6f\x70/tmp' > /tmp/.nippon
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: echo
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start echo
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ echo
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 1.]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD:   cat /tmp/.nippon
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: cat
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start cat
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ cat
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 2.]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] CMD:   rm -f /tmp/.nippon
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Command found: rm
2017-12-05 15:27:39+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] start to get action ..
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] getAction start rm
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] get current command -============================ rm
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] do action ==================================
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Action performed:  [ 0.]
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] ****************** [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] saving the case ...
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] case is saving [ 0.511285488      0.11285488     0.11285488]
2017-12-05 15:27:40+0000 [SSHChannel session (0) on SSHService ssh-connection on HoneyPotTransport,17164,122.225.109.120] Got remote error, code 11
reason: disconnected by user
```

FIGURE 5.4: HARM log file after 16 interactions with Mirai variant

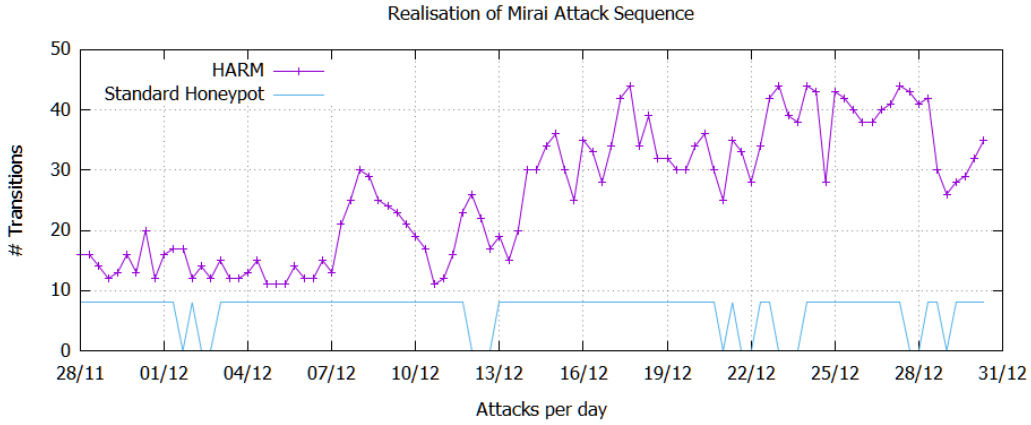


FIGURE 5.5: Attack Interaction Evolution for Mirai Variant

At attack 22 HARM blocked a compound *echo* command causing the attack sequence to increase. It continued to learn until attack 59 when it finally realised the entire attack sequence for the Mirai variant (17^{th} December 2107). Fig. 5.5 graphically demonstrates the realisation of the Mirai-like bot command sequence for both honeypots. HARM is deployed with ϵ-greedy set to = 0.05 allowing it to continue to explore resulting in increases and decreases in the capture of the variant. This is a very interesting result when compared to the standard honeypot's linear accumulation. There exists many similar honeypots that are deployed to capture data on various attack vectors [68]. These operate for long periods capturing repetitive, automated and incomplete attack sequences. It is pointed out that the repetitive, automated variant had 44 commands in the attack sequence. Fig. 5.5 clearly shows that the state action space formalism for automated, repetitive malware quickly rewarded the learning agent until all commands in the attack sequence were realised. It also shows that HARM realised this at attack 59 making approximately 45% of the subsequent attack interactions and data collection redundant. The frequency of the Mirai variant was uniform for the duration of the deployments. The repetitive nature of the bot provides the adaptive environment with the opportunity to learn from each attack iteration.

5.3 Increased Data Capture

Any honeypot deployed on the Internet with a popular attack vector such as SSH, HTTP, HTTPS, etc. will capture malicious activity. This activity can be pre-compromise or post-compromise. Pre compromise activity is invariably bruteforce and dictionary attacks. This can be controlled by observing the source IP of this activity and restricting its ingress with access lists locally or on a cloud platform such as AWS. Pre-compromise activity can provide important information for research such as source, frequency and login credentials. But is not relevant for this thesis which is interested in post-compromise activity. This activity is automated showing no human cognitive behaviour and appears frequently. The most frequent attack was the Mirai variant which appeared at least once every 8 hours or the duration of the deployment of HARM and the standard Kippo honeypot.

The standard Kippo honeypot only ever experienced the first 8 commands in the Mirai command sequence before the attack terminated. HARM initially only experienced the first 8 commands but then the sequence count started to increase as the honeypot learned from its state actions and rewards. Reward 3.7 has a simple goal of maximising transitions. It explores the environment and exploit previous knowledge prolonging attack interaction. Fig. 5.6 presents a comparison of the cumulative transitions for both honeypots. This includes all command transitions for state and attacker tool commands. The high interaction honeypot only ever executed 8 commands before the attack ended. This accounts for the linear accumulation of transitions.

Overall it can be seen that HARM captured approximately 4 times more attack commands than the standard honeypot. This task demonstrates that HARM significantly increased the size of the dataset that can be captured on a honeypot. Although the dominant trending malware captured was the Mirai variant, other malware interacted on HARM. Some of these lesser malware types had commands which were facilitated by HARMs preconfigured 75 states. These generated rewards and contributed to the learning evolution and cumulative counts. Some were not facilitated but did contribute to the cumulative count. These were included in the transition extractions as seen in appendix C.4. HARM increased the dataset size as it learned the best actions to take to prolong interaction. The purpose of a honeypot is to capture a dataset to evaluate attack behaviour. Not only did HARM increase the dataset size captured but more importantly, it captured a *better* dataset with much more relevant attack information. In this case better has been demonstrated in section 5.2 in the realisation of the entire attack sequence.

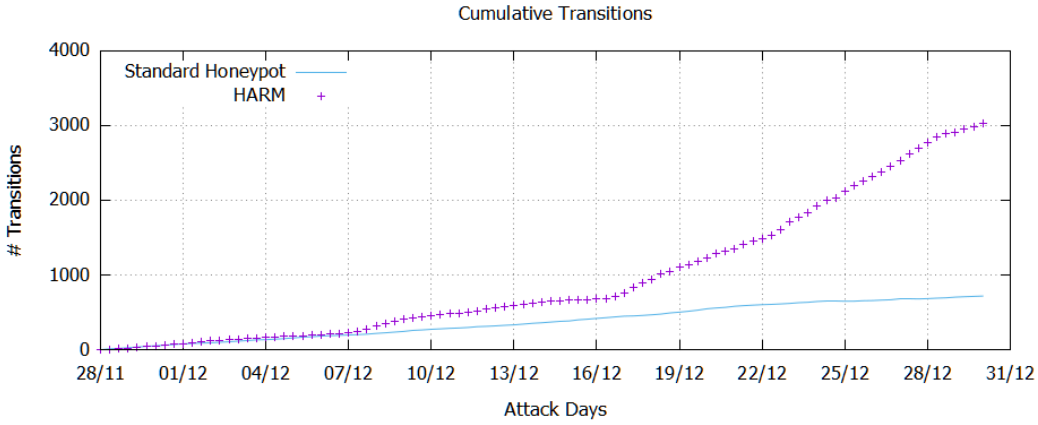


FIGURE 5.6: Cumulative transitions for all commands

5.4 Discussion

Honeypot technology can become irrelevant as new malware, variants and methods emerge. Honeypots will still capture large datasets of truncated attacks as malware determines that:

- a - the compromised host is not real
- b - the compromised host is not viable

If it's "a" then malware has determined that the system is not real. If it's "b" then malware decides that the device can not fulfil it's end goal as a compromised device. All honeypots, be they LiHP, MiHP or HiHP can only simulate real systems. Automated malware presents an inflexible set of commands that are executed or not executed by the honeypot. If a command is *not* executed then the attack fails. Or if a command is *not* executed then the honeypot evades a trap, such as false services, designed into malware operations to determine underlying honeypot activity. The inverse is also true as executed commands can both prolong and terminate an attack. Therefore blocking a command can be positive of negative towards prolonging attack interaction. This all contributes to ambiguity concerning honeypot functionality. In an attempt to be *real*, honeypots are inadvertently terminating attacks. Adaptive honeypots can overcome this issue by exploiting the repetitive nature of malware and learn the best response to attack commands over time. Large datasets of truncated do not contribute to immediate malware threats.

This hypothesis examines the dataset from hypothesis 1 in more depth. It concentrated on the dominant variant and assessed HARMs' operations at distinct points in the learning evolution. It was evident from a cursory glance at the log files that HARM captured fewer commands during initial attack interactions, than during later interactions. This prompted the extraction of all Mirai variant interactions and further examination of the command sequences. It is only by close inspection of the log files that attack behaviours begin to appear. This requires identifying attack types and patterns in their attack sequences.

The realisation of the entire sequence after 19 days of the deployment acts as a catalyst for exploring hypothesis 3 and the cyclic process of continual improvement. An obvious impact of this improved data capture is the size of the resultant dataset. HARM captured so much more nuanced information both in quantity and quality. The realisation of an attack sequence after such a short period highlights the need for shorter deployment periods. Honeypots deployed for long periods capture the same information repetitively. This may have some advantages for longitudinal analysis but not for immediate threats. Hypothesis 3 examines the idea of shorter deployment periods being continually optimised. It is informed by the in dept analysis of the log files that can only be done by literally reading them. Once patterns and malware types start to emerge, coding as presented in appendix C extracts the information hidden within.

Chapter 6

Hypothesis 3: Evaluating Adaptive Performance for Optimised Redeployment

Reinforcement learning algorithms and policies can be evaluated using current captured datasets in a controlled environment. This allows for the optimisation of subsequent deployments for immediate malware threats.

Agile Honeypot design is explored in this hypothesis. Initial optimisation techniques from journal publication *Using Reinforcement Learning to Conceal Honeypot Functionality* in hypothesis 2 acted as a catalyst for further optimisation of reinforcement learning algorithms and explorer policies. These findings are presented in journal publications:

- *Self HARMing Malware: An Adaptive IoT Honeypot for Automated, Repetitive Malware*, In **Concurrency and Computation: Practice and Experience** https://onlinelibrary.wiley.com/journal/15320634, (under review - See appendix D.1)
- *A Framework for Adaptive and Agile Honeypots*, Seamus Dowling, Michael Schukat, and Enda Barrett, 2019, In **ETRI Journal: Special issue on Cyber Security and AI** https://onlinelibrary.wiley.com/journal/22337326, (under review- See appendix D.2)

The previous hypotheses demonstrate that the integration of reinforcement learning with efficient parameters for automated and repetitive malware, improve honeypot functionality resulting in improved data capture. Hypothesis 3 can be considered the *final* step in this process. Or it can be considered to be the *next* step in the cyclic process as detailed in fig. 1.2. It aims to build upon the previous improvements to optimise adaptive honeypot operations for further deployments. Hypothesis 2 (chapter 5) demonstrates that the improved learning in hypothesis 1 (chapter 4) resulted in the realisation of an entire attack sequence after a short period. To challenge HARM further, it should be optimised to reduce the realisation period. Reinforcement learning values such as algorithms, explorer policies and parameters

may perform better against other malware variants and methods. This can inform honeypot operators about the effectiveness of time specific honeypot deployments. New malware variants are spawned with the release of the source code [9]. Having a honeypot deployed for long periods may not be the most effective method of collecting relevant data. This is also the case for a *static* adaptive honeypot that is not continuously assessed. The evolving nature of malware makes longitudinal deployments very redundant if they are collecting masses of automated and repetitive data. Immediate threats can be identified faster if shorter deployment periods identify changes in the learning evolution of the adaptive honeypot.

PyBrain is an intuitive machine learning library. It provides for the implementation of standard and some advanced algorithms using python and facilitates supervised, unsupervised and reinforcement learning algorithms. It has been stated that reinforcement learning has proven efficacy in solving MDPs where an incomplete model exists. This is the case for a honeypot environment. Therefore optimisation involves examining the effectiveness of Q-learning and SARSA under a variety of policy configurations. Two elements need to be considered for this optimisation. The first element is the reuse of the captured dataset on the Internet deployed adaptive IoT HARM. The second element is the selection of algorithms and policies as provided for by PyBrain configurations within a controlled environment. Section 3.3.2 details the code function within PyBrain which facilitates the implementation of SARSA and Q-Learning algorithms and explorer policies. This flexibility within the controlled environment provides for further optimization of HARM. By modifying reinforcement learning values within the controlled environment and streaming a captured dataset, an optimum version of HARM can be determined. The reinforcement learning values that are modified within PyBrain are:

- Algorithms: SARSA, Q-Learning
- Explorer Policies: ϵ-greedy, Boltzmann Softmax, State dependent,
- Parameters: ϵ-greedy values, alpha(α), gamma(γ)

6.1 Optimisation

The optimisation process examines the performance of HARM configured with varying combinations of learning algorithms and explorer policies, against the malware variant in table 4.1. There is a multitude of possible combinations of reinforcement learning values that can be assessed with the optimisation process. The dataset from the deployment process (chapter 5) was streamed through the adaptive honeypot in a controlled Eclipse environment. The dataset is modified to reflect malware methods and is detailed in subsection 3.3.2.2. It involves the intervention of the action daemon to replicate HARM's behaviour during a live deployment. The daemon replaces the ϵ-greedy was streamed three times with settings of 0.05, 0.10 and 0.15. Therefore for SARSA, the dataset was streamed five times; once each for Boltzmann

Softmax and State Dependent, three times for ϵ-greedy. The process was then repeated for Q-Learning.

Therefore, the dataset was streamed ten times. The first sequence (five times) with the learner set to SARSA, the second sequence with the learner set to Q-Learning. The honeypot captured the interactions, actions and number of commands in the Mirai variant attack sequence that represent transitions from s to s_{t+1}. These transitions were collated and HARM was reset after each streaming. Fig. 6.1 and fig. 6.2 demonstrates the performance of the explorer policies for SARSA and Q-Learning respectively. It can be seen that SARSA and State Dependent realised the entire attack sequence after 38 iterations (fig.6.1). When combined with Boltzmann Softmax, it took Q-Learning 31 iterations to reach that goal.

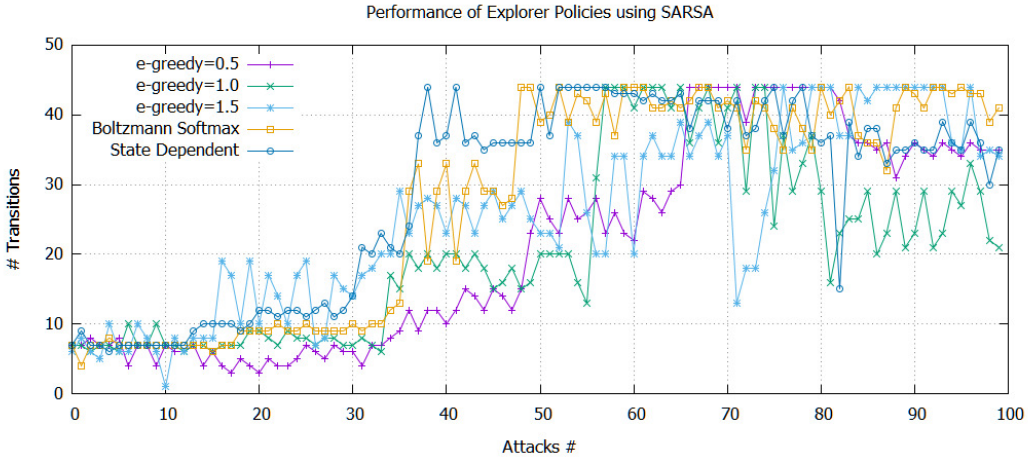


FIGURE 6.1: Performance of Explorer Policies using SARSA

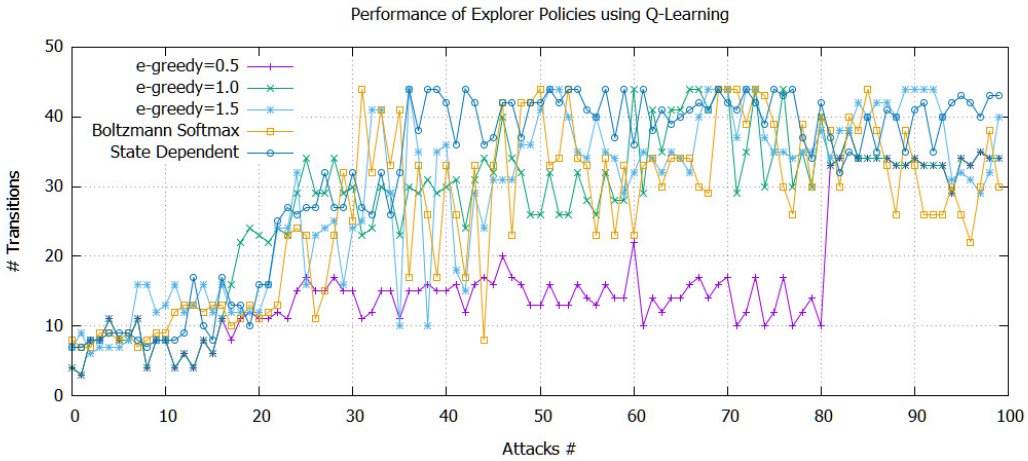


FIGURE 6.2: Performance of Explorer Policies using Q-Learning

Two further controlled experiments were performed with the dataset. The best performing combinations, namely *SARSA/State Dependent* and *Q-Learning/Boltzmann Softmax* were configured on separate honeypots. These combinations performed best against one particular malware variant. The entire dataset was streamed through

both combinations and the cumulative number of transitions were collated. From fig. 6.3, it can be seen that the SARSA/State Dependent combination performed approximately 6% better than the Q-Learning/Boltzmann Softmax combination. This experiment reinforces the requirement for agility in honeypot deployment. Although Q-Learning/Boltzmann Softmax realised the Mirai variant attack sequence faster (31 v 38 attack iterations), SARSA/State Dependent accumulated more transitions on the entire dataset, beginning at attack 59 (fig. 6.3). Continuous monitoring and analysis of honeypot datasets can inform new combinations for redeployment.

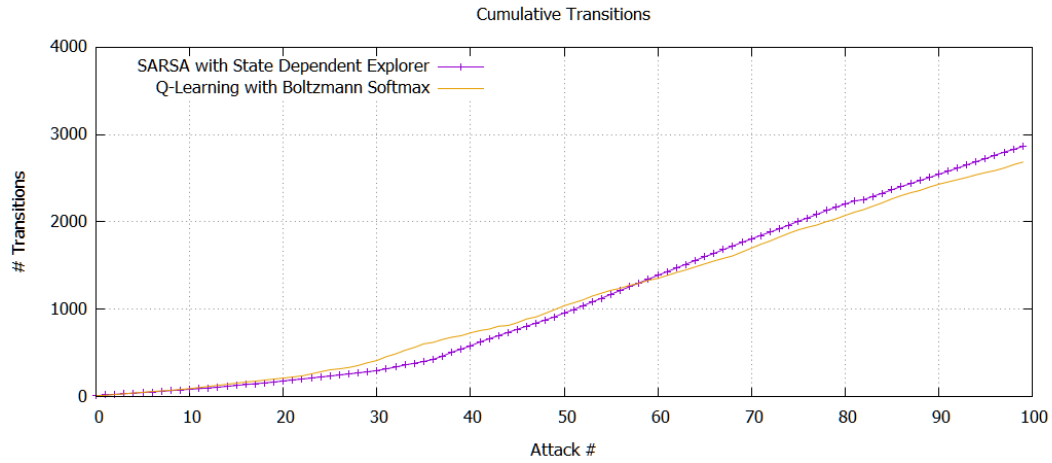


FIGURE 6.3: Comparison of SARSA/State Dependent with Q-Learning/Boltzmann Softmax

Other **Parameters** are used within the SARSA and Q-Learning algorithms. The algorithms' equations are reproduced here as eq. 6.1 and eq. 6.2 respectively:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{6.1}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{6.2}$$

Both SARSA and Q-Learning have α and γ parameters. The learning rate parameter α determines the extent to which new information is merged with older information. A "0" value ensures that the agent will not learn from new information whereas values nearer 1 will make the agent consider only the most recent information. Similarly γ is the measure of importance placed on future rewards. A "0" value makes the agent only consider the current rewards whereas values nearer "1" will make the agent consider a long term high reward.

These parameters are modified within the PyBrain code. By default α = 0.5 and γ = 0.98999999999999999. Both were modified for SARSA and Q-Learning for the assessment process. The results were interesting yet inconclusive. For example modifying γ made no discernible difference to the performances in fig. 6.3. Reducing α increased the Q-Values but not the cumulative transitions. When α was less than 0.1 and γ was less than 0.5 the accumulation of transitions flat-lined. The erratic

behaviour of the parameters requires more research. With no pattern to the results it is unknown if this is normal, abnormal, an issue with the action daemon or HARM in the controlled environment. Modifying α and γ does impact the functionality of HARM but further research is required to ascertain the extent and relevancy of this impact.

6.2 Redeployment

The cyclic process of the hypotheses can be considered to have the following steps:

1. Development (Hypothesis 1)
2. Monitoring (Hypothesis 2)
3. Optimisation (Hypothesis 3)
4. Redeployment

Honeypots need to continuously improved to be relevant for evolving cyber threats. Therefore *redeployment* is another step in this process leading to step 2 and so on. It can be step 4 or step 1. This cyclic process and subsequent framework (chapter 5) evolved after initial deployment in December 2017. The evolution required analysis of the captured dataset to determine results. This informed the development of articles for publications and conference proceedings ([137] and [143] respectively). Critical feedback from these directly impacted the design of the framework. Step 4 was therefore unavoidably delayed. There are many variables involved in redeployment that can impact HARM's performance. Some variables are controllable, some are uncontrollable and some are unknown.

The optimised configurations for HARM were *controllable*. Exact variables for SSH access and standard Kippo functionality were identical. The optimisation process was completed in a controlled Eclipse development environment. The resultant code modifications for learning algorithms (SARSA) and explorer policies (State Dependent) were implemented on AWS for HARM's redeployment.

The biggest *uncontrollable* variable was the new malware encountered on the redeployed HARM. This was due to the time duration between initial deployment (step 1) and redeployment (step 4), a period of approximately 15 months. The Mirai malware was still present in the logs but was not dominant. Other as yet unidentified malware significantly impacted the learning process. This may not be an issue if timely redeployments are involved.

The redeployment of HARM on AWS required updated installations. New versions of libraries were available due to the time delay. For example, new forks of Kippo solving bugs; new versions of Python and Twisted (event-driven network emulating framework). These updates should not affect HARM's functionality but their impact is *unknown*.

A further question for redeployment in the context of this thesis, is what purpose does it serve? Hypothesis 3 clearly demonstrated that an optimisation process impacts on the realisation of attack sequences and the resultant dataset. Even if the 15 month time lapse did not exist, how would a live redeployment be evaluated? What metrics indicate optimisation is functioning? The optimisation process in the controlled environment demonstrates improved functionality and (avoiding delay) this should translate to a live redeployment. The 15 month interval actually serves to highlight the need for a cyclic process. A *static* adaptive honeypot will quickly cease to be relevant. New malware trends and variants require continual HARM optimisation. With evolving malware, standard honeypots become irrelevant. Adaptive honeypots can become irrelevant. Adaptive and agile honeypots stay relevant.

6.3 Discussion

This hypothesis took PyBrain as the reinforcement learning standard and modified algorithms, explorer policies and parameters. In doing so it demonstrated that these modifications impact the performance of HARM in the controlled environment. The goal of this hypothesis is to demonstrate that reinforcement learning as a machine learning option can be optimised for current malware trends and methods. A default implementation of reinforcement learning is not in itself, a perfect fit for provisioning adaptive functionality in honeypots. There is a multitude of machine learning methods that could prove to be more effective against evolving malware. This is discussed further in chapter 8, *Potential for future work* Exploring these can be standardised by creating an *action daemon* that forms part of an overall framework for adaptive and agile honeypots. The action daemon is an important element of this optimisation process. Without this then figs. 6.1 and 6.2 would have realised the entire sequence from attack 1 through to 100. An interesting development to the daemon is a collaboration with researchers at Thales Group[1] in Quebec. Because HARM is freely available researchers have independently created and shared their equivalent *learning script* which process 3 lines at a time on a Docker container. The action daemon has the advantage of replicating the actions of HARM during the live deployment. This work is ongoing and has the potential of expediting and increasing HARM deployments.

The hypothesis also highlighted other potential directions for improvement. The performance of the parameters α and γ was erratic and it is unknown if this is abnormal or an issue with the assessment environment. Critically, a standardised and stable environment such as Eclipse is required to facilitate development and optimisation. Other platforms exist such as Microsoft Visual Studio, PyCharm and LiClipse. These were all installed during initial development of HARM but proved problematic due to library and dependency issues. Other Python platforms and machine learning libraries could be explored.

[1] https://www.thalesgroup.com/en/americas/canada

Thereafter the next logical step is to automate this process. Instead of a) manually extracting attack stream, b) modifying the action daemon and reinforcement learning values accordingly and c) streaming the extracted stream through the action daemon in a controlled environment, this could be a continual automated process. The three processes could run in parallel to a live deployment continually assessing HARM's performance against evolving malware.

6.4 Hypotheses Summary

To conclude this chapter, this section reflects on the 3 hypotheses as a whole. This chapter completes the cyclic nature of the processes espoused by these hypotheses.

AWS facilitated live deployments of both HARM and a comparison standard honeypot. Unintentionally, the trending malware captured on both was Mirai. HARM demonstrated improved learning over previous research [137]. Examining the code interactions found that HARM also overcame honeypot detection techniques [143]. The Mirai attack sequence was realised after 19 days (17^{th} Dec 2017). This compared with the simultaneously deployed honeypot only ever realising 8 commands in the attack sequence. The captured dataset from HARM also proved a valuable resource for the *review of HARM's performance*. Another benefit of using PyBrain is the ability to quickly modify learning algorithms and explorer policies. In a controlled environment the dataset coupled with PyBrain's flexibility was used to assess performance. When taken in totality, the hypotheses described in chapters 4, 5 and 6 form a cyclic process, as seen in fig. 1.2. Therefore hypothesis 1 could be considered the first process or another process in the continuous improvement of honeypot technology. These hypotheses together form the basis of a framework for adaptive and agile honeypots that aims to improve overall cyber forensics (see chapter 7).

The trending malware experienced on the honeypots, namely Mirai, was the dominant attack for the deployment duration. This provided a great opportunity to concentrate the optimisation experiments on one variant amidst the noise of other malware. But it also imposes limitations. The results produced in hypothesis 3 were impacted by the Gaussian noise of other malware events captured on the honeypots. It is conjecture to generalise for all malware by concentrating on one variant. To support this conjecture, malware methods and variants mutate, therefore honeypot deployments will experience the current trending dominant malware. It should also be stated that the honeypots were deployed on only one attack vector (SSH). SSH and Telnet are targeted protocols and are responsible for over 93% of all login attempts on honeypots deployed by the Honeynet Project [134]. Further research is required to ascertain if adaptive and agile honeypots can have a similar impact, with malware variants designed for other attack vectors.

However the cyclic process presented by the hypotheses chapters using SSH as an attack vector, does demonstrate improved learning, data capture and faster attack sequence realisation. The cyclic process requires monitoring, data extraction and intervention in the form of experiments and code modification. It is entirely reasonable to contemplate this being totally automated. An environment of development -> deployment -> optimisation -> redeployment, running in parallel with a live and continually improving HARM, could implement the proposed framework without any intervention.

Chapter 7

Framework for Adaptive and Agile Honeypots

The framework presented in this chapter is a culmination of the 3 hypotheses that are empirically proven in chapters 4, 5 and 6. This final contribution to the state of the art is also presented in journal publication:

- *A Framework for Adaptive and Agile Honeypots*, Seamus Dowling, Michael Schukat, and Enda Barrett, 2019, In **ETRI Journal: Special issue on Cyber Security and AI** https://onlinelibrary.wiley.com/journal/22337326, (under review)

Taxonomies and frameworks for honeypot related technologies have been cited in this thesis. These are specific to areas such as honeynets, SDN and IoT to name a few. The problem statement (section 1.2) maintains that Seifert's taxonomy governing honeypot development could be considered out-dated in the face of evolving technology and subsequent malware threats. The taxonomy predates advances in technology which create new attack vectors. It also predates the use of machine learning techniques for proactive functionality. This has been detailed in section 2.4. The research questions and hypotheses in this thesis set out *adaptability* and *agility* as core outcomes. The results from the practical experimentation for the hypotheses bear this out. For honeypots to be relevant as tools in cybersecurity, they themselves need to mitigate against inadvertent termination, detection and repetitive truncated datasets. It is incumbent on honeypot developers and operators to implement alternative measures. Measures that can provide the honeypot with the adaptability to learn from the attack interactions and the agility to be deployed, optimised and redeployed expeditiously. The existing taxonomy in table 2.4 requires revisiting to consider the relevancy of its classes and actions. The automated and repetitive characteristics of malware impact on this relevancy and therefore the classes and values need to be examined for evolving malware threats. Whilst some elements of this taxonomy still hold, updated classes and values should be considered for a new framework for adaptive, agile honeypots.

7.1 Taxonomy Revisited

Each of the classes in Seifert's taxonomy are revisited apropos evolving malware and their methods. Attack vectors are increasing with new technological concepts. Older technology creates redundant values. New values arising from the hypotheses presented in chapter 3 require consideration.

- *Interaction*: Low interaction honeypots are still available. They simulate Internet services and capture basic interactions. Medium interaction honeypots capture more information. To contribute to meaningful cyber security research, MIHPs or HiHPs should be deployed. An issue with MIHPs and HiHPs is the possibility of the underlying honeypot architecture being compromised and participating in further attacks. This has been mitigated with the use of virtualisation to deploy honeypots and abstract them from the underlying architecture. Machine learning has been integrated into honeypot operations, interacting and learning in real time from attack code. This creates a new value of *adaptive* honeypots, for Interaction Level. MIHPs and HiHPs with adaptive abilities should be deployed to realise attack sequences faster.

- *Data Capture*: All automated interactions are captured by honeypots. The initial brute-force or dictionary attempts to gain access can be considered pre-compromise. They do not offer much analytical value and often skew results towards failed attempts. Initial set up of honeypots need to have a period of monitoring to ensure that the honeypot is functioning correctly regarding irrelevant data capture. Username/password combinations and ingress/egress port filtering can increase or reduce the quantity of honeypot activity. Post compromise data from MIHPs and HiHPs offers the best insights into attack design and behaviour. As demonstrated in section 4.3 (chapter 4), malware will use obfuscated executable code and downloaded files to compromise, communicate and propagate. Longitudinal deployments leads to repetitive data capture. A new value of *Time Limited* should be added to Data Capture class to improve cyber forensics when adaptive honeypots are deployed.

- *Containment*: The values for this class are still very relevant. Virtualisation has abstracted the honeypot from the underlying architecture and therefore has somewhat protected against ethical concerns regarding operations [56].

- *Distribution Appearance*: Post compromise, malware will scan the environment for other potential addresses and services. New networking paradigms such as IoT will have a very different distribution appearance and will encounter evolving propagation methods. Malware could evolve to exploit mesh networks or other non-traditional models. The adaptive functionality of honeypots can respond to prolong this interaction safely in a virtualised environment.

- *Communication Interface*: The Non-Network Interface is a redundant value. Malware uses the Internet's communication protocols and software application programming interfaces (API) to propagate. The physical network interface itself can be compromised at a physical level. Wifi and IoT wireless interfaces become attack vectors to gain access to devices [128].

- *Role in a multi tiered architecture*: Malware does not discriminate post compromise. If a vulnerable device is accessible on an attack vector, malware will launch complex code structures to compromise the underlying architecture. This is irrespective of whether the honeypot advertises client or server services. An adaptive honeypot will learn the best responses to realise all commands in an attack sequence. With IoT deployments gathering pace, reduced function devices (RFD) and full function devices (FFD) create complex mesh networks requiring communication and gateways to Internet services. Traditional client and/or server models need to be expanded to include *function*.

7.2 Proposed Framework

Analysis of the existing taxonomy in section 7.1 can be integrated into a new framework for honeypot development and deployment. Adaptive and agile honeypots can be operated using a cyclical process of 1) adaptive honeypot development, 2) time limited deployment and data capture, 3) honeypot optimisation. The model, shown in fig. 7.1 informs honeypot developers and operators, expedites the capture of datasets with complete attack data and ultimately leads to improved cyber forensics. The model can be separated into 2 distinct sections that correlate with Seifert's taxonomy classes:

- *Adaptive Honeypot*
  Hypothesis 1 details the creation of HARM, a honeypot that exploits malware automation and repetition using a unique state action space formalism. The relevant classes associated with HARM's development are *communications interface*, *distribution appearance*, *interaction level*, *role in a multi-tiered architecture* and *containment*. These classes require consideration prior to development.

  SSH and Telnet dominate attempted logins [134]. Non-network interfaces such as device peripherals (printer ports, USB, CD/DVD) are non-realistic attack vectors and is removed from *communications interface*.

  MiHP and HiHP compromise is negated through virtualisation and provide the best datasets for cyber forensics. The 'adaptive' functionality of HARM creates a new value for *interaction level*

  The exposure of IoT devices is demonstrated with the success of Mirai and its variants. A honeypot can adhere to the terminology of 'FFD' and 'RFD' in their *role in a multi-tiered architecture*.

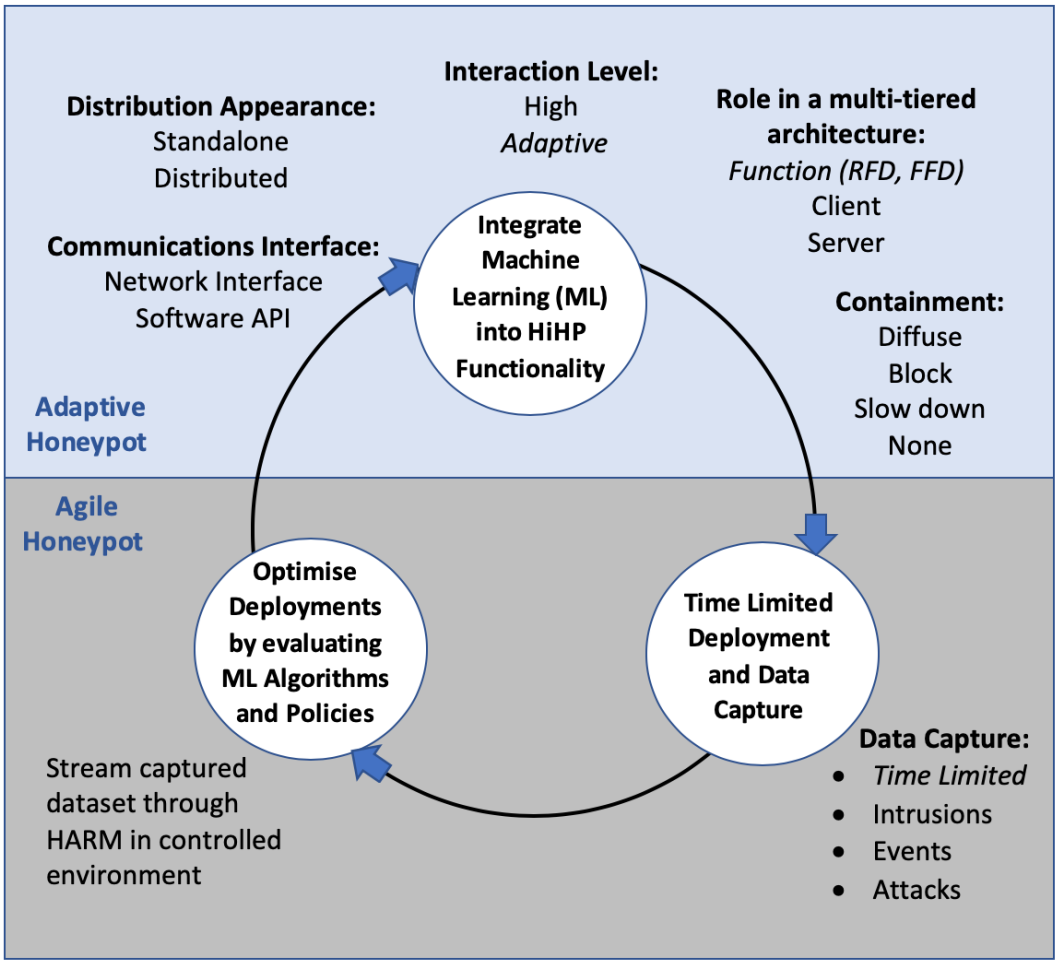


FIGURE 7.1: Framework for Adaptive, Agile Honeypot Development and Deployment

- *Agile Honeypot*
  Hypotheses 2 and 3 reside in this section of the framework. These influence the inclusion of class *data capture* and the addition of 'time limited' as a value. These hypotheses demonstrate that adaptive honeypots capture sufficiently relevant data for meaningful analysis, after a relatively short deployment period. A honeypot with adaptive functionality that does not evolve with new malware is in effect static. Hypothesis 2 realises attack sequences quicker leading to the limit imposed on data capture. It can quickly give insight into the adaptive honeypot configuration that provides the optimal environment for providing relevant data capture. An input stream from an immediate dataset can facilitate this evaluation. This process informs the reconfiguration (hypothesis 1) and redeployment (hypothesis 3) of the adaptive honeypot. Hypothesis 3 provides the agile element of the framework by evaluating learning algorithms and explorer policies.

7.3 Chapter Summary

This chapter is essentially a culmination of the thesis's hypotheses and the practical experiments used to test these hypotheses. There are 2 distinct stages to these experiments detailed as *HARM Implementation* in section 3.3.1, and *HARM Assessment* in 3.3.2. This chapter combines these elements into a framework model seen in fig. 7.1. This framework can be expanded to other honeypot applications targeting malware on different attack vectors. The underlying application software for these honeypots will determine the appropriate machine learning libraries or software. Similarly, differing algorithms may perform better against malware experienced on differing attack vectors. The time limited value will vary depending on the honeypot application and associated malware. It is impossible to put a value on this but it does inform honeypot operators to be cognisant of this characteristic of honeypot functionality. Operators will determine this time period value after a series of deployment, data capture and analysis. This value will change as new trends and variants appear. The agility of the framework can continually assesses this value and honeypot redevelopment. This research has a very narrow focus; reinforcement learning on a SSH attack vector. It stands to reason that other machine learning algorithms are better suited to other malware methods. The framework can not provide a correlation between them. But it can provide a road map as to how honeypot operators could establish the optimal machine learning variables (algorithms, explorers, policies, parameters etc) that will provide the environment to capture the most relevant dataset for cyber forensics.

Chapter 8

Conclusions

This thesis seeks to advance the SOTA for honeypot development and operations. During the period of research (6 years) malware encountered on various honeypot deployments evolved. Trending malware from botnets and DDoS tools to IoT variants waxed and waned in popularity. The honeypot technology itself also evolved which presented both opportunities and complications. The central tenet from the beginning is that a *rethink* is required regarding honeypots in the face of technology and malware evolution. Having multiple honeypots on multiple attack vectors catering for multiple malware types is too reactive to provide meaningful cyber forensics. Plus it is not scalable. Evolving technological concepts breed new malware methods requiring corresponding honeypot development. This rethink involves creating adaptive honeypots that learn from malware interactions. Facilitating this with machine learning is not new and has been proposed previously [79, 98]. What is new however is using machine learning to adapt and learn from automated and repetitive malware *and* using machine learning to periodically assess this adaptiveness vis-à-vis trending malware and evolving variants. This chapter summarises this thesis's work by reexamining the methodologies and analysing the subsequent results. It discuss these results in the context of the hypotheses and research questions from section 1.2.1. This discussion culminates in the proposed framework which is a deliverable of the central tenet, the rethink. This chapter also critiques the entire thesis identifying possible weaknesses and limitations. Finally it suggests solutions to these limitations forming the potential for future work. The following sections are therefore discussed:

- Methodologies and Results
- Research Questions, Hypotheses and Framework
- Critique and Limitations
- Potential for Future Work

Methodologies and Results The title of thesis thesis proposes a *new framework for adaptive and agile honeypots*. It can be considered that the Methodology chapter section 3.3.1, *HARM Implementation* equates to the adaptive element and section 3.3.2

HARM Assessment, equates to the agile element. Three diagrams are important to this discussion:

- Fig. 1.2 identifies the cyclic process of Development -> Deployment -> Optimisation and the corresponding hypotheses.
- Fig. 3.14 is a concept diagram representing the methodologies to realise the hypotheses.
- Fig. 7.1 is the framework incorporating the hypotheses and the adaptive (Implementation) and agile (Assessment) elements.

The concept diagram is central as it is the practical implementation that realises the hypotheses and conceptualises the framework. HARM Implementation is informed by previous research and uses Heliza [79] and RASSH [98] as initial benchmarks for reinforcement learning and the underlying software platform. Heliza (2011) and RASSH (2014) both provisioned human interaction, rewarding the learning algorithm when insults and/or artificial delays invoked further attacker responses. Their reward function is obtained by calculating the Levenstein distance between the attack command and a predetermined set of commands. Exactly how much human interaction was encountered or how it affected the learning process is not known. The article presents the longest and average attack durations. It also states that insults were issued to 86 attackers but does not correlate this with human activity. What is known however is the amount of human interaction on an IoT honeypot deployed more recently [10]. That honeypot collected over 9 million lines of attack interactions including 423228 login attempts, 413362 unsuccessful logins and 9866 successful logins. Only 4 of the successful logins demonstrated obvious human cognition. 99.96% of the activity was repetitive malware types using automated methods, identified as having specific binaries bit sizes stored as TTY files. Contact was made with the Kippo honeypot developer on the Github forum, with concerns about such a high level of automated traffic. The response was "*i didn't see human on my kippo in a very long time*" and "*your honeypot is functioning fine*". There is a requirement for human communication from botmaster to C&C but almost none is encountered on end devices that are compromised.

The first hypothesis (chapter 4) queried whether a simplified reward function (eq. 3.7) would demonstrate improved learning when compared to the aforementioned Heliza and RASSH. And it did. But what was actually happening to the attack commands at discrete points in the convergence towards the optimal policy? The Github forum again acted as a catalyst for further investigation. Ironically the IoT honeypot deployment predated the widespread Mirai bot that compromised vulnerable IoT devices in August 2016, accessible through SSH and running a *BusyBox* shell. It subsequently spawned numerous variant such was its success. Threat advisories, security groups, white papers and publications all dissected Mirai and offered protection advice. Honeypots played their part in this process [3] and the

Cowrie/Kippo developer forum also questioned the capture of the variants [121]. *"You have failed one of the trojan's anti-honeypot checks"* and *"I bet trojan detected cowrie by process list or mounts list"* were some of the comments offering solutions to a lack of data capture.

A Mirai variant was the dominant repetitive malware captured on HARM as it was *trending*. The comparison Kippo honeypot simultaneously deployed failed after 8 command interactions. Together with the forum posting, it suggests that the variant incorporated honeypot detecting methods or determined that the end devices were virtual or non-viable as a potential compromise. HARM selected different actions that rewarded prolonging interactions therefore it established the best actions to overcome these methods. Bash script extracted this Mirai variant interactions and discovered that the it had 44 commands in the attack sequence (table 4.1). The evolution and eventual realisation of the entire 44 commands in an attack sequence could be established. This realisation occured after 17 days, asking the question *"How big is enough?"*. How much information needs to be captured? Or how long does a honeypot need to be deployed for before the captured dataset is adequate for analysis?

HARM Assessment (3.3.2) answers these questions and creates the agile component of the framework. It is the optimisation stage of the cyclic process espoused by the hypotheses. If the adaptive HARM realises an entire attack sequence after a period then what is the purpose of maintaining that deployment, capturing repetitive information? It could be argued that honeypots are traditionally deployed for longitudinal analysis. But this discussion is about *adaptive* honeypots designed to proactively engage with an attack sequence with a goal of prolonging interaction. At some point in the adaptive process, the honeypot will have interacted long enough to retrieve all relevant information. Thereafter the adaptive honeypot is repeating the capture. Trending malware popularity may wax and wane. Therefore the reinforcement learning variables (algorithms, explorers, policies, parameters) should be modified so as to expedite the convergence towards optimal policy for these trends. This is done quickly in an offline controlled environment as detailed in hypothesis 3 (chapter 6). A dataset can be analysed, parsed and reconstructed to act as an input stream into this offline HARM environment.

The configuration and operations of this environment is outlined in appendix B.3. It uses Eclipse as the application but other Python development environments could be used. The input stream into this environment needs to be assessed on a case by case basis. Analysis of the captured dataset will produce the attack sequences (appendix C.3). Threat advisories and other forums will usually provide the binaries and executable code associated with the the attack stream. This code is often obfuscated. For example, *echo -e ' \x47 \x72 \x6f \x70/' > //.nippon* in table 4.1 has been obfuscated with \x handles and decodes from hex to ASCII to create executable code. This code is executed, > *//.nippon* creating a daemon to communicate with the C&C. The assessment process needs to replicate the actions of the live interactions as closely as possible. Simply streaming the extracted attack sequences

will result in all commands being processed without any intelligence. The action daemon replicates HARM's behaviour as best as possible by analysing the actions taken in a live environment. Standardised decision making by the daemon for all subsequent assessment experiments ensures integrity across the results.

Research Questions, Hypotheses and Framework Subsection 1.2.1 opens by stating "To contribute meaningfully to the rapidly changing world of cyber security and forensics, this thesis proposes a number of enhancements which advance the effectiveness of honeypot operations." The raison d'être of honeypots is capturing data for forensics. Therefore any proposed enhancements have to involve the improvement of this data capture. So how can these enhancements be realised? Honeypots as a cyber defence measure need to be relevant. Relevant in this context is a meaningful contribution to cyber forensics by producing a dataset that gives better insight into malware methods. If a honeypot produces a vast dataset of truncated, repetitive attack sequences after months of deployment then this has very little relevance to cyber forensics. The research questions query if honeypot relevancy can be improved. The questions can be considered to be iterative with one building upon the previous. To paraphrase the query as a statement:

> "If adaptive honeypots overcome inadvertent termination of attack sequences by using reinforcement learning designed for automation and repetition, then they will stay relevant as cyber forensic tools if their reinforcement learning component is continually evaluated for trending and evolving malware."

This statement focuses this research towards keeping honeypots relevant as technology and malware evolves apace. The continual evaluation of honeypot functionality creates a cyclic process of design -> deploy -> optimise. The 3 hypotheses proposed to explore the research questions align with this cyclic process as presented in fig. 1.2. The above statement paraphrasing the research questions align with each numbered hypothesis:

1. "...reinforcement learning designed for automation and repetition..."
2. "...overcome inadvertent termination..."
3. "...continually evaluate..."

Fig. 1.2 also lists the practical elements to explore each hypothesis. Chapters 4, 5 and 6 results goes into detail for both the practicals and hypotheses and have been already discussed in *Methodologies and Results* in this chapter. Each research question, hypothesis and practical builds upon the previous ensuring that the underlying honeypot is continually capturing the optimum dataset for meaningful malware analysis. In doing so the honeypots have achieved their goal and will stay relevant as cyber forensics tools.

As honeypots evolved, new taxonomies and frameworks have appeared. As early as 1993 the term *crackers* was coined by Bellovin [1]. A decade passed until early malware became established followed by early honeypots. Zhang [75] initially created a taxonomy in 2003, Seifert [11] updating this in 2006. Honeypot architectures appeared for new technology such as honeynets [80], SDN [74] and IoT [81].

This chapter began by stating a rethink is required for honeypots in an evolving technological world. The methodologies and hypotheses culminate in the proposed framework for adaptive and agile honeypots. It takes the still relevant classes from an existing taxonomy, updates and incorporates them into the new framework. Seifert's taxonomy's classes are still required for design, deployment and capture of honeypots. Implementation (3.3.1) and assessment (3.3.2) are overlaid on the cyclic hypotheses' processes ensuring a robustness to development and deployment. The resultant framework fig. 7.1 has established honeypot standards incorporated into adaptive and agile methods, which re-imagines honeypot functionality and ensures their relevancy as cyber forensic tools.

Critique and limitations The proposed framework evolved over the lifetime of this research. Honeypots were designed and deployed, the data captured and analysed informing the development of new honeypots. Initially in 2013 research concentrated on IoT aware and architecture specific malware. Honeypots were deployed on bare metal machines and Raspberry Pi with banners such as *ZigBee-Gateway* and *IoT-Hub* to pique interest. Log checking on an hourly and daily basis soon waned as it became obvious that the same automated malware was probing and compromising the devices. The honeypot distribution was queried and logs posted to the developer forum, to determine what was wrong with the honeypot deployment. The response from the developer forum was *"your honeypot is working fine"*.

Automation and repetition were the common characteristics to all the datasets captured on a variety of initial honeypots. The only change to this over the intervening years is the malware types encountered. Ironically this is problematic when designing the methodologies. The malware trends and types from the initial IoT honeypot (section 3.2) informed the implementation of HARM. The IoT honeypot was deployed during December 2015 till February 2016. It took approximately two years to analysis the captured dataset, design HARM and deploy it in December 2017. The multiple dominant malware types in 2015/2016 were all but absent in 2017 and HARM was compromised by one dominant malware type, the Mirai variant. Similarly the assessment of HARM was initially tested in a controlled environment. An attempt to validate the assessment process by deploying an optimised HARM on AWS produced completely difference results in August 2018. The intervening eight months saw new malware types and trends whereas HARM had been optimised using an input stream from and older dataset. This was a result of the evolutionary nature of the research itself. This could be ameliorated with a quick

turnaround of the assessment process. This has been previously discussed in section 6.2, *Redeployment*

The initial setup of the honeypot impacts the amount of data captured. Variables can be configured such as username/password credentials for access or ingress/egress protocols allowing for further malware operations. These affect the quality and quantity of attack information. Invariably multiple attack types will be collected, some of which may be infrequent malware, trending malware or the dominant capture on the honeypot (see fig. 3.4) . Adaptive functionality also impacts the data capture. The machine learning algorithms, explorers, policies and parameters will perform differently against malware variations. Analysis of the dataset is also open to interpretation as to what constitutes the input stream into the *assessment* process of HARM. Bash script from fig. C.3 for example, could include some 'noisy' data from all malware encountered on the honeypot or just a particular dominant malware type. Analysis of the data may or may not identify a dominant type.

The consequence of these variables impacts the performance of HARM Implementation and HARM Assessment. To refine the process, this thesis used the Mirai bot variant (table 4.1) as input into the assessment phase. During the live deployment however, the learning evolution of HARM was impacted by other malware encountered and affected the realisation of the entire attack stream (fig. 5.5). This research may suggest a generalisation for all malware based on one malware type. But this is how HARM was configured and this is what it captured. It has been stated that timely deployments could ameliorate malware diversity; a quick redeployment after the assessment phase will encounter the same malware. A similar conclusion can be drawn here regarding honeypot variables and the interpretation of the dataset. However the honeypot is initially configured or the data stream is extracted, these variables must become standardised across all deployments. If HARM reinforcement learning combinations are subjected to a common standard then the integrity of the implementation and assessment phases is maintained.

The Eclipse development environment was chosen after installing and attempting to implement HARM on other platforms. Microsoft Visual Studio, PyCharm, LiClipse were just some of the failed attempts at getting a functioning PyDev environment that could run HARMs elements such as Kippo, PyBrain, Twisted, Python 2.7 and other libraries and dependencies. This was essential for the development of HARM but especially for the *Assessment* phase. Other platforms could be used as long as a standardised approach for all processes in the framework is adhered to.

Table 2.2 presents a sample of protocols used by IoT devices for communication. Some of these involve transmitting data between IoT nodes, synchronising devices in a topology or accessing the device remotely. SSH has been a ubiquitous network protocol for more than 20 years. It uses PKI to provide secure communications across an unsecured network. An SSH client connects to an SSH server running on a remote

device and logs in with username/password credentials. A preponderance of intermediate Internet and end user devices, particularly IoT, provide remote access using SSH over port 22. Telnet provides the same service over port 23 with no authentication or encryption.

An issue with many SSH accessible devices is the lack of intervention or invention regarding login credentials. SSH and Telnet accounted for 93% of all login attempts on research by the *Honeynet Project* [134]. SMTP, POP3, HTTP, HTTPS, FTP and POP3S were responsible for the remaining 7%. Kippo over SSH was initially chosen for all HARM methodologies so as to compare this thesis's research to previous research. The results associated with these methodologies have demonstrable merit and culminated in the proposal for a new framework. This research has concentrated on using SSH as the attack vector. It is a limitation of this work that only one attack vector is used. It could potentially translate to the other 7% of protocols, IoT protocols from table 2.2 and other potential attack vectors associated with new and emerging technology.

There is a comprehensive collection of honeypots designed to target malware on most Internet protocols and services [70]. It has been highlighted in section 2.5 that as vulnerabilities are discovered, malware is developed to exploit these vulnerabilities and honeypots are deployed to capture the malware. The methods used by malware differs depending on the protocol or services being targeted [47]. Malware tools can target vulnerabilities of base architecture, communication channels, operating systems, services and applications. The tools can consist of complex coding structures in a variety of programming languages. For example, Mirai used C and Go programming to examine the accessible host, download and execute architecture specific tools, compromise the host and hide its existence. Honeypots are then designed to simulate or emulate this interaction. It requires a complex matrix of honeypot development and operations to facilitate the required interactions for all attack vectors and known vulnerabilities. Such a matrix could systematically apply HARMs methodology and the proposed framework to honeypot development and operations. Two possible candidates for such a matrix are:

- Lockheed Martin's *Cyber Kill Chain* [144]
- Mitre's *ATT&CK Matrix* [145]

These models are freely available and developed to help organisations mitigate against malware operations. These resources detail the tactics, techniques and procedures that malware uses to expose and compromise system vulnerabilities. The Cyber Kill Chain is a model that identifies the steps taken by malware such as advanced persistent threats (APT) to penetrate a system and execute its objective. The ATT&CK matrix (Adversarial Tactics Techniques and Common Knowledge) is a structured list of known attacker behaviour expressed in a matrix of tactics and techniques. The Cyber Kill Chain model sees attacker behaviour move linearly to achieve the objective. The ATT&CK matrix facilitate non linear movement of malware behaviours. As an

example, malware tactics could exfiltrate data or move laterally with the network. Pertaining to HARM's future work, these matrices facilitate the systematic roll out of adaptive and agile honeypots on multiple attack vectors, for known vulnerabilities. Each matrix gives comprehensive responses to each attack incident. Adopting a comprehensive matrix can ensure that HARM deployments cover all known malware attack vectors and methods. The adaptive nature of HARM ensures that malware variants continue to have full interaction on honeypot deployments. The agile nature of HARM's development and deployment ensures honeypot technology stays relevant as a cybersecurity tool.

Potential for future work Malware will always create the potential for future work. This thesis explored evolving technology, malware and honeypots narrowing its focus to reinforcement learning on an SSH attack vector. Reinforcement learning was chosen because of its proven efficacy in solving MDPs and also to compare with previous research. New and existing machine learning methods are an obvious choice for expanding this research. With such a diverse array of malware, the framework could be further refined towards malware groupings such as bot traffic, DDoS tools, rootkits, ransomware etc. One potential enhancement to the framework is the isolation of attack types when encountered on HARM. If an adaptive honeypot is able to identify an attack type from the initial command sequence then it initiates or continues with a unique machine learning process for that type. In this way, the adaptive honeypot could maintain multiple learning processes, each one specific to a malware type. It would separate the Gaussian noise associated with other types from each other, which impact convergence towards optimal policy. Machine learning has already been discussed in relation to retrospective analysis in subsection 2.4.2. A comprehensive list of machine learning techniques pertaining specifically to honeypot data is visible in table 2.6. Progressing retrospective analysis further could see HARM use machine learning for secondary learning on the captured dataset. Patterns in attacker behaviour such as command sequencing, responses to HARMs' actions, decision making, temporal and spatial analysis could be discerned from the data. Supervised and unsupervised machine learning methods could be used to develop training processes for specific malware types. Classifiers are trained using data from HARM and then employed, depending on the real time attack command sequences encountered during live deployments. Automated and repetitive malware facilitates this sequential learning. For example, sequence mining could identify recurring patterns in repetitive malware allowing for accurate prediction of the next command in an attack sequence. Once trained, prediction models can predict the expected next command during a live deployment. The absence of an expected predicted command, signals a shift in attacker behaviour. A recurring shift in this behaviour identifies potential new variants or the discovery of a zero day attack.

The next logical step is to automate the entire process of *a)* manually extracting attack stream, *b)* modifying the action daemon and reinforcement learning values

accordingly and *c)* streaming the extracted stream through the action daemon in a controlled environment. The framework itself evolved from multiple deployments, redeployments and redevelopments. The datasets captured informed publications and acted as next steps to the formulation of the framework. Automating the continual improvement of HARM would require tangential or concurrent processes performing steps *a)*, *b)* and *c)*. This in turn could automatically modify HARMs reinforcement learning values optimised towards current malware trends and methods.

Another obvious option for future work is expanding the attack vector beyond SSH. This has been alluded to in the previous paragraph. Well established honeypot technology operates within these attack vectors. The adaptive functionality of these honeypots would be need to be explored regarding compatibility due to software dependencies. The implementation and assessment stages of the methodology should still adhere to the adaptive and agile principles espoused by the framework. Machine learning algorithms can be explored in relation to malware types experienced on these honeypots. Software libraries facilitating machine learning may need to be written or may exist depending on the honeypot platform. It would provide for an exciting experiment to see if the framework would result in improved data capture for cyber forensics, on other attack vectors and honeypot technologies.

The flexibility of AWS allows HARM to be deployed effortlessly in different global regions. This provides for the potential to explore regional variances whereby malware targets IP ranges or localised vulnerabilities. The images in section 2.2.1 demonstrate temporal variances which indicate the measure of security and compromised hosts in regions roughly equating to the Americas, Europe/Africa and Asia. Malware has been shown to have localised targets [49]. This may be due to global IP allocations or regional bias. HARM deployed at a particular AWS location may perform completely differently at another location. Therefore it is not sufficient to optimise HARM and deploy them at various AWS locations. Each deployment irrespective of location will require a quick independent assessment process prior to redeployment. A recent development related to this point involves collaboration with researchers in Quebec replicating HARM on Docker containers. As HARM is freely available, this could further expand both the quantity and distribution of HARM and also continue its development as a relevant tool against cyber attacks.

> **"A perfect scenario would have geographically dispersed HARM deployments on a variety of honeypot technologies, running diverse machine learning algorithms on separate processes, being automatically assessed, optimised and redeployed in a timely manner. This could ensure the ongoing relevancy of honeypot technology as a cyber forensics tool."**

Appendix A

Repositories

These repositories can be used in conjunction with appendices B and C to install HARM, view the data capture format, manipulate the dataset, reproduce images or produce new datasets.

A.1 Link to IoT HPot *dataset* on Github

https://github.com/sosdow/RepetitiveDataset
This dataset demonstrates repetitive attacks producing figures in section 3.2

A.2 Link to Adaptive HARM *code* on Github

https://github.com/sosdow/RLHPot
HARM can be installed using these files and instructions in appendix B.

A.3 Link to Adaptive HARM *dataset* on Github

https://github.com/sosdow/HARM-Logs
These log files can be viewed in a terminal window or with any text editor. Example of dataset analysis is shown in appendix C.

Appendix B

Installations

The deployment of HARM on AWS EC2 instances and Eclipse development environment requires various installations of resources, libraries and environments. For example, Kippo, Python, Twisted (event-driven network emulating framework) and MySQL are all required for successful compilation. To ensure consistency in reproducing HARM's functionality the following appendices should act a a guide. Due to new versions of required resources and BASH installations methods, some code segments will require updates and upgrades. Modify as required.

B.1 Install HARM on Eclipse and Amazon AWS EC2 Instance

There are lots of sources for step-by-step installation guides for the Kippo honeypot distribution. Following any of these guides will install a standard Kippo instance on a physical or virtual machine. Python is installed as part of these guides. Thereafter the installation requires customisation to set up the environment for HARM's development and deployment.
After installing the standard Kippo distribution, replace the file contents with the HARM files from the repository in appendix A.2. These files require further libraries and configurations. The Bash scripts in fig. B.1 are used to make these installations and configurations.

```
502  curl http://bootstrap.pypa.io/get-pip.py > get-pip.py
503  python get-pip.py
504  pip install crypto
505  pip install pycrypto
506  pip install twisted
507  pip install pygeoip
508  pip install mysql
509  pip install mysql-python
510  git clone git://github.com/pybrain/pybrain.git
511  pip install mysql-connector-python-rf
512  touch .bash_profile
513  nano .bash_profile
```

FIGURE B.1: Bash Installation Scripts

After installing *pip* (lines 502 and 503), use pip to install *crypto*, *pycrypto*, *twisted*, *pygeoip*, *mysql* and *mysql-python* (lines 504 - 509). Some of these may already have

been installed during the standard Kippo installation. A bug exists in the *crypto* installation requiring it to be renamed to *Crypto* in the Python/site-packages folder (*mv crypto Crypto*).
Change the MySQL root password (use MySQL tutorials online). Install the mysql-python connector (line 511) and test it using the python code in fig. B.2.

```
Seamuss-MacBook-Pro:~ sosdow$ python
Python 2.7.10 (default, Feb 22 2019, 21:55:15)
[GCC 4.2.1 Compatible Apple LLVM 10.0.1 (clang-1001.0.37.14)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import mysql.connector as mysql
>>> conn = mysql.connect(user='root', password = '[redacted]', host = '127.0.0.1')
>>> print(conn)
<mysql.connector.connection.MySQLConnection object at 0x1098d3e10>
>>> conn.close
<bound method MySQLConnection.close of <mysql.connector.connection.MySQLConnection object at 0x1098d3e10>>
>>> exit()
```

FIGURE B.2: Testing MySQL Connection in Python

Make sure MySQL can run from any location by including the following in the *.bash_profile* file (lines 512 and 513): *export PATH=$PATH:/usr/local/mysql/bin*.

The necessary *.sql* files required to create the MySQL tables and populate the fields are available in the **/doc/sql** folder of HARM's GitHub repository in appendix A.2.
Once these installations and configurations have been made, HARM is ready to be deployed.

B.2 Deploy HARM on Eclipse and AWS EC2 Instance

Prior to deploying HARM on either Eclipse or AWS EC2, two actions need to be performed. These can be completed at the terminal window:

1. *PYTHONPATH* needs to be set. It is an environment variable which indicates additional directories where python will look for PyBrain modules and packages.

2. *IPTABLES* need to be amended to redirect all SSH port 22 traffic to the honeypot which is configured on port 2222. To do this, iptables need to be stopped and restarted.

These two actions are displayed in fig. B.3. Thereafter HARM can be launched using *./start.sh*. To test connectivity, connect to the honeypot using a terminal window and the the public AWS IP address (displayed in the figure):
- ssh root@34.255.198.165 -p 2222

```
Seamuss-MacBook-Pro:Downloads sosdow$ ssh -i "20Oct17kippo.pem" ubuntu@ec2-34-255-198-165.eu-west-1.compute.amazonaws.com
The authenticity of host 'ec2-34-255-198-165.eu-west-1.compute.amazonaws.com (34.255.198.165)' can't be established.
ECDSA key fingerprint is SHA256:K4WDdnFZF84LZO9tmGA8JBwS/vPzPEnbLq/KjrB6MVw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-34-255-198-165.eu-west-1.compute.amazonaws.com,34.255.198.165' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.13.0-149-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

  System information as of Fri Jul  5 13:48:56 UTC 2019

  System load: 0.0               Memory usage: 6%   Processes:       83
  Usage of /:  17.8% of 7.74GB   Swap usage:   0%   Users logged in: 0

  Graph this data and manage this system at:
    https://landscape.canonical.com/

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

114 packages can be updated.
86 updates are security updates.


Last login: Tue Jul  2 10:10:04 2019 from 95-44-237-215-dynamic.agg2.wst.rsl-rtd.eircom.net
ubuntu@ip-172-31-35-172:/June20/home$ export PYTHONPATH=/June20/home/rlharm/pybrain:$PYTHONPATH
ubuntu@ip-172-31-35-172:/June20/home$ sudo service ssh stop
ssh stop/waiting
ubuntu@ip-172-31-35-172:/June20/home$ sudo iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 22 -j REDIRECT --to-port 2222
ubuntu@ip-172-31-35-172:/June20/home$ sudo service ssh start
ssh start/running, process 1456
ubuntu@ip-172-31-35-172:/June20/home/rlharm$ ./start.sh
Starting rassh in background...Loading dblog engine: mysql
ubuntu@ip-172-31-35-172:/June20/home/rlharm$
```

FIGURE B.3: Deploying HARM on AWS

B.3 Optimising HARM on Eclipse Development Environment

Eclipse provides a development environment that allows running processes connect to its debug environment. Changes to HARM can be observed in this way. It involves a 4 step process:

1. Modify the code within HARM and PyBrain to optimise HARM's functionality
2. Run HARM within the Eclipse Environment
3. Link the debugger to the running HARM process
4. Log into honeypot to see HARM's functionality

Step 1: This step has already been discussed and identified in chapter 6, fig. 3.11. This allows HARM to select Q-Learning, SARSA, ϵ-greedy, ϵ-greedy values, Boltzmann Softmax and State Dependent.

Step 2: Running HARM's *builder* in Eclipse will display the process ID (PID) in the console window. This is used to link to the debugging process.

Step 3: Linking the PID from step 2 is shown in fig. B.4. HARM's PID is attached to the debugger to observe its functionality.

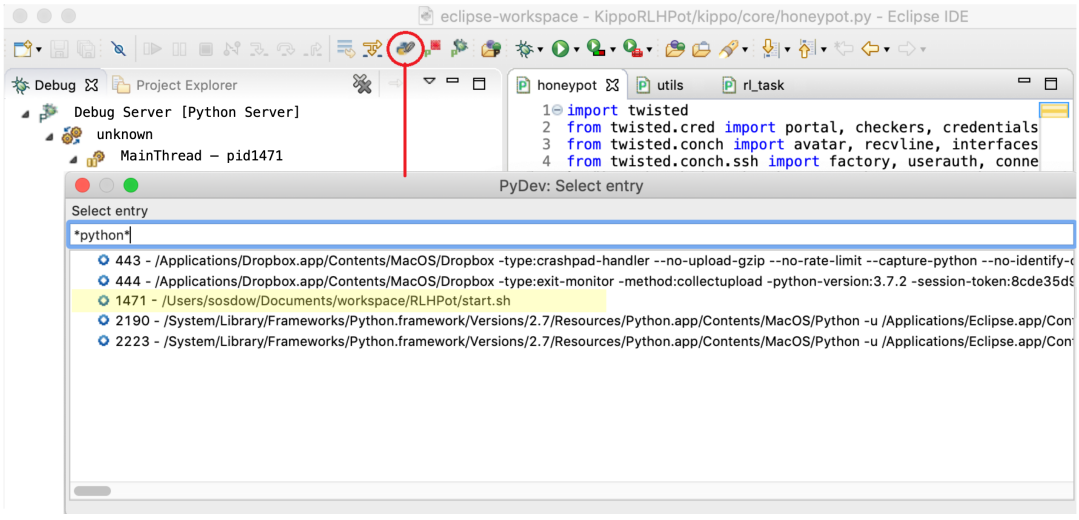


FIGURE B.4: Debugging within Eclipse

Step 4: In order to stream data through HARM and observe the optimisation from step 1, HARM's honeypot has to be accessed through a terminal window or equivalent application (such as PuTTY). Fig. B.5 shows a simple login using loopback IP 127.0.0.1 on port 2222. The password is configured within HARM. Any commands or command streams are immediately observed in the Eclipse environment and stored for analysis.

```
ubuntu@ip-172-31-35-172:/June20/home$ ssh root@127.0.0.1 -p 2222
The authenticity of host '[127.0.0.1]:2222 ([127.0.0.1]:2222)' can't be established.
RSA key fingerprint is c3:07:02:aa:55:ba:ac:e5:3f:74:16:86:5a:ef:99:41.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[127.0.0.1]:2222' (RSA) to the list of known hosts.
Password:
ZoneC-Gateway:~#
```

FIGURE B.5: Logging into HARM's process

After running a streaming process, the data can be extracted from the log files and MySQL database (Appendix C). Thereafter steps 1 - 4 can be run again using different settings from step 1. This requires the current process to be stopped from a terminal window (*kill -9 <PID>*).

B.4 Modifying the Captured Dataset to Create an Input Stream

The captured dataset is extracted as *stream.txt* in appendix C.3. Optimisation involves streaming this file through HARM in a controlled environment. Subsection 3.3.2.2 explains the functionality of this process that reflects the malware methods of the stream.txt file. The Python code to create this *Action Daemon* is displayed in fig. B.6.

```
#!env python

import argparse
import pprint
import sys

def read_harm(harm):
    # Pass in the name of the harm file, default 'prev.txt'
    action = []
    harm_in = open(harm, 'r')
    harm_values = harm_in.readlines()

    for i in range(4):
        print("i = %d" % (i))
        index = -1 - i
        print('i = %d, index = %d' % (i, index))
        action.append( harm_values[index])
    harm_in.close()
    return action

def write_harm(harm, value):
    harm_out = open(harm, 'a')
    harm_out.append(value)

parser = argparse.ArgumentParser(description='Stream safety check')
parser.add_argument('--input', dest='input', action='store', default='stream.txt',
                    help='Input stream file')
parser.add_argument('--harm', dest = 'harm', action='store', default='prev.txt',
                    help='"Harm" file')

args = parser.parse_args()

input = args.input
harm = args.harm
print('input = %s, harm = %s' % (input, harm))

first_line = True

print("Input file is %s, harm file is %s" % (input, harm))
try:
    input_f = open(input, 'r')
except IOError as e:
   print("Input file not found. Aborting")
   sys.exit(0)

action_array = read_harm(harm)
try:
    harm_f = open(harm, 'r+')
    harm_data = harm_f.readlines()
except IOError as e:
    print("Unable to open harm file %s" % (harm))
    sys.exit(0)

action = read_harm(harm)

command = open(input)
while True:
    print()
    try:
        line = command.readline()
    except:
        sys.exit(0)
    print(line)
    if first_line:
        print(line)
        #return line - To honeypot.py
    else:
        if action[0] == 'block':
            if action[1] == 'block' and action[2] == 'block':
                sys.exit(0)
            else:
                print(line)
                #return line - To honeypot.py
        elif action[0] == 'allow':
            print(line)
            #return line - To honeypot.py
        elif action[0] == 'substitute':
            if action[1] == 'substitute' and action[2] == 'substitute' and action[3] == 'substitute':
                sys.exit(0)
            else:
                print(line)
                #return line - To honeypot.py
```

FIGURE B.6: Python Code for Action Daemon

Appendix C

Data Analytics

C.1 Schematic

The flowchart in fig. C.1 presents the datastores, processes and resultant datasets used to generate images in journal presentations. This figure is meant to assist the understanding of how the datasets and code in the repositories of Appendix A are manipulated by code presented here in Appendix C.

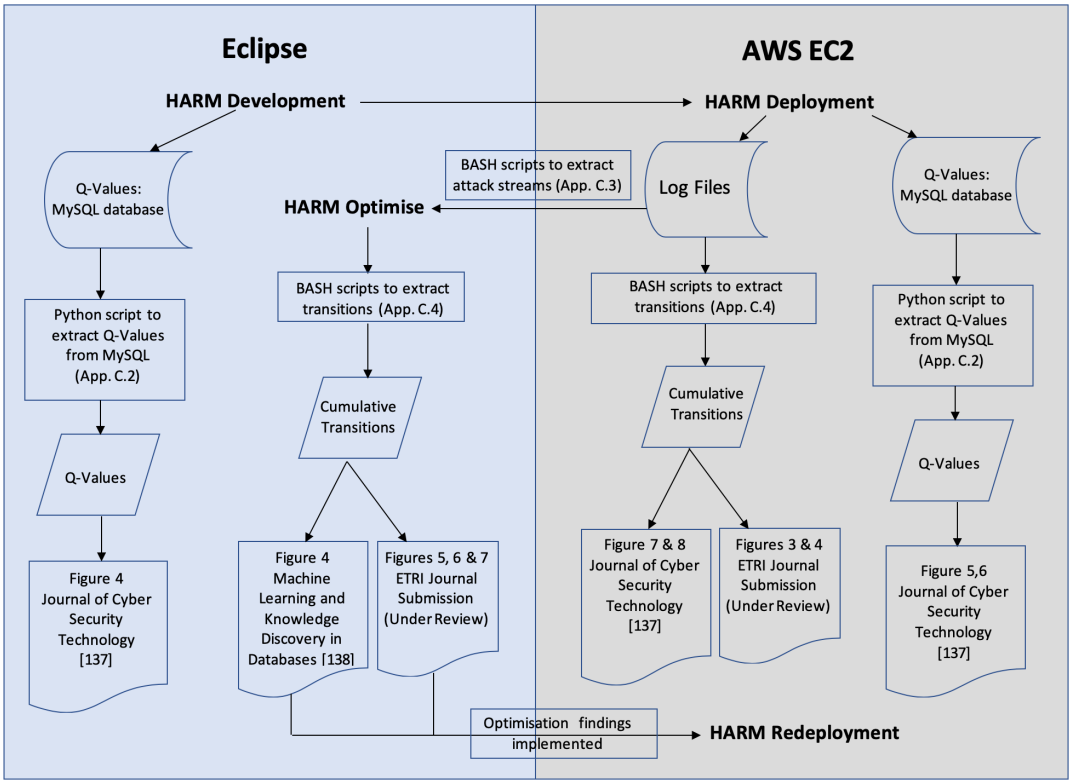


FIGURE C.1: Data Analytics on Datasets

C.2 Python script to extract Q-Values from MySQL

The Q-values calculated by the PyBrain module are returned to Kippo and stored in a MySQL database. These Q-values are extracted using the code in fig. C.2 and are used to generate graphs in journal *Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware* [137].

```
import sys
sys.path.append("../")

import kippo
from zope.interface import implements

from kippo.core import ttylog, fs, utils
from kippo.core.userdb import UserDB
from kippo.core.config import config
from kippo.dblog.kippo_mysql import *
import commands
import ConfigParser
from kippo.core.config import *
from kippo.core.constants import *

for row in getDB().getCases():
    if row["rl_params"] != "" and row["rl_params"] is not None:
        l = row["rl_params"].replace("[", "").replace("]", "")
        l = l.strip().split()
        print ",".join(l)
```

FIGURE C.2: Python script to extract Q-Values from MySQL

C.3 BASH Scripts to Extract Attack Stream

Every command interaction has *CMD* in the log file therefore this is used to identify attack lines. The Bash script in fig. C.3 is executed in the Kippo log files directory. Depending on the complexity of the attack code, other *$ columns* are required to display the entire line. A cropped example of the attacking stream is displayed. The result of the Bash script can be outputted to the *stream.txt* as seen in the figure. This file was further parsed to create other charts and images for conference papers, journal submissions and this thesis document. It was also used to inform the input stream into HARM's optimisation.

```
Seamuss-MacBook-Pro:log sosdow$ grep CMD kipp* | grep "$line" | awk '{ print $12, $13, $14, $15 }'
/gweerwe323f
echo -e \x47\x72\x6f\x70/dev' > /dev/.nippon
cat /dev/.nippon
rm -f /dev/.nippon
echo -e \x47\x72\x6f\x70/dev/shm' > /dev/shm/
cat /dev/shm/.nippon
rm -f /dev/shm/.nippon
echo -e '\x47\x72\x6f\x70/dev/pts' > /dev/pts/.nippon
cat /dev/pts/.nippon
rm -f /dev/pts/.nippon
/gweerwe323f
cat /bin/echo
/bin/busybox cp
/gweerwe323f
mount
/gweerwe323f
echo -e '\x47\x72\x6f\x70/dev' > /dev/.nippon
cat /dev/.nippon
rm -f /dev/.nippon
echo -e \x47\x72\x6f\x70/dev/shm' > /dev/shm/
cat /dev/shm/.nippon
rm -f /dev/shm/.nippon
echo -e '\x47\x72\x6f\x70/dev/pts' > /dev/pts/.nippon
cat /dev/pts/.nippon
rm -f /dev/pts/.nippon
/gweerwe323f
cat /bin/echo
/gweerwe323f
cd /
<output cropped>
Seamuss-MacBook-Pro:log sosdow$ grep CMD kipp* | grep "$line" | awk '{ print $12, $13, $14, $15 }' > stream.txt
```

FIGURE C.3: BASH Scripts to Extract Attack Stream

C.4 BASH Scripts to Extract Transitions

The number of transitions of the attack sequences is used to compare with previous research and to compare the performance of HARM's policies of reinforcement learning. The extraction requires a combination of Bash commands to create text files and the subsequent use of these text files to act as inputs to create datasets. This section presents some of the methods used to extract specific information from the captured honeypot datasets referenced in appendix A. Variations of these methods were also used for images in journal publications and conference proceedings.

Fig. C.4 displays the Bash command to extract the dates and times of successful attack interactions. Every successful attack generates a *TTY* file therefore this is used to identify them. This figure shows a cropped output and also a Bash command outputting the data to *dates.txt*. This text file will be used as an input into the the next phase of the data extraction.

```
Seamuss-MacBook-Pro:log sosdow$ grep TTY kipp* | awk '{print $1, $2}'| sed 's/^[^:]*://g'| sort -n | uniq
2017-11-28 15:49:51+0000
2017-11-28 15:54:32+0000
2017-11-28 16:01:03+0000
2017-11-28 18:35:11+0000
2017-11-28 18:36:46+0000
2017-11-29 02:33:18+0000
2017-11-29 03:55:11+0000
2017-11-29 03:58:44+0000
2017-11-29 05:03:51+0000
2017-11-29 05:04:52+0000
2017-11-29 05:05:41+0000
2017-11-29 05:05:51+0000
2017-11-29 05:08:09+0000
2017-11-29 05:13:48+0000
2017-11-29 05:15:32+0000
2017-11-29 05:18:16+0000
2017-11-29 05:19:23+0000
2017-11-29 05:21:01+0000
2017-11-29 05:22:24+0000
2017-11-29 05:23:26+0000
2017-11-29 09:57:22+0000
2017-11-29 09:57:54+0000
2017-11-29 10:03:35+0000
2017-11-29 10:10:56+0000
2017-11-29 10:12:58+0000
2017-11-29 10:41:14+0000
2017-11-29 10:43:58+0000
2017-11-29 10:49:15+0000
2017-11-29 10:56:25+0000
2017-11-29 11:04:43+0000
2017-11-29 11:07:42+0000
2017-11-29 12:01:49+0000
2017-11-29 12:01:56+0000
<output cropped>
Seamuss-MacBook-Pro:log sosdow$ grep TTY kipp* | awk '{print $1, $2}'| sed 's/^[^:]*://g'| sort -n | uniq > dates.txt
```

FIGURE C.4: BASH Script to Extract Attack Dates and Times

Fig. C.5 is used to collate and count the number of attacks in 8-hour segments, for all the attacks occurring in the *dates.txt* file. For this example, the Bash script is saved as *Ep15Sarsa2.sh* and is made executable *(chmod 700)*.

```
#!/bin/bash
while read line
do
        echo -e "$line"

        # Search for the first 8 hours of attacks 00:00:00 to 07:59:59
        echo -en "00:00:00 -> 07:59:59\t\t"
        grep CMD ~/Dropbox/AWSLogs/kipp* | grep $line | awk '{print $1 " " $2}' | egrep ' 0[0-7]:' | wc -l

        # Search for the second 8 hours of attacks 08:00:00 to 15:59:59
        echo -en "08:00:00 -> 15:59:59\t\t"
        grep CMD ~/Dropbox/AWSLogs/kipp* | grep $line | awk '{print $1 " " $2}' | egrep ' 0[8-9]:| 1[0-5]:' | wc -l

        # Search for the third 8 hours of attacks 16:00:00 to 23:59:59
        echo -en "16:00:00 -> 23:59:59\t\t"
        grep CMD ~/Dropbox/AWSLogs/kipp* | grep $line | awk '{print $1 " " $2}' | egrep ' 1[6-9]:| 2[0-3]:' | wc -l

        echo ""

done < $'dates.txt'
```

FIGURE C.5: BASH Script to Count the Number of Attacks in 8-hour Segments

Ep15Sarsa2.sh is executed using *dates.txt* as input. Fig. C.6 displays a cropped output from executing this command. The image also shows that the result of this can then be outputted to *Ep15Sarsa.dat* file. This dataset was used to create other charts and images for conference papers, journal submissions and this thesis document.

```
Seamuss-MacBook-Pro:log sosdow$ ./Ep15Sarsa2.sh < dates.txt
2017-12-04
00:00:00 -> 07:59:59                12
08:00:00 -> 15:59:59                14
16:00:00 -> 23:59:59                13

2017-12-05
00:00:00 -> 07:59:59                15
08:00:00 -> 15:59:59                12
16:00:00 -> 23:59:59                12

2017-12-06
00:00:00 -> 07:59:59                12
08:00:00 -> 15:59:59                15
16:00:00 -> 23:59:59                12

2017-12-07
00:00:00 -> 07:59:59                12
08:00:00 -> 15:59:59                18
16:00:00 -> 23:59:59                19

2017-12-08
00:00:00 -> 07:59:59                24
08:00:00 -> 15:59:59                38
16:00:00 -> 23:59:59                12

2017-12-09
00:00:00 -> 07:59:59                29
08:00:00 -> 15:59:59                18
16:00:00 -> 23:59:59                36

<output cropped>
Seamuss-MacBook-Pro:log sosdow$ ./Ep15Sarsa2.sh < dates.txt > Ep15Sarsa.dat
```

FIGURE C.6: Executing Bash Script to Generate Number of Attacks in 8-hour Segments

Daily attack commands can be collated and exported (*Cumulative.dat*) using a similar method as seen in fig. C.7. This dataset was also used to create other charts and images for conference papers, journal submissions and this thesis document.

```
Seamuss-MacBook-Pro:log sosdow$ cat Collate.sh
#!/bin/bash
while read line
do
     echo -en "$line"
     grep CMD ~/Dropbox/AWSLogs/kipp* | grep "$line" | awk '{ print $12, $13 }' | wc -l
done < $'dates.txt'
Seamuss-MacBook-Pro:log sosdow$ ./Collate.sh < dates.txt
2017-11-28      32
2017-11-29      55
2017-11-30      68
2017-12-01      48
2017-12-02      46
2017-12-03      39
2017-12-04      39
2017-12-05      49
2017-12-06      74
2017-12-07      83
2017-12-08      45
2017-12-09      26
2017-12-10      46
2017-12-11      71
<Output Cropped>
Seamuss-MacBook-Pro:log sosdow$ ./Collate.sh < dates.txt > Cumulative.dat
```

FIGURE C.7: Executing Bash Script to Generate Number of Daily Attack Commands

Appendix D

Journal Articles Under Review

D.1 SelfHARMing Malware: An Adaptive IoT Honeypot for Automated, Repetitive Malware

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Self HARMing Malware†

Seamus Dowling*[1] | Michael Schukat[2] | Enda Barrett[2]

[1]Mayo Campus, Galway Mayo Institute of Technology, Mayo, Ireland
[2]College of Engineering and Informatics, National University of Ireland Galway, Galway, Ireland

Correspondence
*Seamus Dowling, GMIT, Castlebar, Mayo, Ireland. Email: seamus.dowling@gmit.ie

Summary

Internet-of-Things (IoT) contains physically constrained devices which impacts on the security of IoT deployments. This can make them vulnerable to dedicated IoT attack software such as Mirai and increases the attack surface available to IoT malware developers. This article describes the functionality, deployment and assessment of HARM, an adaptive IoT **H**oneypot for **A**utomated, **R**epetitive **M**alware. It presents findings from the live deployment of HARM that uses reinforcement learning to learn from attack interactions. It also demonstrates that dedicated IoT honeypot datasets contain attack information that is repetitive and automated. Standard honeypot deployments have scripted responses which terminates attack interactions when malware determines that there is nothing to be gained by continuing. This article enhances standard honeypots by exploiting malware characteristics of automation and repetition. It identifies that the learning evolution of HARM correlates with the discovery of the entire attack sequence. It also uses the captured dataset to assess reinforcement learning policies so as to optimise further HARM deployments.

KEYWORDS:
Honeypot, Reinforcement Learning, Adaptive, Internet of Things

1 | INTRODUCTION

The security of data collected and transmitted by Internet-of-Things (IoT) devices depends on the physical resources available to those devices. Full function devices (FFD) and reduced function devices (RFD) will have constrained resources when implementing services such as security. This leaves them vulnerable to compromise and increases the attack surface for malware [1]. It is estimated that there will be over 20 billion IoT devices connected to the Internet by 2020 [2]. These can have both wired and wireless accessibility and facilitated with the implementation of IPv6 into the architectural stack [3]. Large volumes of data processed by these devices give rise to new challenges [4]. Privacy concerns arise in relation to the storage and use of this data [5]. Already attack code specifically targeting IoT devices has been made available, spawning new variants and using new attack vectors and methods [6]. Honeypots can play a reactive role in analysing the methods used by new malware [7] but are typically deployed to gather information for retrospective analysis. This can make them redundant for rapid detection of new malware variants as the data collection process is longitudinal [8]. New honeypot methods need to be considered to adapt to the emerging threat posed by increased IoT deployments. This article first describes an IoT honeypot deployment. The resultant dataset contains predominately repetitive data of automated malware methods [9]. The article then describes the integration of reinforcement learning into the existing IoT honeypot technology to exploit these characteristics, namely automation and repetition. The resultant adaptive IoT honeypot, a **H**oneypot for **A**utomated, **R**epetitive **M**alware (HARM) is deployed to attract and capture attack sequences targeting IoT devices. It demonstrates that the adaptive IoT HARM uses these malware characteristics to learn the best actions to take when interacting with a known IoT bot. By doing so it expeditiously determines the entire attack command sequence. The attack dataset is then used to evaluate reinforcement learning policies so as to assess HARMs performance and optimise for future deployments.

† An Adaptive IoT Honeypot for Automated, Repetitive Malware.
[0]**Abbreviations:** HARM, Honeypot for Automated Repetitive Malware

Our proposed state action space formalism is designed to target automated and repetitive malware. Thus we propose the following contributions over existing work:

- Demonstrate that the learning evolution of HARM, correlates with key increases in attack commands and culminates with the realisation of the entire attack sequence.
- The captured automated and repetitive dataset can be used in a controlled environment to evaluate reinforcement learning policies so as to optimise future adaptive IoT HARM deployments.
- Q-Learning outperformed SARSA as a learning agent on HARM, in controlled experiments.

2 | EVOLVING IOT THREATS

As the Internet evolves to include IoT, society will be connected in more valuable and relevant ways. A network of combined devices can impact society with the deployment of sensors to monitor societal ecosystems. Optimising power grids, controlling traffic flows, monitoring pollution and urban environmental ecosystems create smart cities for the betterment of its population. Heterogeneous technologies will require integration at a physical level. Data collected and collated will require transmission in a compatible format and then collated in a manner that ensures relevant mining. Information from IoT systems will come from a multitude of sources. As well as smart objects from wireless sensor networks (WSN) and other IoT installations, society itself will contribute to the collection of data. Data captured will require a communications stack built onto the electronics, to transmit and receive as part of a network [10] and provide reachability to all connected devices. Data, security and communications all differ for smart objects within these components. IEEE defines two device types that can participate in an 802.15.4 network; a full-function device (FFD) and a reduced-function device (RFD). An FFD is capable of serving as a personal area network (PAN) coordinator or a coordinator. An RFD is not capable of serving as a PAN coordinator. An RFD is intended for applications that are extremely simple and does not have to send large amounts of data. An RFD uses minimal resources and memory capacity. Communications capabilities for smart objects will range from very basic to highly complex. With such a diversity of smart objects, interoperability will require collaboration and compromise. Security measures implemented will depend on the objects processing capabilities, leaving some objects more vulnerable than others. The value of the data processed or stored also depends on the physical characteristics of the smart object. Security of communications and data, and access and participation are critical components to IoT deployments. Encryption techniques such as PKI are often too power and resource hungry for IoT devices therefore alternatives need to be considered [11]. Previous research on IoT honeypots highlight the diversity of IoT objects [12].

Table 1 presents an example of protocols used at data, application, transport and infrastructure layers for FFD and RFD devices. Large volumes of big data will be processed by IoT deployments. Concepts such as smart cities give rise to issues concerning the privacy of their citizens and the use of data [4]. Information stored and communicated about location, social activity and profiling need to be secure to engender trust and promote the use of smart city applications. The diversity of data from objects and citizens will require secure storage and meaningful integration [13]. All smart objects and user end devices will require the capabilities to securely exchange data between each other, coordinating devices, gateways and subsequent storage locations. All objects and communication channels are open to probing and compromise. The US Department of Homeland Security produced a risk assessment for cyber-physical systems (CPS) in a smart city [14]. It highlighted potential vulnerabilities for CPS or other related technologies as integral parts of the fabric of smart city infrastructure. The Industrial IoT (IIoT) Consortium published a security framework and an approach to assess cybersecurity in IIoT systems [15]. WSNs can be a target of cyber attacks as security may not be feasible on constrained nodes. Security of a smart object depends on embedded security, if any and their ability to defend against cyber attacks.

TABLE 1 FFD and RFD Protocols

Layer	Protocol
Data	XML, HTML, EXI
Application and Transport	DDS, CoAP, AMQP, MQTT, XMPP HTTP, REST, TCP, UDP, SSH
Infrastructure	RPL, 6LoWPAN, IPv4, IPv6, 802.15.4, ZigBee, Bluetooth, LoraWAN, 3G, 4G, LTE-A, EPCGlobal, Z-Wave, WiFi, RFID

3 | EVOLVING HONEYPOTS

3.1 | Standard Honeypots

A honeypot is an analytical tool and it's role is to deceive and collect attack information. This information can be analysed retrospectively to determine the modus operandi of attackers. Honeypots can have low or high interaction. Provos [16] in 2003 presented Honeyd, an easy to deploy, low risk honeypot. It details how to deploy virtual honeypots with different IPs safely. Honeyd acted as a catalyst for the development of further low interaction honeypots. Nepenthes [17] and Argos [18] became very popular global honeypot tools. High interaction honeypots such as Kippo, provide backend databases to collect all activity such as IP addresses, timestamp, attempts, interactions, commands, downloads and executions [19] . Downloaded files can be sandboxed and analysed [20] . After an attacker has compromised a honeypot, it will attempt to interact in a structured manner [21] . The initial engagement for an attack, post compromise, is to examine the hardware and software to determine if progression is relevant. On a live production system, this will return the underlying architecture, CPU, uptime, operating system, user privileges and further relevant information. An attack sequence may then attempt to modify the host system. On a honeypot, engaging the attack sequence at this point prolongs activity. It then attempts to download, install and run malware to complete the compromise. There are often legal and ethical issues associated with operating honeypots [22]. In a desire to gather as much information as possible on attacker behaviour, a honeypot could allow the execution of malicious code [23] . The honeypot developer could be liable if their honeypot inadvertently becomes involved in further attacks. Entrapment could be a mitigating factor when it comes to the prosecution of an attacker.

3.2 | Adaptive Honeypots

The operation of standard honeypots highlights the need for creating honeypots that prolong attacker interaction for an optimum time period, without breaching legal or ethical responsibilities. Machine learning techniques have previously been applied to honeypots. These techniques have analysed attacker behaviour retrospectively on a captured dataset. Supervised and unsupervised methods are used to model malware interaction and to classify attacks [24, 25, 26]. Wagener [27] uses reinforced learning to extract as much information as possible about the intruder. A honeypot called Heliza was developed to use reinforcement learning when engaging a human attacker. The honeypot implemented behavioural strategies such as blocking commands, returning error messages and issuing insults. Pauna [28] also presents an adaptive honeypot using similar reinforced learning algorithms as seen in Heliza. He proposes improvements in scalability, localisation and learning capabilities.

3.3 | IoT Honeypots

The popularity of IoT deployments has attracted malware targeting end devices [29]. Recently, the Mirai botnet spawned multiple variants targeting IoT devices through various attack vectors [6]. Honeypots played an active part in capturing and analysing the Mirai structure [7]. Variants were captured on Cowrie, a version of the popular Kippo honeypot [19]. New IoT versions of honeypots are being designed continuously to capture new IoT malware on different attack vectors. IoTPot is a bespoke honeypot designed to analyze malware attacks targeting IoT devices [30]. ConPot is a SCADA honeypot developed for critical IIoT architectures [31]. The repetitive and automated nature of the malware is visible on a smaller dataset from a IoT honeypot [9]. ThingPot [32], simulated vulnerable end devices. IoTCandyJar presented as a range of industrial, commercial and end devices which could be accessed across multiple attack vectors [33].

4 | AUTOMATION AND REPETITION

Honeypot deployments are longitudinal and the resultant dataset contains large amounts of repetitive data. This allows researchers to ascertain patterns in the behaviour over a period of time. Global honeypots often operate for long periods with a view to collecting large datasets. Temporal variances and IP blocks provide interesting insight into the propagation methods used by bots and botnets. Diurnal patterns can be used to model malware activity [34].

This diurnal pattern was evident in the captured dataset from an IoT honeypot deployed over 3 months in late 2015 to early 2016 [9]. Fig. 1 demonstrates the global pattern of infected devices contributing *repetitively* to attacks on the honeypot, extracted from the resultant dataset. Ironically it predated the Mirai bot which was deployed in mid 2016 to target IoT devices [6]. The IoT honeypot simulated a ZigBee gateway on a SSH attack vector. The attack types can be identified by sandboxing the downloaded files, observing shell interactions and consulting threat advisories. Overall, 99.6% of the traffic encountered on the honeypot was *automated*. Automated attack types were examined and collated. They were identified as *Dictionary*, *Recon*, *Failed*, *Launch* commands, *XOR DDoS* and *BillGates*. Dictionary attacks continuously probed with username/password

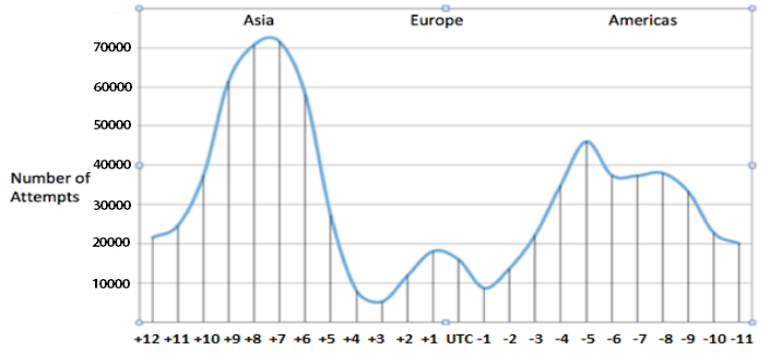


FIGURE 1 Temporal patterns of honeypot activity [9]

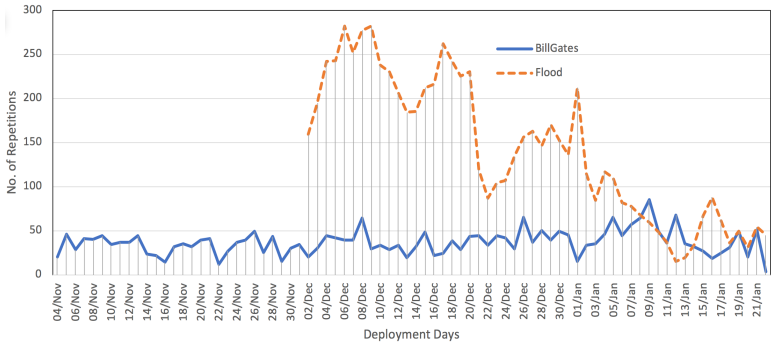


FIGURE 2 Repetition of *BillGates* and *Launch Flood* Attack Types

combinations. They were eliminated from the dataset as they were pre-compromise. Failed attacks made an initial connection with correct username and password but failed on authentication. The XOR DDoS and BillGates Botnet provided better material for examination. The downloaded files were sandboxed and the scripts were analysed. They demonstrated automated methods to gather information on variables such as compilers, CPU and operating systems. Both treated the honeypot as an SSH device primarily and concentrated on compromising it in that regard. Further analysis revealed in fig. 2, shows the constant repetition of two of the more complex automated scripts.

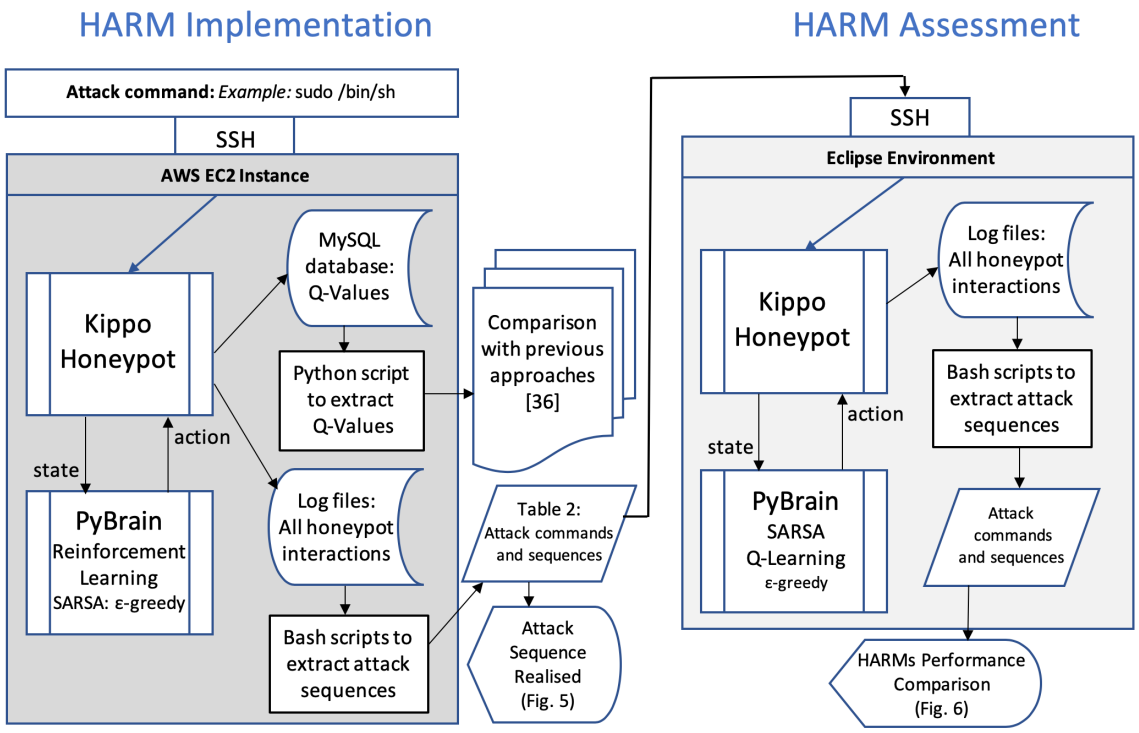


FIGURE 3 SelfHARMing Malware: Implementing and Assessing HARM

5 | HARM DEVELOPMENT METHODOLOGY

The development of HARM involves 2 stages:

- **Implementation:** Integrating reinforcement learning into an IoT honeypot
- **Assessment:** Deploying and Assessing HARMs performance

Together these two stages use the very characteristics of malware, automation and repetition to assess the performance of HARM in order to improve subsequent deployments. Fig. 3 presents the 2 stages as a concept. The *implementation* and *assessment* stages are detailed in sections 5.1 and 5.2 respectively.

5.1 | Implementation

Our proposed **H**oneypot for **A**utomated, **R**epetitive **M**alware involves the integration of reinforcement learning into the existing IoT honeypot technology (section 4) to exploit the automated and repetitive characteristics of malware. The reinforcement learning state/action space and reward function is designed to increase the number of commands from the attack sequence.

5.1.1 | Reinforcement Learning

Reinforcement learning is a machine learning technique in which a learning agent learns from its environment, through trial and error interactions. Rather than being instructed as to which action it should take given a specific set of inputs, it instead learns based on previous experiences as to which action it should take in the current circumstance. Reinforcement learning problems can generally be modelled using Markov Decision Processes (MDPs). In fact reinforcement learning methods facilitate solutions to MDPs in the absence of a complete environmental model. This is particularly useful when dealing with real world problems such as honeypots, as the model can often be unknown or difficult to approximate. MDPs are a particular mathematical framework suited to modelling decision making under uncertainty. If there is no complete model available, then reinforcement learning methods have proven efficacy in solving MDPs. During the reinforcement learning process the agent can select an action which exploits its current knowledge or it can decide to use further exploration. Reinforcement learning provides parameters to help the learning environment decide on the reward and exploration values. Throughout its deployment, HARM is considered to be an environment with integrated

reinforcement learning. We are using SSH as an access point and a simulated Linux server as a vulnerable environment. SSH is responsible for 62% of all compromise attempts [35]. Within this environment, the server has states that are examined and changed with bash scripts. Examples are *iptables*, *wget* and *sudo*. The reinforcement learning agent can perform actions on these states such as to allow, block or substitute the execution of the scripts. The environment issues a reward to the agent for performing that action. The agent learns from this process as the honeypot is attacked and over time learns the optimum policy π^*, mapping the optimal action to be taken each time, for each state s. The learning process will eventually converge as the honeypot is rewarded for each attack episode. This temporal difference method for on-policy learning uses the transition from one state/action pair to the next state/action pair, to derive the reward. *State*, *Action*, *Reward*, *State*, *Action* also known as SARSA, is a common implementation of on-policy reinforcement learning (3). The reward policy Q is estimated for a given state s_t and a given action a_t. The environment is explored using a random component ϵ or exploited using learned Q values. The estimated Q value is expanded with a received reward r_t plus an estimated future reward $Q(s_{t+1}, a_{t+1})$, that is discounted (γ). A learning rate parameter is also applied (α).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{1}$$

As each attack is considered an episode, the policy is evaluated at the end of each episode. SARSA is implemented with epsilon-greedy policy = 0.10, discount factor γ=1 and step size α=0.5.

Previous contributions demonstrate that there is a relationship between honeypot and malware development and evolution. Botnet traffic is highly automated as it attacks, compromises and reports without any intervention. From a malware developer's perspective, there is very little human interaction, post launch. Honeypot evasion techniques can be implemented, as new honeypots are uncovered. HARM uses reinforcement learning to prolong interaction with an attack sequence. Initial bot commands will attempt to return known honeypot responses, or positives to false requests. Depending on the response, the bot will either cease or amend its actions. We want our honeypot to learn and be rewarded for prolonging interaction. Therefore our reward function is to increase the number of commands from the attack sequence. Attacker commands can be categorised into the following:

L Known Linux bash commands (wget, cd, mount, chmod, etc.)

C Customised attack commands (Commands executing downloaded files)

CC Compound commands (Multiple commands with bash separators/operators)

NF known commands not facilitated by honeypot (The honeypot is configured for 75 of the most commonly used bash commands)

O Other commands (Unhandled keystrokes such as ENTER and unknown commands)

We propose a transition reward function whereby the learning agent is rewarded if a bot command is an input string i, comprised of bash commands (L), customised commands (C) or compound commands (CC). Therefore Y = C ∪ L ∪ CC. For example *iptables stop* is an input string i that transitions the state (s) of iptables on a Linux system to the next command. Other commands or commands not facilitated by the honeypot are not given any reward. We propose an action set a = allow, block, substitute. *Allow* and *Block* are realistic responses to malware behaviour. Botnets can use complex if-else structures to determine next steps during compromise [9]. Using *Substitute* to return an alternative response to an attack command potentially increases the number of attack transitions to newer commands. This action set coupled with state set Y, creates a discrete state/action space. The reward function is as follows:

$$r_t(s_i, a) = \begin{cases} 1, \text{if i} \in \text{Y} \\ 0, \text{otherwise} \end{cases} \tag{2}$$

$$\text{where Y = C} \cup \text{L} \cup \text{CC}$$

The formula for transition reward r_t, based on the state action pair (s,*a*), is as follows:

- If the input string i is a customised attacker command C or a known bash command L or a compound command CC, then reward $r_t(s_i, a) = 1$
- otherwise the reward $r_t(s_i, a) = 0$

The adaptive honeypot has the following elements:

- Modified honeypot. Kippo is a widely used honeypot distribution that logs all SSH interactions in a MySQL database. This has been modified to generate the parameters to pass to the learning agent. Depending on the action selected by the learning agent, the honeypot will allow, block or substitute attack commands

- SARSA agent. This module receives the required parameters from the adaptive honeypot and calculates $Q(s_t, a_t)$ as per equation 2. It determines the responses chosen by the adaptive honeypot and learns over time which actions yield the greatest amount of reward.

More detailed reward functionality and comparison with previous research for HARM is available [36].

5.2 | Assessment

The captured dataset can be used to evaluate the policies of reinforcement learning. The command sequences in the dataset can function as an input stream into the adaptive honeypot within a controlled Eclipse environment [1], modified to mirror malware methods. HARM was deployed on the Internet using SARSA (eq. 1). Sutton and Barto [37] presents algorithms for both SARSA and Q-Learning. Algorithm 1 shows that SARSA chooses s_{t+1} and a_{t+1} prior to updating the Q function. Algorithm 2 shows Q-Learning which first updates the Q function and then chooses the next action based on the updated Q function.

HARM's implementation of SARSA is provided by PyBrain [38]. PyBrain is a machine learning library which provides for the definition of the state/action space. Within PyBrain, the implementation of Algorithms 1 and 2 are facilitated with Python code as follows:

- **Algorithm 1:** *self.module.updatevalue(self.laststate, self.lastaction, qvalue + self.alpha * (self.lastreward + self.gamma * qnext - qvalue))*
- **Algorithm 2:** *self.module.updatevalue(self.laststate, self.lastaction, qvalue + self.alpha * (self.lastreward + self.gamma * maxnext - qvalue))*

For this code to function, the adaptive honeypot stores variables for the current state, action and reward. These are passed to PyBrain to select the best action. This action is subsequently passed back to HARM to perform that action and update the Q function. As part of the assessment process, PyBrain also provides for the following policies:

- Epsilon greedy
  A discrete explorer that mostly executes the original policy but sometimes returns a random action.
- Boltzmann Softmax
  A discrete explorer that executes actions with a probability that depends on their action values.
- State dependent
  A continuous explorer that disrupts the resulting action with added, distributed random noise.

Algorithm 1 Reinforcement Learning SARSA

Initialise $Q(s,a)$ arbitrarily
repeat (for each episode):
Initialize s
Choose a from s using policy derived from Q (e.g., ε-greedy)
repeat
Take action a, observe r, s_{t+1}
Choose a_{t+1} from s_{t+1} using policy derived from Q (e.g., ε-greedy)
$Q(s_t, a_t) \leftarrow Q(s_t,a_t)+\alpha[r + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t, a_t)]$
$s \leftarrow s_{t+1}, a \leftarrow a_{t+1}$;
until s is terminal

Algorithm 2 Reinforcement Learning Q-Learning

Initialise $Q(s,a)$ arbitrarily
repeat (for each episode):
Initialize s
repeat
Choose a from s using policy derived from Q (e.g., ε-greedy)
Take action a, observe r, s_{t+1}
$Q(s_t, a_t) \leftarrow Q(s_t,a_t)+\alpha[r + \gamma \max_{a+1} Q(s_{t+1},a_{t+1}) - Q(s_t, a_t)]$
$s \leftarrow s_{t+1}$;
until s is terminal

[1] www.eclipse.org

```
from pybrain.rl.agents import LearningAgent
from pybrain.rl.learners import Q,SARSA
from pybrain.rl.explorers import EpsilonGreedyExplorer

import threading

class RL:
    def __init__(self):
    learner = SARSA()
    learner._setExplorer(EpsilonGreedyExplorer(0.1))
```

FIGURE 4 Adaptive Honeypot Reinforcement Learning Selection

HARM can import and use SARSA or Q-Learning from PyBrain. It can also specify the ϵ-greedy explorer component. These learning policies can be coded as *pybrain.rl.explorers*, as seen in fig. 4. This flexibility within the controlled environment provides for the assessment of HARMs performance. By modifying the variables (SARSA, Q-Learning, ϵ-greedy, ϵ-greedy values, Boltzmann Softmax, State dependent) within a controlled environment and streaming a captured dataset modified to mirror malware methods, the performance of HARM can be assessed. The results of this assessment stage can inform further deployments.

6 | RESULTS

The adaptive IoT HARM was developed to generate rewards on 75 states and is freely available [39]. The definition of the state/action space was facilitated within PyBrain [38]. Amazon Web Services (AWS) EC2 was used to facilitate the Internet facing honeypots. Kippo, PyBrain, MySQL and other dependencies were installed on the adaptive IoT honeypot EC2 instance. It was designed to be restrictive so as to reduce the capture of excessively repetitive malware. This required a period of initial monitoring, adjusting security access such as username/password combinations and restricting some repetitive state transitions that overly affected rewards. It was accessible through SSH for a period of 30 days (Nov/Dec 2017) and immediately started to record repetitive malware activity. A standard Kippo honeypot was deployed simultaneously for comparison purposes.

6.1 | Learning Evolution and Attack Sequence Realisation

Initially HARM logged dictionary, bruteforce and failed attempts. These are excluded as they represent pre-compromise interactions. Thereafter it captured other malware traffic including a Mirai-like bot. These commands all represent interactions post-compromise. This bot became the dominant repetitive attacking tool. Other SSH malware interacted with the honeypot and impacted the learning process and the realisation of the entire attack sequence of the Mirai variant. This Mirai variant was also repetitive in interacting with HARM averaging over 5 attacks per day. Table 2 presents a sample of the Mirai like bot command structure [6].

HARM's reward function is designed to prolong interaction and by default, increase the number of commands in an attack sequence (eq. 2). It calculated the reward values as it encountered each command in table 2. These reward values were extracted and collated for 100 distinct attack events [36]. The log files capturing the interactions (fig. 3) highlighted key points in the malware interaction. On the 8th, 12th and 17th December for example, the adaptive IoT HARM substituted a result for *cat* command, blocked a *mount* command or blocked a compound *echo* command causing the interaction with the attack sequence to increase. The honeypot continued to learn until attack 60 when it allowed all compound commands in the sequence to be executed (17th December). The adaptive HARM learning was also affected by the other infrequent malware types also encountered by HARM. This impacted upon the realisation of the entire attack sequence for dominant Mirai variant. Fig. 5 presents the learning evolution and realisation of the command sequence for this Mirai variant. A standard Kippo honeypot was deployed at the same time and also encountered the Mirai variant. It only ever executed the first 8 commands in the sequence before interaction terminated. It is included in fig. 5 for comparison purposes and demonstrates the improved functionality of HARM in prolonging interaction, realising entire attack sequences and ultimately capturing more relevant datasets for analysis.

TABLE 2 Mirai like bot sample [6]

Sequence	Bot Command
1	/gweerwe323f
2	sudo/bin/sh
3	/bin/busybox
4	/gweerwe323f
5	mount
6	/gweerwe323f
7	echo -e '\x47\x72\x6f\x70' >//.nippon
8	cat //.nippon
--	----------
38	/gweerwe323f
39	cat /bin/echo
40	cd/
41	wget http:// <RedactedIP>/bins/usb_bus.x86 -O - >usb_bus ; chmod 777 usb_bus
42	chmod 777 usb_bus
43	./usb_bus
44	/gweerwe323f

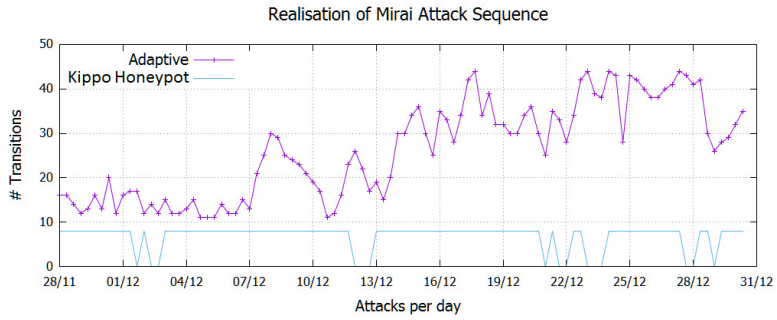


FIGURE 5 Attack Command Sequence Realisation

6.2 | HARM Assessment

To compare the performance of SARSA and Q-Learning implementations, the dataset from section 6.1 was streamed twice through HARM in a controlled Eclipse environment, modified to reflect malware methods. The first time the learner variable in fig. 4 was set to SARSA, the second time with it set to Q-Learning. ϵ-greedy remained constant at 0.1 for both. The honeypot captured the interactions, actions and number of commands in the attack sequence that represent transitions from s to s_{t+1}. These transitions were collated allowing for the comparison of SARSA and Q-Learning performance on the adaptive honeypot. Fig. 6 demonstrates the result of using two sets of variables (ϵ-greedy=0.1 for SARSA and Q-Learning). SARSA performed better initially having more interactions with the data stream. However Q-Learning was the first to realise the entire 44 command attack sequence at attack 63 compared with SARSA at attack 66.

The assessment process uses SARSA and Q-Learning with a set ϵ-greedy value. It could be expanded to create combinations of SARSA, Q-Learning, ϵ-greedy, ϵ-greedy values, Boltzmann Softmax and State dependent. This could be performed quickly in the controlled environment to determine the optimum combination against a trending variant. Subsequent timely deployments of HARM are then optimised to adapt and learn from automated and repetitive malware.

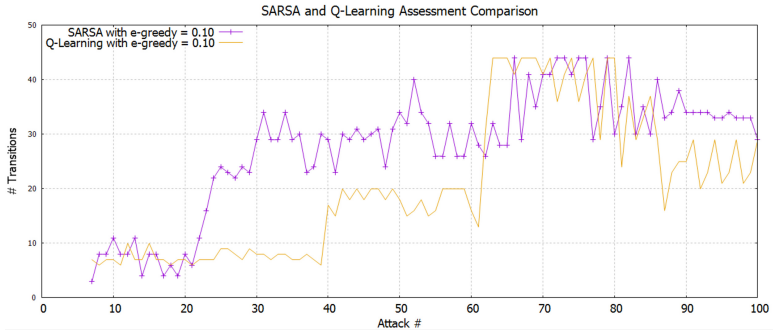


FIGURE 6 Performance of SARSA and Q-Learning

7 | CONCLUSIONS

Malware operations are predominately automated and repetitive. Bots provide a mechanism to propagate, compromise hosts and communicate with command and control (C&C). Human social engineering may play a part in facilitating infection of the end devices. Human interaction may be required as a botmaster communicates with the C&C. But the global process of generating a botnet of compromised hosts is highly automated and highly repetitive.

The design, deployment and findings of HARM presented in this paper turn those very characteristics of automation and repetition upon the malware itself. It identifies that these traits can be used to facilitate learning on the adaptive IoT honeypot and quickly realise an entire attack stream. There exists many tools that act as early warning systems for new malware such as zero-day attacks. Intrusion detection and prevention systems can flag and defend against anomalies. Honeypots have proven efficacy in capturing and analysing emerging malware threats. It has been pointed out that honeypots played a role in originally identifying the Mirai bot. As more IoT devices become connected to the Internet, new attack vectors emerge. Constrained resources on intermediate and end devices provide malware developers with a fertile ground for exploitation. To counter this, new methods of detection need to be considered. Methods that can expedite the analysis and identification of attacks, will improve overall security of IoT deployments and the Internet.

When honeypots are deployed, a period of monitoring and adjustment is required. This is done to ensure the honeypot is capturing relevant data and is not set up as too vulnerable or too robust. This limitation arises with HARM as well. It needed to be initially monitored so as to restrict the types of malware accessing it. It required the hardening of the honeypot against some attacks by modifying username/password combinations and denying some state transitions that excessively affected reward calculation. This initial setup allows for the thorough analysis of HARM on a bot such as a Mirai variant.

The efficacy of HARMs functionality needs to be ascertained against non IoT specific malware. Further deployment and assessment is required to observe *normal* malware interactions, to determine if HARM demonstrates improved functionality against all malware types. Future work on HARM would also have it identify attack types by initial command sequences, running separate reinforcement learning processes for each type and maintaining multiple reward datasets. The assessment of HARM identified Q-learning as being more efficient when realising the entire attack sequence, within a controlled environment. It would be relevant for future work to see if this is specific to a particular bot command stream or if this is the case for all malware attacks.

ACKNOWLEDGMENTS

Author contributions

Financial disclosure

None reported.

Conflict of interest

The authors declare no potential conflict of interests.

References

[1] Lei Xu, Chunxiao Jiang, Jian Wang, Jian Yuan, and Yong Ren, *Information security in big data: privacy and data mining*, Ieee Access **2** (2014), 1149–1176.

[2] R van der Meulen, *6.4 billion connected things will be in use in 2016*, Tech. Report Technical Report, Gartner, 2017, https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016. Accessed July 27, 2016.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito, *The internet of things: A survey*, Computer networks **54** (2010), no. 15, 2787–2805.

[4] Gema Bello-Orgaz, Jason J Jung, and David Camacho, *Social big data: Recent achievements and new challenges*, Information Fusion **28** (2016), 45–59.

[5] Liesbet Van Zoonen, *Privacy concerns in smart cities*, Government Information Quarterly **33** (2016), no. 3, 472–480.

[6] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al., *Understanding the mirai botnet*, 26th USENIX Security Symposium (USENIX Security 17), 2017, pp. 1093–1110.

[7] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas, *Ddos in the iot: Mirai and other botnets*, Computer **50** (2017), no. 7, 80–84.

[8] David Watson and Jamie Riden, *The honeynet project: Data collection tools, infrastructure, archives and analysis*, 2008 WOMBAT Workshop on Information Security Threats Data Collection and Sharing, IEEE, 2008, pp. 24–30.

[9] Seamus Dowling, Michael Schukat, and Hugh Melvin, *A zigbee honeypot to assess iot cyberattack behaviour*, Signals and Systems Conference (ISSC), 2017 28th Irish, IEEE, 2017, pp. 1–6.

[10] Luca Mottola and Gian Pietro Picco, *Programming wireless sensor networks: Fundamental concepts and state of the art*, ACM Computing Surveys (CSUR) **43** (2011), no. 3, 19.

[11] Tao Yan and Qiaoyan Wen, *A trust-third-party based key management protocol for secure mobile rfid service based on the internet of things*, Knowledge Discovery and Data Mining, Springer, 2012, pp. 201–208.

[12] Seamus Dowling, Michael Schukat, and Hugh Melvin, *Data-centric framework for adaptive smart city honeynets*, Smart City Symposium Prague (SCSP), IEEE, 2017, pp. 1–7.

[13] Kehua Su, Jie Li, and Hongbo Fu, *Smart city and the applications*, International conference on electronics, communications and control (ICECC), IEEE, 2011, pp. 1028–1031.

[14] Department of Homeland Security, *The future of smart cities: Cyper-physical infrastructure risk*, Tech. Report Technical Report, Department of Homeland Security, 2015, https://ics-cert.us-cert.gov/Future-Smart-Cities-Cyber-Physical-Infrastructure-Risk. Accessed June 10, 2018.

[15] Industrial Internet Consortium, *Industrial internet of things volume g4: Security framework*, Tech. Report Technical Report, Industrial Internet Consortium, 2016, http://www.iiconsortium.org/IISF.htm. Accessed May 3, 2019.

[16] Niels Provos, *Honeyd-a virtual honeypot daemon*, 10th DFN-CERT Workshop, Hamburg, Germany, vol. 2, 2003, p. 4.

[17] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling, *The nepenthes platform: An efficient approach to collect malware*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2006, pp. 165–184.

[18] Georgios Portokalidis, Asia Slowinska, and Herbert Bos, *Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation*, ACM SIGOPS Operating Systems Review, vol. 40, ACM, 2006, pp. 15–27.

[19] Craig Valli, Priya Rabadia, and Andrew Woodward, *Patterns and patter-an investigation into ssh activity using kippo honeypots*, (2013).

[20] Athina Provataki and Vasilios Katos, *Differential malware forensics*, Digital Investigation **10** (2013), no. 4, 311–322.

[21] Daniel Ramsbrock, Robin Berthier, and Michel Cukier, *Profiling attacker behavior following ssh compromises*, 37th Annual IEEE/IFIP international conference on dependable systems and networks (DSN'07), IEEE, 2007, pp. 119–124.

[22] Bradley S Rubin and Donald Cheung, *Computer security education and research: handle with care*, IEEE security & privacy **4** (2006), no. 6, 56–59.

[23] Bill McCarty, *The honeynet arms race*, IEEE Security & Privacy **99** (2003), no. 6, 79–82.

[24] Osama Hayatle, Hadi Otrok, and Amr Youssef, *A markov decision process model for high interaction honeypots*, Information Security Journal: A Global Perspective **22** (2013), no. 4, 159–170.

[25] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula, *Characterization of attacks collected from the deployment of web service honeypot*, Security and Communication Networks **7** (2014), no. 2, 338–351.

[26] Katerina Goseva-Popstojanova, Goce Anastasovski, and Risto Pantev, *Using multiclass machine learning methods to classify malicious behaviors aimed at web systems*, 2012 IEEE 23rd International Symposium on Software Reliability Engineering, IEEE, 2012, pp. 81–90.

[27] Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al., *Heliza: talking dirty to the attackers*, Journal in computer virology **7** (2011), no. 3, 221–232.

[28] Adrian Pauna and Ion Bica, *Rassh-reinforced adaptive ssh honeypot*, Communications (COMM), 2014 10th International Conference on, IEEE, 2014, pp. 1–6.

[29] Kapersky, *New iot-malware grew three-fold in h1 2018*, 2018, Available from: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018 [last accessed October 2018].

[30] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow, *Iotpot: analysing the rise of iot compromises*, EMU **9** (2015), 1.

[31] Arthur Jicha, Mark Patton, and Hsinchun Chen, *Scada honeypots: An in-depth analysis of conpot*, 2016 IEEE conference on intelligence and security informatics (ISI), IEEE, 2016, pp. 196–198.

[32] Meng Wang, Javier Santillan, and Fernando Kuipers, *Thingpot: an interactive internet-of-things honeypot*, arXiv preprint arXiv:1807.04114 (2018).

[33] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang, *Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices*, Black Hat (2017).

[34] David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee, *A taxonomy of botnet structures*, Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual, IEEE, 2007, pp. 325–339.

[35] J Vestergaard, *Initial analysis of four million login attempts*, Tech. Report Technical Report, Honeypot Project, 2016, http://www.honeynet.org/node/1328. Accessed November 17, 2017.

[36] Schukat Michael Dowling, Seamus and Enda Barrett., *Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware*, Journal of Cyber Security Technology **2** (2018), no. 2, 75–91.

[37] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[38] et al T. Schaul, *Pybrain*, Journal of Machine Learning Research **11** (2010), no. Feb, 743–746.

[39] Seamus Dowling, *An adaptive honeypot using reinforcement learning implementation*, 2017, Available from: https://github.com/sosdow/RLHPot. Accessed December 12 2018.

AUTHOR BIOGRAPHY

D.2 A New Framework for Adaptive and Agile Honeypots

Received xx March 2019; | Revised xx XXX 2019; | Accepted xx XXX 2019

DOI: xxx/xxxx

WILEY ETRI Journal

ARTICLE TYPE

A New Framework for Adaptive and Agile Honeypots

Seamus Dowling*[1] | Michael Schukat[2] | Enda Barrett[2]

[1]Mayo Campus, Galway Mayo Institute of Technology, Mayo, Ireland
[2]Discipline of IT, College of Engineering & Informatics, National University of Ireland Galway, Galway, Ireland

Correspondence
*Seamus Dowling, GMIT, Mayo Campus, Castlebar, Mayo, Ireland.
Email: seamus.dowling@gmit.ie

Abstract

This article proposes a new framework for the development and deployment of honeypots for evolving malware threats. As new technological concepts appear and evolve, attack surfaces are exploited. Internet-of-things (IoT) significantly increases the attack surface available to malware developers. Previously independent devices are accessible through new hardware and software attack vectors. Taxonomies governing the development and deployment of honeypots are inadequate for evolving malware and their variants. Malware methods of propagation and compromise are highly automated and repetitious. These automated and repetitive characteristics, can be exploited using embedded reinforcement learning within a honeypot. A **H**oneypot for **A**utomated and **R**epetitive **M**alware (HARM) can be adaptive, to learn the best responses when interacting with attack sequences. HARM deployments can be agile with periodic policy evaluation so as to optimise redeployment. The necessary enhancements for adaptive, agile honeypots require a new framework to govern their development and deployment.

KEYWORDS:
Honeypots, Reinforcement Learning, Adaptive, Agile, Framework

1 | INTRODUCTION

As new technological concepts appear and evolve, cyber-attack surfaces and vectors are exploited. Every Internet facing device or service is vulnerable from the untrusted external Internet. Previously independent devices are accessible through new software protocols and physical wired and wireless vectors. IoT significantly increases the attack surface available to malware developers. Mirai [1], in 2016 compromised IoT devices and contributed to large scale DDoS attacks. To react to new changes in malware evolution, cybersecurity measures have to evolve also. Zero day attack discovery is sought whereby newly deployed malware is detected before getting established. If established, malware uses very automated and repetitive methods to propagate globally, infecting and compromising hosts and forming large scale botnets. These botnets can be used to launch enormous DDoS attacks or can further propagate and morph into new variants. Kapersky states that the number of malware modifications targeting IoT devices in the first half of 2018 was greater than all of 2017 [2]. To understand malware and their variants, Spitzner [3] in 2003 suggested capturing attack data so as to understand the motives of their developers and the behaviour of the tools. The resulting *honeypots* facilitate uncovering attack behaviour. They are associated with longitudinal deployments capturing large datasets for retrospective analysis. Early versions of honeypots had low interaction capabilities. They were simple devices simulating Internet services and detected the presence of an attacker. Mid and high interaction honeypots provided for more interaction with an attacker. To adapt to new malware methods, honeypots have evolved to utilize new technologies.

Honeypots as a dynamic real time analytical tool has limitations. Their design, configuration and operations also require careful consideration. A compromised honeypot could itself

Abbreviations: HARM, Honeypot for Automated, Repetitive Malware

inadvertently participate in further attacks therefore honeypot operations require constant monitoring. Honeypots facilitate attack interaction with scripted responses to attack command streams. If the honeypot encounters an attack command it cannot process then the attack terminates. Automated malware also employs honeypot detecting mechanisms within its code. Anti-honeypot checks cause attacks to fail when processes are exposed [4]. Once honeypot functionality has been exposed, malware such as botnets will cease the attempted compromise. Subsequent malware variants employ similar techniques to evade detection by known honeypots. This reduces the potential size of a captured dataset and subsequent analysis. With the growth of new attack surfaces and vectors for malware developers, cybersecurity measures such as honeypots will need to adapt dynamically to new threats. New methods of honeypot design and deployment are required to overcome their limitations against evolving malware. Honeypots are required to be adaptive and agile to provide better datasets for faster forensics. Reinforcement learning can be used in conjunction with honeypot operations to provide adaptability. The state action space formalism outlined in section 4 is designed to target automated and repetitive malware. Deployment strategies need to be reexamined to provide agility for new variants. A new framework is required to facilitate the evolution of honeypot development. Thus we propose the following contributions over existing work:

- A new framework for honeypots is proposed. Existing taxonomies are assessed for relevancy. Updated classes and values are created, incorporating adaptive and agile functionality into honeypot development and deployment.
- Data capture on adaptive honeypots can be used to evaluate reinforcement learning algorithms and policies. Agile honeypot deployment is informed by examining Q-learning and SARSA (state, action, reward, state, action) under a variety of policy configurations.

2 | PREVIOUS RESEARCH

2.1 | Honeypot Evolution

Since their introduction in the 1990s, honeypots have evolved to meet the changing landscape of cyber threats. Just as IoT deployments evolve, so to does the deployment of bots and malicious code targeting IoT end devices [5]. In 1992, USENIX conferences presented work on captured *crackers* activities [6]. Terminal machines were deployed to lure and monitor the activity of unauthorised users. Spitzner defined honeypot as being a 'security resource whose value lies in being probed, attacked or compromised' [3]. Scientific research into honeypots and honeynets has increased since then. *Honeyd* in 2003 was an easy to deploy, low risk honeypot [7]. It details how to deploy virtual honeypots with different IPs safely. Honeyd is considered a low interaction honeypot, gathering information on the activities of an attacker in a virtual, confined space. At this point in their evolution, Honeypots could be compromised and inadvertently partake in subsequent attacks [8]. Terminology such as low-interaction, medium interaction and high-interaction came into honeypot parlance. Low interaction honeypots (LiHP) capture base information such as IP addresses, port numbers and services. They will not permit the installation or execution of downloaded malware and are considered low risk in their implementation. High interaction honeypots (HiHP) give the attacker more scope for installing malware and exploring the operating system and file structure. This has the advantage of maintaining the interest of the attacker and capturing more information on their behaviour. HiHPs are real systems, often mirroring live production systems. The obvious disadvantage of a more interactive honeypot is the potential for compromising the Honeypot itself. Unwittingly, this could lead to access to the live production networks, or participation in subsequent attacks. New LiHPs such as Nepenthes [9] and Argos [10] used new technology to emulate honeypot functionality. Virtualisation provided a means of deploying high interaction honeypots with low risk [11], isolating attack traffic from connected hardware and networks [12]. The popularity of honeypots prompted comprehensive surveys of honeypot technology, the most relevant being the most recent [13, 14, 15]. The *European Network and Information Security Agency* is a centre of network and information security expertise for the EU, its Member States, the private sector and Europe's citizens [16]. In 2012, it produced the *Proactive Detection of Security Incidents* report which tested and evaluated over 30 existing honeypots. The report found difficulties with honeypot usage, documentation, software stability and developer support. It made recommendations for the future of honeypot development.

Emerging networking technologies have facilitated new directions in honeypot deployment. Real-time visualisation of global attacks are provided by Deutsche Telekom Honeypot Project [17]. This current honeypot development community created *T-Pot*, which is used to collectively capture and visualise attacks on multiple well known honeypots. Other substantial advances in honeypot deployment include the use of *software defined networking* (SDN) to completely abstract bare metal from honeypot deployments. Intelligent solutions

have been proposed to design and flexibility deploy next generation honeynets via SDN [18, 19, 20, 21].

2.2 | Taxonomies

Zhang [22] introduced a taxonomy to standardise the development and deployment of honeypots. It identifies security as the role or class of a honeypot with *prevention*, *detection*, *reaction* or *research* as values. As more devices connected and threats evolved on the Internet, Seifert introduced an updated taxonomy in 2006 [23]. As can be seen in table 1, six core classes were proposed, each with their own set of values. Since then technology, malware and honeypot development have evolved. The heterogeneous nature new devices coupled with emerging attack vectors impact the relevancy of this taxonomy. This is explored in section 3. Other research classifies honeypot operations and propose a *honeynet* taxonomy to gain insight into honeynet architecture [24].

TABLE 1 Seifert's taxonomy for honeypot development

Class	Value
Interaction Level	High
	Low
Data Capture	Events
	Attacks
	Intrusions
	None
Containment	Block
	Diffuse
	Slow Down
	None
Distribution Appearance	Distributed
	Standalone
Communications Interface	Network Interface
	Non-Network Interface
	Software API
Role in Multi-tiered Architecture	Client
	Server

2.3 | Malware Evolution

In the 1980's the connected world of ARPANET and personal computers (PC) was increasing. The opportunity to create malicious software was also evolving. The *Creeper* worm infected ARPANET terminals, posted a message and opened new connections to other terminals [25]. Fred Cohen coined the term *virus* in 1987, designing a computer program that could infect a computer, make a copy of itself and spread to other machines [26]. HTML facilitated the creation and expansion of the www in the early 1990's. This expansion of connected computers facilitated the spread of malicious software. Evolutionary Web X.0 paradigms created new methods and variants of malware.

It is a great irony that a botnet provided a census of connected routers on the Internet. The Carna botnet scanned the IPv4 address space to create an image of fixed line Internet connectivity [27]. This automated and repetitive program globally propagated and compromised devices, predominately routers, to measure the extent of Internet access. The original Creeper worm used automated and repetitive method to propagate and infect terminals. These characteristics are shared with the various types of malware in the above list. The human factor involves designing, coding and launching the malware. Infected machines may communicate with the command and control (C&C) which is also controlled by a human *botmaster*. Human naivety contributes to infection as end users unwittingly enable content. But predominately, global infection uses automation and repetition methods. Botnets provide a mechanism for global propagation of cyber attack infection and control. They are defined as large networks of compromised machines used to carry out further attacks [28]. There are many real time systems employed to detect an prevent malware. Firewalls, IDS, IPS, anti-virus, access lists are some of the commonly used and wholly accepted measures implemented to negate the impact of malware. Malware can exploit vulnerabilities at all layers of the Open Systems Interconnection (OSI) model. From physical IoT wireless infrastructure being vulnerable to *war-driving* [29], to application services such as SQL and HTML being targeted [30]. Coupled with this is the vast array of potential attack vectors available to malware developers. Every wired and wireless communications protocol becomes a potential entry point. Bruteforce and dictionary attacks gain entry for subsequent exploitative software. From a security perspective the benefit of deploying honeypots is knowing that all activity captured is malicious.

2.4 | Malware Capture on Honeypots

Honeypots actively seek to interact with cyber attacks by simulating vulnerable Internet services and devices. It is their raison d'être. This results in honeypot datasets rich in repetitive attacks of automated sequences [31]. The analytical value associated with datasets of this type is longitudinal, providing information on attack patterns such as spatial and temporal [32]. Internet facing honeypots are themselves vulnerable. Cognisant of their existence, honeypot detection tools and evasion techniques were designed into malware [33]. For example, Honeypot Hunter identifies honeypot functionality

by creating false services and observing their execution [34]. Successful execution of these services identifies the connected device as a honeypot. Anti-detection techniques identified the presence of virtual infrastructure by executing simple kernel commands [35]. Conficker and Spybot scan for the presence of virtual machines (VM) and terminates or modifies its attack methodology accordingly [36]. As a consequence of this, honeypot effectiveness is compromised. The quality of the resultant dataset is reduced. Inadvertent termination of interactions truncates the attack sequences captured by the honeypot.

There exists a vast collection of LiHPs, MiHPs and HiHPs simulating server and client services, available on multiple attack vectors [14]. To coordinate this diversity, a versatile virtual honeynet framework focusing on the management of automatic honeypot deployment has been proposed [37]. Predominately LiHPs and MiHPs are simulations providing scripted responses. They are not at risk of being compromised and provide basic information for analysis such as IP addresses and timestamps. These can be deployed quickly and require little maintenance. For better analysis, HiHPs are required. They are real systems, actively engaging with the attacks. These honeypots gather extra information on attack code, C&C communication and downloaded files. They require time to configure correctly, deploy and maintain. Malware developers use obfuscation to avoid detection of installed software and C&C communications [38]. When released, the Mirai botnet spawned multiple variants with similar attack methods. A sample of the Mirai bot is shown in table 2 and is relevant for discussion in section 5. The type of malware captured will therefore depend upon the interaction level, attack vectors and supposed services configured on the honeypot. The latest survey found that most of the researchers tend to pose the first three presented questions, which refer to the attack source, attack target and the frequency [14]. This requires significant deployment periods to capture the necessary data to perform the longitudinal analysis.

TABLE 2 Mirai bot sample [39]

Sequence	Bot Command
1	/gweerwe323f
2	sudo /bin/sh
3	/bin/busybox
4	/gweerwe323f
5	mount
6	/gweerwe323f
7	echo -e '\x47\x72\x6f\x70' >//.nippon
8	cat //.nippon
- -	- - - - - - - - - - - - - - - - - - - -
38	/gweerwe323f
39	cat /bin/echo
40	cd/
41	wget http:// <RedactedIP>/bins/usb_bus.x86 -O - >usb_bus ; chmod 777 usb_bus
42	chmod 777 usb_bus
43	./usb_bus
44	/gweerwe323f

3 | CONCEPTUALISED FRAMEWORK

The classes from Seifert's taxonomy in table 1 are evaluated for relevancy and incorporated into the new framework. Adaptive and agile honeypots can be developed using processes of 1) adaptive honeypot development, 2) time limited deployment and data capture, 3) honeypot optimisation, in a cyclic manner. The framework shown in fig. 1, informs honeypot developers and operators, expedites the capture of datasets with complete attack data and ultimately leads to improved cyber forensics. The top half of the framework uses five classes that inform the process of adaptive honeypot creation. The functionality and performance of the adaptive element is detailed in section 4. The bottom half of the framework uses one class and two processes to inform the agile functionality of the framework. The functionality and performance of the agile element is detailed in section 5.

In order to mitigate against inadvertent termination, detection and repetitive truncated datasets, it is incumbent on honeypot developers and operators to implement alternative measures. Measures that can provide the honeypot with the adaptability to learn from the attack interactions and the

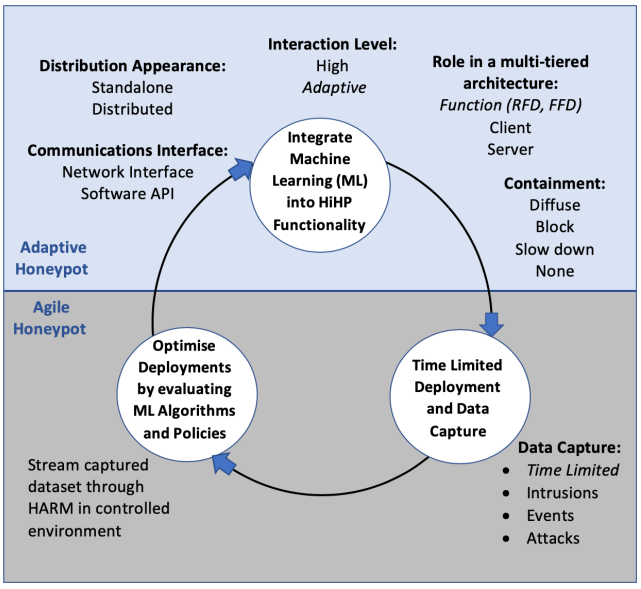


FIGURE 1 Framework for Adaptive, Agile Honeypot Development and Deployment

agility to be deployed, optimised and redeployed expeditiously. The existing taxonomy in table 1 requires revisiting to consider the relevancy of its classes and actions. The automated and repetitive characteristics of malware impact on this relevancy and therefore the classes and values need to be examined for evolving malware threats. Whilst some elements of this taxonomy still hold, updated classes and values should be considered for a new framework for adaptive, agile honeypots.

- *Interaction*: Low interaction honeypots are still available. They simulate Internet services and capture basic interactions. MiHPs and HiHPs capture more information. To contribute to meaningful cyber security research, MiHPs and HiHPs should be deployed. An issue with both is the possibility of the underlying honeypot architecture being compromised and participating in further attacks. This has been mitigated with the use of virtualisation and secure infrastructure to deploy honeypots. Machine learning functionality can learn from attack code interacting with the honeypot. This creates a new value of *adaptive* honeypots, for Interaction Level. MiHPs and HiHPs with adaptive abilities should be deployed to realise attack sequences faster.
- *Data Capture*: All automated interactions are captured by honeypots. The initial brute-force or dictionary attempts to gain access can be considered pre-compromise. They do not offer much analytical value and often skew results towards failed attempts. Post compromise data from HiHPs offers the best insights into attack design and behaviour. As demonstrated in section 2.3, malware will use obfuscated executable code and downloaded files to compromise, communicate and propagate. Longitudinal deployments leads to repetitive data capture. A new value of *Time Limited* should be added to Data Capture class to improve cyber forensics when adaptive honeypots are deployed.
- *Containment*: The values for this class are still very relevant. Virtualisation has abstracted the honeypot from the underlying architecture and therefore has somewhat protected against ethical concerns regarding operations [8]. Virtual and cloud platforms improve containment by providing mechanisms to allow or restrict specific traffic types and protocols.
- *Distribution Appearance*: Post compromise, malware will scan the environment for other potential addresses and services. New networking paradigms such as IoT will have a very different distribution appearance and will encounter evolving propagation methods. Malware could evolve to exploit mesh networks or other non-traditional models. The adaptive functionality of honeypots can respond to prolong this interaction safely in a virtualised environment.
- *Communication Interface*: The Non-Network Interface is a redundant value. Malware uses the Internet's communication protocols and software application programming interfaces (API) to propagate. The physical network interface itself can be compromised at a physical level. Wifi and IoT wireless interfaces become attack vectors to gain access to devices [29].
- *Role in a multi tiered architecture*: Malware does not discriminate post compromise. If a vulnerable device is accessible on an attack vector, malware will launch complex code structures to compromise the underlying architecture. This is irrespective of whether the honeypot advertises client or server services. An adaptive honeypot will learn the best responses to realise all commands in an attack sequence. With IoT deployments gathering pace, reduced function devices (RFD) and full function devices (FFD) create complex mesh networks requiring communication and gateways to Internet services. Traditional client and/or server models need to be expanded to include *function*.

4 | ADAPTIVE HONEYPOT

The growth in artificial intelligence (AI) and machine learning (ML) libraries heralded renewed interest in deploying honeypots. *Supervised learning* is ideally suited to retrospective analysis as the algorithm can learn from, or be trained by existing data. This learning is then used to classify new occurrences. Some examples of classifiers used on honeypot datasets are *Linear Regression* [40], *Naive Bayes*, *Support Vector Machines* (SVM), *Decision Trees*, *Random Forest* and *Nearest Neighbour* [41, 42]. Similarly *Unsupervised learning* can organise the data in different ways. Given a dataset an unsupervised model can analyse the data to find structures within [43]. In *Reinforcement learning*, an agent learns through trial and error interactions with its environment. The learning agent selects which action it should take based on previous experiences. Reinforcement Learning problems can generally be modelled using *Markov Decision Processes* (MDPs). Honeypots are examples of real-world problems where the environment is incomplete. Reinforcement learning methods facilitate solutions to MDPs where the model can often be unknown or difficult to approximate. Our proposed **H**oneypot for **A**utomated, **R**epetitive **M**alware (HARM) involves the integration of reinforcement learning into existing honeypot technology to exploit the characteristics of malware. The reinforcement learning state/action space and reward function is designed to increase the number of commands from the attack sequence.

During the reinforcement learning process the agent can select an action which exploits its current knowledge or it

can decide to use further exploration. Reinforcement learning provides parameters to help the learning environment decide on the reward and exploration values. Throughout its deployment, the honeypot is considered to be an environment augmented with reinforcement learning functionality. The honeypot *states* in this environment are examined and changed with bash scripts. The states are represented as Bash commands such as *wget* and *sudo*. The reinforcement learning agent performs actions on these states such as to *allow*, *block* or *substitute* the execution of the attack scripts. The environment issues a reward to the agent for performing that action. The repetitive nature of the automated attack sequences facilitate learning over time. Eventually the agent learns the optimum policy π^*, mapping the optimal action to be taken each time, for each state s. The learning process will converge as the honeypot is rewarded for each attack episode. This is a temporal difference method for on-policy learning and uses the transition from one state/action pair to the next state/action pair, to derive the reward. HARM uses SARSA for the implementation of on-policy reinforcement learning (eq. 1). The reward policy Q is estimated for a given state s_t and a given action a_t. The environment is explored using a random component ϵ or exploited using learned Q values. The estimated Q value is expanded with a received reward r_t plus an estimated future reward $Q(s_{t+1}, a_{t+1})$, that is discounted (γ). A learning rate parameter is also applied (α). The policy is evaluated at the end of each attack episode.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1)$$

From a functional perspective, HARM has the following elements:

- Kippo honeypot modified to pass variables to the learning agent. Depending on the action selected by the learning agent, the honeypot will allow, block or substitute attack commands.
- SARSA learning agent. This module receives the required variables from HARM and calculates $Q(s_t, a_t)$ as per equation 1. It determines the responses chosen by HARM and learns over time which actions yield the greatest amount of reward. PyBrain is a machine learning library and is used to facilitate this reinforcement learning functionality [44].

All external attack command sequences interact with the honeypot only. HARM was developed to generate rewards on 75 states and is available [45]. Fig. 2 identifies the relationship between HARM's elements, the captured data and the subsequent use and analysis of that data. Both Kippo and PyBrain are written in Python providing seamless interaction. The attack commands are parsed as the current state and passed to the PyBrain module. PyBrain selects the action that will return the maximal value for a given state. The PyBrain module is separate from the Kippo honeypot and only communicates with the honeypot. Previous approaches such as Heliza [46] and RASSH [47] have also used reinforcement learning to prolong the attack duration. However these approaches provisioned human attackers rendering these

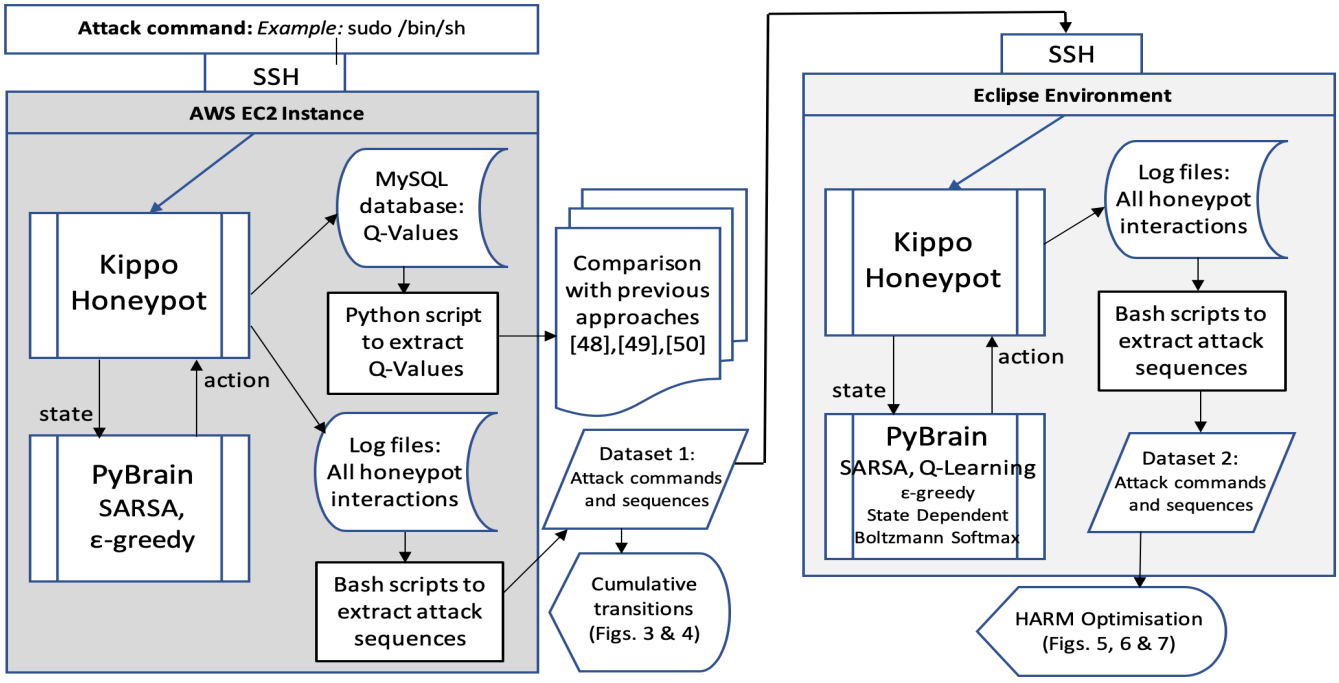


FIGURE 2 HARM's Adaptive Elements and Data Capture

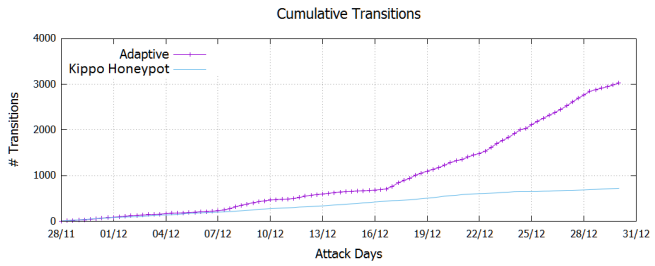


FIGURE 3 Cumulative Transition Comparison [48]

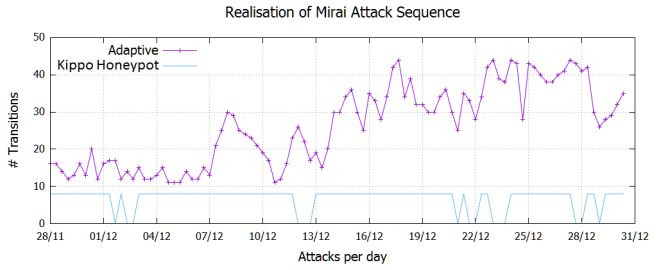


FIGURE 4 Realisation of Entire Attack Sequence

techniques inadequate with automated, repetitive malware. HARM's improved learning and reward functionality including performance comparisons with previous approaches is available [48]. Amazon Web Services (AWS) EC2 was used to facilitate an Internet facing honeypots; a normal Kippo honeypot and HARM. Kippo, PyBrain, MySQL and other dependencies were installed on the adaptive honeypot EC2 instance. It was accessible through SSH for a period of 30 days and immediately started to record repetitive malware activity. Initially it logged dictionary and bruteforce attempts. It then captured other malware traffic including a Mirai-like bot (table 2). These commands all represent interactions post-compromise. This bot became the dominant attacking tool over a 30-day period, until over 100 distinct, repetitive attacks were recorded on both honeypots. Other SSH malware interacted with the honeypot. The novel state action space formalism demonstrated improved adaptive learning and an increase in the number of commands captured by the honeypot when compared with standard honeypots [48]. Examining the malware code interactions further, it can be seen that the adaptive honeypot overcame detection techniques employed by malware [49]. These techniques are used to uncover honeypot functionality and results in the capture of truncated attack sequences. When standard honeypots encounter these techniques, it results in datasets containing truncated sequences of automated and repetitive attacks. Fig. 3 demonstrates prolonged interaction and improved data capture when compared with standard honeypots.

5 | AGILE HONEYPOT

Data capture on HARM can be used to evaluate reinforcement learning algorithms and policies. Agile honeypot deployment is informed by examining Q-learning and SARSA under a variety of policy configurations. The adaptive ability of HARM results in the capture of a dataset that contains prolonged attack interactions. Therefore the deployment period needs to be considered to ascertain if continuing with the deployment is relevant. Optimising and redeploying HARM may expedite the capture of more relevant information.

5.1 | Deployment Period

The diversity of malware as it evolves is discussed in section 2.3. To further enhance efficacy of honeypot technology, the deployment of honeypots needs to be considered. Longitudinal honeypots collect large datasets of repetitive attacks. These operate for long periods capturing repetitive, automated and incomplete attack sequences. Analysing the captured dataset from the deployed honeypots provides further information ('Dataset 1', fig. 2). The standard honeypot only ever interacted with the first 8 commands in the Mirai attack sequence. This accounts for the linear evolution of cumulative transitions in fig. 3. It never evolved to realise the entire attack sequence. It is pointed out that the repetitive, automated variant had 44 commands in the attack sequence (table 2). Fig. 4 graphs the increases in attack interactions and identifies the distinct attacks resulting in this increase.

It demonstrates that the state action space formalism for automated, repetitive malware rewarded the learning agent until all commands in the attack sequence were realised. This realisation of the entire attack sequence is affected by other malware interactions on HARM. It also shows that the adaptive honeypot realised this after 19 days (17/12/2017), making subsequent attack interactions and data collection redundant. The frequency of the Mirai variant was uniform for the duration of the deployments. The repetitive nature of the bot provide the adaptive environment with the opportunity to learn from each attack iteration. Continuing to operate the adaptive IoT honeypot was a unnecessary exercise, as redundancy began when the honeypot realised the entire attack sequence. For this short deployment of 30 days, approximately 45% of the period and dataset was redundant.

5.2 | Optimisation

The captured dataset can be used to evaluate the policies of reinforcement learning. The command sequences

in the dataset can function as an input stream into the adaptive honeypot within a controlled Eclipse environment (www.eclipse.org), modified to reflect current malware methods. HARM was deployed on the Internet using SARSA (eq. 1). Using Python, the adaptive honeypot can import and use SARSA or Q-Learning. It can also specify the ϵ-greedy explorer component and the following explorer policies:

- Epsilon greedy
  A discrete explorer that mostly executes the original policy but sometimes returns a random action.

- Boltzmann Softmax
  A discrete explorer that executes actions with a probability that depends on their action values.

- State Dependent
  A continuous explorer that disrupts the resulting action with added, distributed random noise.

The optimisation process examines the performance of HARM, configured with varying combinations of learning algorithms and explorer policies, against the malware variant in table 2. ϵ-greedy was streamed three times with settings of 0.05, 0.1 and 0.15. Therefore for SARSA, the dataset was streamed five times; Once each for Boltzmann Softmax and State Dependent, three times for ϵ-greedy. The process was then repeated for Q-Learning. The dataset from section 4 was streamed ten times through the adaptive honeypot in a controlled Eclipse environment. The first sequence (five times) with the learner set to SARSA, the second sequence with the learner set to Q-Learning. The honeypot captured the interactions, actions and number of commands in the Mirai variant attack sequence that represent transitions from s to s_{t+1}. These transitions were collated and HARM was reset after each streaming. Fig. 5 and fig. 6 demonstrate the performance of the explorer policies for SARSA and Q-Learning respectively. It can be seen that SARSA and State Dependent realised the entire attack sequence after 38 iterations (fig.5). For Q-Learning when combined with Boltzmann Softmax, it took 31 iterations.

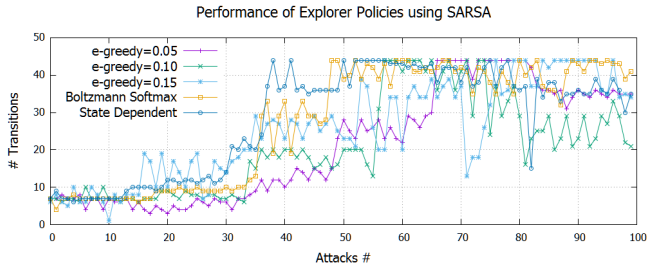


FIGURE 5 Performance of Explorer Policies using SARSA

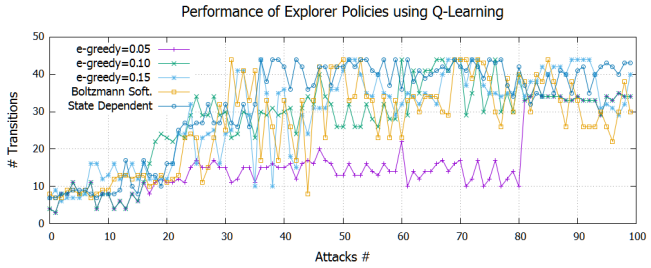


FIGURE 6 Performance of Explorer Policies using Q-Learning

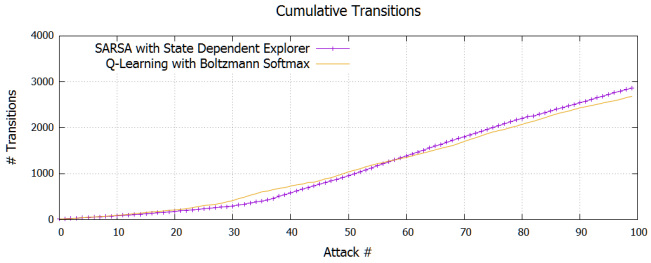


FIGURE 7 Comparison of SARSA/State Dependent with Q-Learning/Boltzmann Softmax

Two further controlled experiments were performed with the dataset. The best performing combinations, namely *SARSA/State Dependent* and *Q-Learning/Boltzmann Softmax* were configured on separate honeypots. These combinations performed best against one particular malware variant. The entire dataset was streamed through both combinations and the cumulative number of transitions were collated. Fig. 7 shows that the SARSA/State Dependent combination performed approximately 6% better than the Q-Learning/Boltzmann Softmax combination. This experiment reinforces the requirement for agility in honeypot deployment. Although Q-Learning/Boltzmann Softmax realised the Mirai variant attack sequence faster (31 v 38 attack iterations), SARSA/State Dependent accumulated more transitions on the entire dataset, beginning at attack 59 (fig. 7). Continuous monitoring and analysis of honeypot datasets can inform new combinations for redeployment.

6 | CONCLUSIONS

Cyber forensics needs to evolve with malware methods. Technological advancements provide new attack vectors to exploit and tools such as honeypots need to quickly adapt, to contribute to exploit detection. This paper proposes a new framework that uses machine learning to create adaptive honeypots and deployment methods to make them agile. It demonstrates adaptability and agility on a live honeypot dataset captured on a SSH attack vector. A number of surveys have presented

honeypot technology targeting malware on other attack vectors. The development and operations of these are informed by older taxonomies, designed to produce large datasets over longitudinal deployments. This framework can be applied to honeypots on other attack vectors. To be relevant for cyber forensics, honeypot technology needs an updated development framework that 1) is cognisant of malware evolution, 2) can use new technology in honeypot development to adapt to this evolution and 3) can ensure agility in deployment and optimisation. Honeypot datasets suffer from repetition. Adaptive, agile honeypots can exploit the characteristics of automation and repetition to expedite the exposition of malware attack methods. It has been stated that the targeted Mirai bot was the dominant attacking tool. Other malware was very infrequent for the deployment duration. Short adaptive deployments coupled with optimisation and redeployment can improve the discovery of new malware variants.

Author contributions

Financial disclosure

None reported.

Conflict of interest

The authors declare no potential conflict of interests.

References

1. Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas, *Ddos in the iot: Mirai and other botnets*, Computer **50** (2017), no. 7, 80–84.
2. Kapersky, *New iot-malware grew three-fold in h1 2018*, 2018, Available from: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018 [last accessed October 2018].
3. Lance Spitzner, *Honeypots: Catching the insider threat*, Computer Security Applications Conference, 2003. Proceedings. 19th Annual, IEEE, 2003, pp. 170–179.
4. Michel Oosterhof, *Not capturing any mirai samples*, 2017, Available from: https://github.com/micheloosterhof/cowrie/issues/411 [last accessed February 2018].
5. Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow, *Iotpot: analysing the rise of iot compromises*, EMU **9** (2015), 1.
6. Steven M Bellovin, *Packets found on an internet*, ACM SIGCOMM Computer Communication Review **23** (1993), no. 3, 26–31.
7. Niels Provos, *Honeyd-a virtual honeypot daemon*, 10th DFN-CERT Workshop, Hamburg, Germany, vol. 2, 2003, p. 4.
8. Bill McCarty, *The honeynet arms race*, IEEE Security & Privacy **99** (2003), no. 6, 79–82.
9. Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling, *The nepenthes platform: An efficient approach to collect malware*, International Workshop on Recent Advances in Intrusion Detection, Springer, 2006, pp. 165–184.
10. Georgios Portokalidis, Asia Slowinska, and Herbert Bos, *Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation*, ACM SIGOPS Operating Systems Review, vol. 40, ACM, 2006, pp. 15–27.
11. Xuxian Jiang, Xinyuan Wang, and Dongyan Xu, *Stealthy malware detection and monitoring through vmm-based out-of-the-box semantic view reconstruction*, ACM Transactions on Information and System Security (TISSEC) **13** (2010), no. 2, 12.
12. Iyad Kuwatly, Malek Sraj, Zaid Al Masri, and Hassan Artail, *A dynamic honeypot design for intrusion detection*, Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on, IEEE, 2004, pp. 95–104.
13. Navneet Kambow and Lavleen Kaur Passi, *Honeypots: The need of network security*, International Journal of Computer Science and Information Technologies **5** (2014), no. 5, 60986101.
14. Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder, *A survey on honeypot software and data analysis*, arXiv preprint arXiv:1608.06249 (2016).
15. Wenjun Fan, Zhihui Du, David Fernández, and Victor A Villagra, *Enabling an anatomic view to investigate honeypot systems: A survey*, IEEE Systems Journal (2017), no. 99, 1–14.
16. T Grudziecki, P Jacewicz, Ł JUSZCZYK, P Kijewski, and P Pawliński, *Proactive detection of security incidents*, Honeypots. ENISA (2012).
17. Deutsche Telekom, *Dtag community honeypot project*, 2018, Available from: http://dtag-dev-sec.github.io/ [last accessed October 2018].
18. Sukwha Kyung, Wonkyu Han, Naveen Tiwari, Vaibhav Hemant Dixit, Lakshmi Srinivas, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn, *Honeyproxy: Design and implementation of next-generation honeynet via sdn*, Communications and Network Security (CNS), 2017 IEEE Conference on, IEEE, 2017, pp. 1–9.
19. Wonkyu Han, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn, *Honeymix: Toward sdn-based intelligent honeynet*, Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization, ACM, 2016, pp. 1–6.
20. Wenjun Fan and David Fernández, *A novel sdn based stealthy tcp connection handover mechanism for hybrid honeypot systems*, 2017 IEEE Conference on Network Softwarization (NetSoft), IEEE, 2017, pp. 1–9.
21. Wenjun Fan, Zhihui Du, Max Smith-Creasey, and David Fernández, *Honeydoc: An efficient honeypot architecture enabling all-round design*, IEEE Journal on Selected Areas in Communications **37** (2019), no. 3, 683–697.
22. Feng Zhang, Shijie Zhou, Zhiguang Qin, and Jinde Liu, *Honeypot: a supplemented active defense system for network security*, Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on, IEEE, 2003, pp. 231–235.
23. Christian Seifert, Ian Welch, and Peter Komisarczuk, *Taxonomy of honeypots*, (2006).
24. Wenjun Fan, Zhihui Du, and David Fernández, *Taxonomy of honeynet solutions*, SAI Intelligent Systems Conference (IntelliSys), 2015, IEEE, 2015, pp. 1002–1009.

25. John F Shoch and Jon A Hupp, *The âĂIJwormâĂİ programsâĂŤearly experience with a distributed computation*, Communications of the ACM **25** (1982), no. 3, 172–180.
26. Fred Cohen, *Computer viruses*, Computers & security **6** (1987), no. 1, 22–35.
27. Erwan Le Malécot and Daisuke Inoue, *The carna botnet through the lens of a network telescope*, Foundations and practice of security, Springer, 2014, pp. 426–441.
28. David Dagon, Guofei Gu, Christopher P Lee, and Wenke Lee, *A taxonomy of botnet structures*, Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual, IEEE, 2007, pp. 325–339.
29. Joshua Wright, *Killerbee: practical zigbee exploitation framework*, 11th ToorCon conference, San Diego, 2009.
30. Banyatsang Mphago, Ontiretse Bagwasi, B Phofuetsile, and H Hlomani, *Deception in dynamic web application honeypots: case of glastopf*, Proceedings of the International Conference on Security and Management (SAM), The Steering Committee of The World Congress in Computer Science, Computer âĂę, 2015, p. 104.
31. Seamus Dowling, Michael Schukat, and Hugh Melvin, *A zigbee honeypot to assess iot cyberattack behaviour*, Signals and Systems Conference (ISSC), 2017 28th Irish, IEEE, 2017, pp. 1–6.
32. Yu-Zhong Chen, Zi-Gang Huang, Shouhuai Xu, and Ying-Cheng Lai, *Spatiotemporal patterns and predictability of cyberattacks*, PloS one **10** (2015), no. 5, e0124472.
33. Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C Zou, *Honeypot detection in advanced botnet attacks*, International Journal of Information and Computer Security **4** (2010), no. 1, 30–51.
34. Neal Krawetz, *Anti-honeypot technology*, IEEE Security & Privacy **2** (2004), no. 1, 76–79.
35. Thorsten Holz and Frederic Raynal, *Detecting honeypots and other suspicious environments*, Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC, IEEE, 2005, pp. 29–36.
36. Sheharbano Khattak, Naurin Rasheed Ramay, Kamran Riaz Khan, Affan A Syed, and Syed Ali Khayam, *A taxonomy of botnet behavior, detection, and defense*, IEEE communications surveys & tutorials **16** (2014), no. 2, 898–924.
37. Wenjun Fan, David Fernández, and Zhihui Du, *Versatile virtual honeynet management framework*, IET Information Security **11** (2016), no. 1, 38–45.
38. Ilsun You and Kangbin Yim, *Malware obfuscation techniques: A brief survey*, Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on, IEEE, 2010, pp. 297–300.
39. Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al., *Understanding the mirai botnet*, 26th {USENIX} Security Symposium ({USENIX} Security 17), 2017, pp. 1093–1110.
40. Eric Alata, Marc Dacier, Yves Deswarte, M Kaaâniche, Kostya Kortchinsky, Vincent Nicomette, Van-Hau Pham, and Fabien Pouget, *Collection and analysis of attack data based on honeypots deployed on the internet*, Quality of Protection, Springer, 2006, pp. 79–91.
41. Frank Vanhoenshoven, Gonzalo Nápoles, Rafael Falcon, Koen Vanhoof, and Mario Köppen, *Detecting malicious urls using machine learning techniques*, Computational Intelligence (SSCI), 2016 IEEE Symposium Series on, IEEE, 2016, pp. 1–8.
42. Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa, and Baijian Yang, *Predicting network attack patterns in sdn using machine learning approach*, Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on, IEEE, 2016, pp. 167–172.
43. Philippe Owezarski, *Unsupervised classification and characterization of honeypot attacks*, Network and Service Management (CNSM), 2014 10th International Conference on, IEEE, 2014, pp. 10–18.
44. Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas RĆ1/4ckstieĆ, and JĆ1/4rgen Schmidhuber, *Pybrain*, Journal of Machine Learning Research **11** (2010), no. Feb, 743–746.
45. Seamus Dowling, *An adaptive honeypot using reinforcement learning implementation*, 2017, Available from: https://github.com/sosdow/RLHPot [last accessed December 2018].
46. Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al., *Heliza: talking dirty to the attackers*, Journal in computer virology **7** (2011), no. 3, 221–232.
47. Adrian Pauna and Ion Bica, *Rassh-reinforced adaptive ssh honeypot*, Communications (COMM), 2014 10th International Conference on, IEEE, 2014, pp. 1–6.
48. Seamus Dowling, Michael Schukat, and Enda Barrett, *Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware*, Journal of Cyber Security Technology **2** (2018), no. 2, 75–91.
49. Seamus Dowling, Michael Schukat, and Enda Barrett., *Using reinforcement learning to conceal honeypot functionality*, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2018, pp. 341–355.

Bibliography

[1] Steven M Bellovin. "Packets found on an internet". In: *ACM SIGCOMM Computer Communication Review* 23.3 (1993), pp. 26–31.

[2] Paul Anderson. *Web 2.0 and beyond: Principles and technologies*. Chapman and Hall/CRC, 2016.

[3] Constantinos Kolias et al. "DDoS in the IoT: Mirai and other botnets". In: *Computer* 50.7 (2017), pp. 80–84.

[4] Kapersky. *New IoT-Malware grew three-fold in H1 2018*. 2018. URL: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018 (visited on 10/09/2018).

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

[6] Niels Provos. "Honeyd-a virtual honeypot daemon". In: *10th DFN-CERT Workshop, Hamburg, Germany*. Vol. 2. 2003, p. 4.

[7] Yin Minn Pa Pa et al. "IoTPOT: analysing the rise of IoT compromises". In: *EMU* 9 (2015), p. 1.

[8] Neal Krawetz. "Anti-honeypot technology". In: *IEEE Security & Privacy* 2.1 (2004), pp. 76–79.

[9] Manos Antonakakis et al. "Understanding the mirai botnet". In: *USENIX Security Symposium*. 2017, pp. 1092–1110.

[10] Seamus Dowling, Michael Schukat, and Hugh Melvin. "A ZigBee honeypot to assess IoT cyberattack behaviour". In: *Signals and Systems Conference (ISSC), 2017 28th Irish*. IEEE. 2017, pp. 1–6.

[11] Christian Seifert, Ian Welch, and Peter Komisarczuk. *Taxonomy of honeypots*. 2006. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.5339 (visited on 07/04/2014).

[12] George Strawn. "Masterminds of the Arpanet". In: *IT Professional* 16.3 (2014), pp. 66–68.

[13] nternational Telecommunication Union. *ITU Statistics*. 2018. URL: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx (visited on 10/15/2018).

[14] World Data Bank. *World Bank Open Data*. 2018. URL: https://data.worldbank.org/ (visited on 10/15/2018).

[15] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. "The Internet of Things—A survey of topics and trends". In: *Information Systems Frontiers* 17.2 (2015), pp. 261–274.

[16] Kevin Ashton et al. "That 'internet of things' thing". In: *RFID journal* 22.7 (2009), pp. 97–114.

[17] EWT Ngai et al. "RFID research: An academic literature review (1995–2005) and future research directions". In: *International Journal of Production Economics* 112.2 (2008), pp. 510–520.

[18] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. "The survey on near field communication". In: *Sensors* 15.6 (2015), pp. 13348–13405.

[19] Brent A Miller and Chatschik Bisdikian. *Bluetooth revealed: the insider's guide to an open specification for global wireless communication*. Prentice Hall PTR, 2001.

[20] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[21] Alexander Gluhak et al. "A survey on facilities for experimental internet of things research". In: *IEEE Communications Magazine* 49.11 (2011), pp. 58–67.

[22] P Scully. "The Top 10 IoT Segments in 2018—Based on 1,600 Real IoT Projects". In: *IoT Analytics: Market Insights for the Internet of Things* (2018).

[23] JeongGil Ko et al. "Connecting low-power and lossy networks to the internet". In: *IEEE Communications Magazine* 49.4 (2011).

[24] IEEE. *IEEE 802.15.4 Standard*. 2018. URL: https://standards.ieee.org/content/ieee-standards/en/standard/802_15_4-2015.html (visited on 10/15/2018).

[25] Daniele Miorandi et al. "Internet of things: Vision, applications and research challenges". In: *Ad hoc networks* 10.7 (2012), pp. 1497–1516.

[26] Luca Mottola and Gian Pietro Picco. "Programming wireless sensor networks: Fundamental concepts and state of the art". In: *ACM Computing Surveys (CSUR)* 43.3 (2011), p. 19.

[27] Wassim Masri and Zoubir Mammeri. "Middleware for wireless sensor networks: A comparative analysis". In: *Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on*. IEEE. 2007, pp. 349–356.

[28] Paulo ACS Neves and Joel José Puga Coelho Rodrigues. "Internet protocol over wireless sensor networks, from myth to reality". In: *Journal of communications* (2010), pp. 189–196.

[29] Zhengguo Sheng et al. "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities". In: *IEEE Wireless Communications* 20.6 (2013), pp. 91–98.

[30] Huaiyuan Wang et al. "Node failure restoration in WSN via cooperative communication". In: *Control Conference (CCC), 2015 34th Chinese*. IEEE. 2015, pp. 7709–7714.

[31] Einar Mykletun, Joao Girao, and Dirk Westhoff. "Public key based cryptoschemes for data concealment in wireless sensor networks". In: *Communications, 2006. ICC'06. IEEE International Conference on*. Vol. 5. IEEE. 2006, pp. 2288–2295.

[32] Paolo Baronti et al. "Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards". In: *Computer communications* 30.7 (2007), pp. 1655–1695.

[33] Liesbet Van Zoonen. "Privacy concerns in smart cities". In: *Government Information Quarterly* 33.3 (2016), pp. 472–480.

[34] Lei Xu et al. "Information security in big data: privacy and data mining". In: *IEEE Access* 2 (2014), pp. 1149–1176.

[35] European Commission. *Internet of Things Architecture*. 2013. URL: https://cordis.europa.eu/project/rcn/95713_en.html (visited on 01/22/2017).

[36] National Institute of Standards and Technology. *International Technical Working Group on IoT-Enabled Smart City Framework*. 2018. URL: https://pages.nist.gov/smartcitiesarchitecture/ (visited on 01/22/2017).

[37] Industrial Control Systems Cyber Emergency Response Team ICS-CERT. "The Future of Smart Cities: Cyber-Physical Infrastructure Risk". In: (2015).

[38] Industrial Internet Consortium. *Industrial Internet Security Framework Technical Report*. 2016. URL: https://www.iiconsortium.org/IISF.htm (visited on 10/15/2018).

[39] Erwan Le Malécot and Daisuke Inoue. "The carna botnet through the lens of a network telescope". In: *Foundations and practice of security*. Springer, 2014, pp. 426–441.

[40] John Von Neumann. "Theory and organization of complicated automata". In: *Burks (1966)* (1949), pp. 29–87.

[41] Fred Cohen. "Computer viruses". In: *Computers & security* 6.1 (1987), pp. 22–35.

[42] John F Shoch and Jon A Hupp. "The "worm" programs—early experience with a distributed computation". In: *Communications of the ACM* 25.3 (1982), pp. 172–180.

[43] Stephen Cass. "Anatomy of malice [computer viruses]". In: *IEEE Spectrum* 38.11 (2001), pp. 56–60.

[44] David Dagon et al. "A taxonomy of botnet structures". In: *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE. 2007, pp. 325–339.

[45] Gregory Fedynyshyn, Mooi Choo Chuah, and Gang Tan. "Detection and classification of different botnet C&C channels". In: *International Conference on Autonomic and Trusted Computing*. Springer. 2011, pp. 228–242.

[46] Felix C Freiling, Thorsten Holz, and Georg Wicherski. "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks". In: *European Symposium on Research in Computer Security*. Springer. 2005, pp. 319–335.

[47] David Dagon, Cliff Changchun Zou, and Wenke Lee. "Modeling Botnet Propagation Using Time Zones." In: *NDSS*. Vol. 6. 2006, pp. 2–13.

[48] F Pouget, M Dacier, VH Pham, et al. "on the Advantages of Deploying a Large Scale Distributed Honeypot Platform". In: *Proceedings of the E-Crime and Computer Evidence Conference*. 2005.

[49] Yu-Zhong Chen et al. "Spatiotemporal patterns and predictability of cyberattacks". In: *PloS one* 10.5 (2015), e0124472.

[50] Seamus Dowling, Michael Schukat, and Hugh Melvin. "Using analysis of temporal variances within a honeypot dataset to better predict attack type probability". In: *Internet Technology and Secured Transactions (ICITST), 2017 12th International Conference for*. IEEE. 2017, pp. 349–354.

[51] Tao Yan and Qiaoyan Wen. "A trust-third-party based key management protocol for secure mobile RFID service based on the Internet of Things". In: *Knowledge Discovery and Data Mining*. Springer, 2012, pp. 201–208.

[52] Seamus Dowling, Michael Schukat, and Hugh Melvin. "Data-centric framework for adaptive smart city honeynets". In: *Smart City Symposium Prague (SCSP), 2017*. IEEE. 2017, pp. 1–7.

[53] Ala Al-Fuqaha et al. "Internet of things: A survey on enabling technologies, protocols, and applications". In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376.

[54] Open Web Application Security Project. *IoT Attack Surface Areas*. 2014. URL: https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project#tab=IoT_Attack_Surface_Areas (visited on 09/10/2016).

[55] Lance Spitzner. *Honeypots: tracking hackers*. Vol. 1. Addison-Wesley Reading, 2003.

[56] Bill McCarty. "The honeynet arms race". In: *IEEE Security & Privacy* 99.6 (2003), pp. 79–82.

[57] Paul Baecher et al. "The nepenthes platform: An efficient approach to collect malware". In: *International Workshop on Recent Advances in Intrusion Detection*. Springer. 2006, pp. 165–184.

[58] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. "Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation". In: *ACM SIGOPS Operating Systems Review*. Vol. 40. 4. ACM. 2006, pp. 15–27.

[59] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. "Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction". In: *ACM Transactions on Information and System Security (TISSEC)* 13.2 (2010), p. 12.

[60] Iyad Kuwatly et al. "A dynamic honeypot design for intrusion detection". In: *Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on*. IEEE. 2004, pp. 95–104.

[61] Renuka B Prasad et al. "Design and Efficient Deployment of Honeypot and Dynamic Rule Based Live Network Intrusion Collaborative System". In: *International Journal of Network Security & Its Applications* 3.2 (2011).

[62] David Watson and Jamie Riden. "The honeynet project: Data collection tools, infrastructure, archives and analysis". In: *Information Security Threats Data Collection and Sharing, 2008. WISTDCS'08. WOMBAT Workshop on*. IEEE. 2008, pp. 24–30.

[63] T Grudziecki et al. "Proactive detection of security incidents". In: *ENISA report (A. Belasovs, Ed.)* (2012).

[64] Fabien Pouget, Marc Dacier, and Hervé Debar. "White paper: honeypot, honeynet, honeytoken: terminological issues". In: *Rapport technique EURECOM* 1275 (2003).

[65] Noor Al-Gharabally et al. "Wireless honeypots: survey and assessment". In: *Proceedings of the 2009 conference on Information Science, Technology and Applications*. ACM. 2009, pp. 45–52.

[66] Abhishek Mairh et al. "Honeypot in network security: a survey". In: *Proceedings of the 2011 international conference on communication, computing & security*. ACM. 2011, pp. 600–605.

[67] Abdulrazzaq Almutairi, David Parish, and Raphael Phan. "Survey of high interaction honeypot tools: Merits and shortcomings". In: *Proceedings of the 13th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting, PGNet2012. PGNet*. 2012.

[68] Matthew L Bringer, Christopher A Chelmecki, and Hiroshi Fujinoki. "A survey: Recent advances and future trends in honeypot research". In: *International Journal of Computer Network and Information Security* 4.10 (2012), p. 63.

[69] Navneet Kambow and Lavleen Kaur Passi. "Honeypots: The need of network security". In: *International Journal of Computer Science and Information Technologies* 5.5 (2014), p. 60986101.

[70] Marcin Nawrocki et al. "A survey on honeypot software and data analysis". In: *arXiv preprint arXiv:1608.06249* (2016).

[71] Deutsche Telekom. *DTAG Community Honeypot Project*. 2018. URL: http://dtag-dev-sec.github.io/ (visited on 10/23/2018).

[72] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2.

[73] Sukwha Kyung et al. "HoneyProxy: Design and implementation of next-generation honeynet via SDN". In: *Communications and Network Security (CNS), 2017 IEEE Conference on*. IEEE. 2017, pp. 1–9.

[74] Wonkyu Han et al. "Honeymix: Toward sdn-based intelligent honeynet". In: *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*. ACM. 2016, pp. 1–6.

[75] Feng Zhang et al. "Honeypot: a supplemented active defense system for network security". In: *Parallel and Distributed Computing, Applications and Technologies, 2003. PDCAT'2003. Proceedings of the Fourth International Conference on*. IEEE. 2003, pp. 231–235.

[76] Lance Spitzner. *Honeytokens: The Other Honeypot*. 2003. URL: https://www.symantec.com/connect/articles/honeytokens-other-honeypots (visited on 02/17/2014).

[77] Craig Valli, Priya Rabadia, and Andrew Woodward. "Patterns and patter-an investigation into SSH activity using kippo honeypots". In: *SRI Security Research Institute, Edith Cowan University, Perth, Western Australia* (2013).

[78] Olivier Thonnard and Marc Dacier. "A framework for attack patterns' discovery in honeynet data". In: *digital investigation* 5 (2008), S128–S139.

[79] Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al. "Heliza: talking dirty to the attackers". In: *Journal in computer virology* 7.3 (2011), pp. 221–232.

[80] Wenjun Fan, Zhihui Du, and David Fernández. "Taxonomy of honeynet solutions". In: *2015 SAI Intelligent Systems Conference (IntelliSys)*. IEEE. 2015, pp. 1002–1009.

[81] Tongbo Luo et al. "Iotcandyjar: Towards an intelligent-interaction honeypot for iot devices". In: *Black Hat* (2017).

[82] Michel Oosterhof. *Kippo Honeypot*. 2016. URL: https://github.com/desaster/kippo (visited on 05/16/2014).

[83] Ioannis Koniaris, Georgios Papadimitriou, and Petros Nicopolitidis. "Analysis and visualization of SSH attacks using honeypots". In: *EUROCON, 2013 IEEE*. IEEE. 2013, pp. 65–72.

[84] Gartner. *6.4 Billion Connected Things Will Be in Use in 2016*. 2016. URL: https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016. (visited on 07/27/2016).

[85] Srinivasa Prasanna and Srinivasa Rao. "An overview of wireless sensor networks applications and security". In: *International Journal of Soft Computing and Engineering (IJSCE), ISSN* (2012), pp. 2231–2307.

[86] ZigBee. *Low-power, low-cost, low-complexity networking for the Internet of Things*. 2015. URL: https://www.zigbee.org/zigbee-for-developers/network-specifications/ (visited on 06/15/2015).

[87] Sven Schindler et al. "IPv6 network attack detection with HoneydV6". In: *International Conference on E-Business and Telecommunications*. Springer. 2013, pp. 252–269.

[88] Google. *Google IPv6 Statistics*. 2018. URL: https://www.google.com/intl/en/ipv6/statistics.html (visited on 10/30/2018).

[89] Jürgen Markert and Michael Massoth. "Honeypot effectiveness in different categories of attacks on wireless sensor networks". In: *Database and Expert Systems Applications (DEXA), 2014 25th International Workshop on*. IEEE. 2014, pp. 331–335.

[90] Arthur Jicha, Mark Patton, and Hsinchun Chen. "SCADA honeypots: An in-depth analysis of Conpot". In: *2016 IEEE conference on intelligence and security informatics (ISI)*. 2016.

[91] Seamus Dowling. *Dataset demonstrating automation and repetition*. 2018. URL: https://github.com/sosdow/RepetitiveDataset (visited on 06/14/2018).

[92] Meng Wang, Javier Santillan, and Fernando Kuipers. "ThingPot: an interactive Internet-of-Things honeypot". In: *arXiv preprint arXiv:1807.04114* (2018).

[93] Daniel Ramsbrock, Robin Berthier, and Michel Cukier. "Profiling attacker behavior following SSH compromises". In: *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*. IEEE. 2007, pp. 119–124.

[94] Thorsten Holz and Frederic Raynal. "Detecting honeypots and other suspicious environments". In: *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*. IEEE. 2005, pp. 29–36.

[95] Bradley S Rubin and Donald Cheung. "Computer security education and research: Handle with care". In: *IEEE security & privacy* 4.6 (2006).

[96] Gérard Wagener, Alexandre Dulaunoy, Thomas Engel, et al. "Self adaptive high interaction honeypots driven by game theory". In: *Symposium on Self-Stabilizing Systems*. Springer. 2009, pp. 741–755.

[97] Tom Schaul et al. "PyBrain". In: *Journal of Machine Learning Research* 11.Feb (2010), pp. 743–746.

[98] Adrian Pauna and Ion Bica. "RASSH-Reinforced adaptive SSH honeypot". In: *Communications (COMM), 2014 10th International Conference on*. IEEE. 2014, pp. 1–6.

[99] Adrian Pauna and Victor Valeriu Patriciu. "CASSHH–case adaptive SSH honeypot". In: *International Conference on Security in Computer Networks and Distributed Systems*. Springer. 2014, pp. 322–333.

[100] David B Leake. *Case-Based Reasoning: Experiences, lessons and future directions*. MIT press, 1996.

[101] Juan José Bello-Tomás, Pedro A González-Calero, and Belén Díaz-Agudo. "Jcolibri: An object-oriented framework for building cbr systems". In: *European Conference on Case-Based Reasoning*. Springer. 2004, pp. 32–46.

[102] Osama Hayatle, Hadi Otrok, and Amr Youssef. "A Markov decision process model for high interaction honeypots". In: *Information Security Journal: A Global Perspective* 22.4 (2013), pp. 159–170.

[103] Athina Provataki and Vasilios Katos. "Differential malware forensics". In: *Digital Investigation* 10.4 (2013), pp. 311–322.

[104] Fabien Pouget, Marc Dacier, et al. "Honeypot-based forensics". In: *AusCERT Asia Pacific Information Technology Security Conference*. 2004.

[105] Eric Alata et al. "Collection and analysis of attack data based on honeypots deployed on the Internet". In: *Quality of Protection*. Springer, 2006, pp. 79–91.

[106] Christian Seifert, Ian Welch, and Peter Komisarczuk. "Identification of malicious web pages with static heuristics". In: *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*. IEEE. 2008, pp. 91–96.

[107] Katerina Goseva-Popstojanova, Goce Anastasovski, and Risto Pantev. "Using multiclass machine learning methods to classify malicious behaviors aimed at web systems". In: *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*. IEEE. 2012, pp. 81–90.

[108] Wira Zanoramy Ansiry Zakaria and Miss Laiha Mat Kiah. "A review on artificial intelligence techniques for developing intelligent honeypot". In: *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*. Vol. 2. IEEE. 2012, pp. 696–701.

[109] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula. "Characterization of attacks collected from the deployment of Web service honeypot". In: *Security and Communication Networks* 7.2 (2014), pp. 338–351.

[110] Mark Hall et al. "The WEKA data mining software: an update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[111] Philippe Owezarski. "Unsupervised classification and characterization of honeypot attacks". In: *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE. 2014, pp. 10–18.

[112] Robin Berthier et al. "Nfsight: netflow-based network awareness tool". In: *Proceedings of LISA'10: 24th Large Installation System Administration Conference*. 2010, p. 119.

[113] Saurav Nanda et al. "Predicting network attack patterns in SDN using machine learning approach". In: *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE. 2016, pp. 167–172.

[114] Marist College. *Marist Longtail Honeypot Project*. 2018. URL: http://longtail.it.marist.edu/honey/ (visited on 10/25/2018).

[115] Frank Vanhoenshoven et al. "Detecting malicious URLs using machine learning techniques". In: *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE. 2016, pp. 1–8.

[116] Gokul Kannan Sadasivam, Chittaranjan Hota, and Bhojan Anand. "Detection of Severe SSH Attacks Using Honeypot Servers and Machine Learning Techniques". In: *Software Networking* 2018.1 (2018), pp. 79–100.

[117] Gokul Kannan Sadasivam and Chittaranjan Hota. "Scalable honeypot architecture for identifying malicious network activities". In: *Emerging Information Technology and Engineering Solutions (EITES), 2015 International Conference on*. IEEE. 2015, pp. 27–31.

[118] World Data Bank. *Urban population (% of total)*. 2018. URL: https://data.worldbank.org/indicator/SP.URB.TOTL.IN.ZS (visited on 12/11/2018).

[119] Ping Wang et al. "Honeypot detection in advanced botnet attacks". In: *International Journal of Information and Computer Security* 4.1 (2010), pp. 30–51.

[120] Niels Provos et al. "A Virtual Honeypot Framework." In: *USENIX Security Symposium*. Vol. 173. 2004, pp. 1–14.

[121] Michel Oosterhof. *Not capturing any Mirai samples*. 2017. URL: https://github.com/micheloosterhof/cowrie/issues/411 (visited on 02/02/2018).

[122] Bontchev. *SSH Mirai-like bot*. 2017. URL: https://pastebin.com/NdUbbL8H (visited on 11/28/2017).

[123] Sheharbano Khattak et al. "A taxonomy of botnet behavior, detection, and defense". In: *IEEE communications surveys & tutorials* 16.2 (2014), pp. 898–924.

[124] Bjorn Stelte and Gabi Dreo Rodosek. "Thwarting attacks on ZigBee-Removal of the KillerBee stinger". In: *2013 9th International Conference on Network and Service Management (CNSM)*. IEEE. 2013, pp. 219–226.

[125] Jie Yang et al. "Detection and localization of multiple spoofing attackers in wireless networks". In: *IEEE Transactions on Parallel and Distributed systems* 24.1 (2013), pp. 44–58.

[126] Anshuman Biswas et al. "A lightweight defence against the packet in packet attack in ZigBee networks". In: *Wireless Days (WD), 2012 IFIP*. IEEE. 2012, pp. 1–3.

[127] David R Raymond and Scott F Midkiff. "Denial-of-service in wireless sensor networks: Attacks and defenses". In: *IEEE Pervasive Computing* 1 (2008), pp. 74–81.

[128] Joshua Wright. "Killerbee: practical zigbee exploitation framework". In: *11th ToorCon conference, San Diego*. 2009.

[129] Eyal Ronen et al. "IoT goes nuclear: Creating a ZigBee chain reaction". In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017, pp. 195–212.

[130] Pierluigi Paganini. *ZigBee-sniffing drone used to map online Internet of Things*. 2015. URL: https://securityaffairs.co/wordpress/39143/security/drone-internet-of-things.html (visited on 11/10/2016).

[131] Nick Statt. *How an army of vulnerable gadgets took down the web today*. 2016. URL: https://www.theverge.com/2016/10/21/13362354/dyn-dns-ddos-attack-cause-outage-status-explained (visited on 11/10/2016).

[132] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[133] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[134] Johnny Vestergaard. *Initial analysis of four million login attempts*. 2016. URL: http://www.honeynet.org/node/1328 (visited on 11/17/2017).

[135] Gonzalo Navarro. "A guided tour to approximate string matching". In: *ACM computing surveys (CSUR)* 33.1 (2001), pp. 31–88.

[136] Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber. "State-dependent exploration for policy gradient methods". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2008, pp. 234–249.

[137] Seamus Dowling, Michael Schukat, and Enda Barrett. "Improving adaptive honeypot functionality with efficient reinforcement learning parameters for automated malware". In: *Journal of Cyber Security Technology* 2.2 (2018), pp. 75–91.

[138] Seamus Dowling. *An adaptive honeypot using reinforcement learning implementation*. 2017. URL: https://github.com/sosdow/RLHPot (visited on 12/19/2017).

[139] Song Bing. *Your questions answered about Mirai Botnet*. 2017. URL: https://blog.apnic.net/2017/03/21/questions-answered-mirai-botnet/ (visited on 01/17/2019).

[140] Ilsun You and Kangbin Yim. "Malware obfuscation techniques: A brief survey". In: *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*. IEEE. 2010, pp. 297–300.

[141] Matthew Fuller. *Kippo Honeypot on Amazon EC2 Instance Free Tier*. 2012. URL: http://blog.matthewdfuller.com/2012/10/kippo-honeypot-on-amazon-ec2-instance.html (visited on 05/14/2014).

[142] Matthew Fuller. *How To Set Up an Artillery Honeypot on an Ubuntu VPS*. 2013. URL: https://www.digitalocean.com/community/tutorials/how-to-set-up-an-artillery-honeypot-on-an-ubuntu-vps (visited on 10/10/2015).

[143] Seamus Dowling, Michael Schukat, and Enda Barrett. "Using Reinforcement Learning to Conceal Honeypot Functionality". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, pp. 341–355.

[144] Tarun Yadav and Arvind Mallari Rao. "Technical aspects of cyber kill chain". In: *International Symposium on Security in Computing and Communication*. Springer. 2015, pp. 438–452.

[145] Mitre. *ATT&CK Matrix for Enterprise*. 2019. URL: https://attack.mitre.org/ (visited on 09/14/2019).