Technical Report

UCAM-CL-TR-944
ISSN 1476-2986

Number 944

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Honeypots in the age of universal attacks and the Internet of Things

Alexander Vetterl

February 2020

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

https://www.cl.cam.ac.uk/

This technical report is based on a dissertation submitted November 2019 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Churchill College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

https://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Honeypots in the age of universal attacks and the Internet of Things

Alexander Vetterl

Summary

Today's Internet connects billions of physical devices. These devices are often immature and insecure, and share common vulnerabilities. The predominant form of attacks relies on recent advances in Internet-wide scanning and device discovery. The speed at which (vulnerable) devices can be discovered, and the device monoculture, mean that a single exploit, potentially trivial, can affect millions of devices across brands and continents.

In an attempt to detect and profile the growing threat of autonomous and Internet-scale attacks against the Internet of Things, we revisit honeypots, resources that appear to be legitimate systems. We show that this endeavour was previously limited by a fundamentally flawed generation of honeypots and associated misconceptions.

We show with two one-year-long studies that the display of warning messages has no deterrent effect in an attacked computer system. Previous research assumed that they would measure individual behaviour, but we find that the number of human attackers is orders of magnitude lower than previously assumed.

Turning to the current generation of low- and medium-interaction honeypots, we demonstrate that their architecture is fatally flawed. The use of off-the-shelf libraries to provide the transport layer means that the protocols are implemented subtly differently from the systems being impersonated. We developed a generic technique which can find any such honeypot at Internet scale with just one packet for an established TCP connection.

We then applied our technique and conducted several Internet-wide scans over a one-year period. By logging in to two SSH honeypots and sending specific commands, we not only revealed their configuration and patch status, but also found that many of them were not up to date. As we were the first to knowingly authenticate to honeypots, we provide a detailed legal analysis and an extended ethical justification for our research to show why we did not infringe computer-misuse laws.

Lastly, we present honware, a honeypot framework for rapid implementation and deployment of high-interaction honeypots. Honware automatically processes a standard firmware image and can emulate a wide range of devices without any access to the manufacturers' hardware. We believe that honware is a major contribution towards re-balancing the economics of attackers and defenders by reducing the period in which attackers can exploit vulnerabilities at Internet scale in a world of ubiquitous networked 'things'.

Acknowledgements

First, and foremost, I am deeply grateful to Richard Clayton for his tireless enthusiasm and encouragement. No matter what else was happening, he always had time and provided me with ample support. Without his guidance and mentorship I would never have been able to complete this thesis.

I am also immensely grateful to Ross Anderson for all the freedom he gave me in pursuing my research interests and for his continuous support. Under his supervision, I was able to grow as a researcher and person through discussions and conversations, which have also helped me to see the bigger picture.

I am indebted to Alastair Beresford for his invaluable advice at crucial times and his sincere interest in my work. I thank Christian Rossow for generously agreeing to examine my dissertation and providing meticulous feedback.

I would like to thank the members of the Cybercrime Centre, past and present, for their support including Alice Hutchings, Daniel Thomas, Sergio Pastrana, Ben Collier, Yi Ting Chua, Maria Bada and Ildikó Pete.

I would like to express my gratitude to the German Academic Exchange Service (DAAD) and the Computer Laboratory for generously providing funding for my PhD.

I would also like to thank my friends for making my time here in Cambridge enjoyable as well as intellectually stimulating, especially Kaspars Karlsons, Diana Popescu, Tiago Azevedo, Marco Caballero, Marija Pajevic, Khaled Baqer, Christian O'Connell, Florian Ströhl, Ilia Shumailov, Mansoor Ahmed-Rengers and Michael Dodson.

Last but not least, I would like to thank my wife, Ann-Kathrin, for the joy she has brought into my life and for believing in me at every step of the way. This dissertation is dedicated to her.

Contents

Chapter 1

Introduction

The Internet is transforming from a global network of computers to a heterogeneous mixture of everyday devices ('things'). The Internet of Things (IoT) market is growing rapidly, creating pressure on manufacturers to ship feature-complete devices as soon as possible, avoiding time-consuming maturity and security considerations. The emphasis on automation and the nature of the devices themselves mean that vulnerabilities and exploits not affecting device functionality are likely to remain unnoticed by owners. This lack of incentive and visibility results in a growing number of vulnerable and compromised devices. This is most obvious in the growth of IoT-based botnets, a collection of Internet-connected devices controlled by a common type of malware. More than 20 years after the first Distributed Denial of Service (DDoS) attack was documented, botnet-based DDoS attacks remain ubiquitous. It is not uncommon to see botnets with more than a hundred thousand devices and accounts of DDoS attacks reaching several hundred gigabits per second (Gbps).

At the same time, we are starting to connect critical infrastructure, including the water and power grid, financial, and communications systems to ease their operation and maintenance. These systems are particularly valuable as targets for disruption and ransom, in addition to the possibility of their use in standard botnets. Telecommunication providers are also making significant progress in deploying 5G-enabled networks. Thus it is reasonable to expect a further significant up-take of Internet-connected devices, especially in areas where fixed-line broadband is currently unavailable. While the wider adoption of IPv6 is progressing slowly, it has become feasible to scan the whole IPv4 address space for vulnerable devices with modest investment. This gives attackers a crucial advantage: once an exploit is found for one technology, device, or specific implementation, attackers can easily find devices with that vulnerability and instantly scale the exploit.

This economic shift in favour of the attackers means that it is economic for attackers to exploit any vulnerability independent of its spread, threat and potential impact, as the only substantive cost is detecting the vulnerability itself. Previously, attackers might find a vulnerability, but it would take considerable effort to capitalise on it since vulnerable devices may not be permanently connected to an accessible network or otherwise easy to detect. Also, the speed with which exploits can be deployed increases drastically in the absence of the need for a complex and time-consuming search for victims. It is therefore harder to defend against such universal attacks and react with patches within a reasonable time frame.

This means that vulnerable Customer Premise Equipment (CPE), a generic term for home networking devices that are connected to the telecommunication provider's networks, and IoT devices are a constant threat to the Internet itself and society at

large. Many regulators have started to look for opportunities to improve IoT security by means of stricter standardisation and certification regulations. However, responsibilities are dispersed, coordination is non-trivial, and technology is advancing rapidly. Thus the early detection of new attacks on these devices is increasingly important in contemporary approaches to improving Internet security. Traditionally, most of the efforts in network security have focused on controlling network flow. Firewalls are designed to block certain types of incoming traffic and intrusion detection systems use attack signatures to detect suspected intrusions. To capture and better understand these attacks instead of simply blocking them, honeypots, resources that appear to be legitimate systems, need to be revisited. Once an attack has been captured by a honeypot, the attack vector can be analysed, and future attacks can be prevented.

For the last decade, honeypot research has received limited attention, with efforts mainly focused on monitoring the activity of human attackers. But as attack patterns change, honeypots should emulate what is actually targeted. The Mirai botnet, the first large botnet to recruit a variety of IoT devices, uses an automated and pseudo-random scanning process to find and infect new targets. This means that the provision of a realistic environment for humans to interact with is of declining importance. Instead, honeypots should focus on correctly implementing low-level protocol interactions with automated scripts. While it is also feasible to build a custom honeypot to monitor Mirai, for example by sending appropriate strings in response to the automated scanning tools, this approach only allows the monitoring of attacks similar to those we already know.

So we need to create *better* honeypots for CPE and IoT devices, and to this end we investigate state-of-the-art honeypots and demonstrate their severe limitations. First, we investigate the effects of previous researchers' assumption that their honeypots were interacting with individuals, while in reality they were capturing the activities of bots. We show that warning banners in an attacked computer system have no deterrent effect and, as a result, previous research not only drew wrong conclusions, but also gave wrong policy advice. Second, we present a 'class break' by demonstrating that the architecture of the current generation of honeypots is fundamentally flawed. Their architectures use off-the-shelf libraries to implement protocols, making it trivial to fingerprint a large number of different honeypots because there are numerous differences on-the-wire between entirely standards-compliant implementations. Third, having identified thousands of such honeypots on the Internet, we found that many are not kept up-to-date and demonstrate that honeypot operators largely believe that they are dealing with naïve human adversaries.

Fourth, we developed a virtual honeypot framework that is capable of rapid deployment to emulate devices and thereby capture the real attacks along with malware samples. The framework uses the original code base without incurring significant overhead. In particular, different patch level and software versions can be efficiently emulated allowing the monitoring of the many attackers who are now going after a wide range of devices with automated tools. Our framework will help to detect when criminals have identified vulnerabilities in (IoT) devices that might otherwise be exploited for considerable periods of time without anyone noticing.

For too long in this arms race, the defenders' costs were magnitudes higher than the attackers. Our work significantly contributes to re-balancing this cost asymmetry by reducing the period in which attackers can exploit vulnerabilities at Internet scale – in a world of universal attacks an endeavour previously limited by a fatally flawed generation of honeypots.

1.1 Chapter outline

- **Chapter 2** establishes the background of CPE and IoT devices and discusses common characteristics and vulnerabilities. It outlines prevalent threats and targeted protocols and discusses the evolution of honeypots. We particularly focus on honeypots for IoT and the ease with which they can be fingerprinted in the age of universal attacks.
- **Chapter 3** presents a detailed study in which we use honeypots to show that warning banners in an attacked computer system have no deterrent effects and that honeypot data has been misinterpreted by previous research. We find that honeypots mainly capture automated activities and that the number of human trespassers is orders of magnitude lower than previously assumed. (We did attempt to understand the motivation and psychological traits of the few criminals interacting with our honeypots, but none of the trespassers was willing to take part in our survey.)
- **Chapter 4** highlights the weaknesses of the current generation of low- and medium-interaction honeypots. The use of standard libraries to implement the protocols means that we can easily fingerprint any such honeypot solely based on their protocol implementations. We present a 'class break' for systematically generating probes that can find these honeypots with just one or two packets at Internet scale. Our technique means that even if honeypot developers fix single protocol fingerprints, thousands more fingerprints can be found. Instead of focusing on the monitoring of human activity and the provision of a realistic environment for humans to interact with, a new generation of honeypots should put more emphasis on the verisimilitude in the lower levels of the networking stack.
- **Chapter 5** provides a detailed analysis of honeypot deployments and deployment practices. Based on the architectural flaws described in Chapter 4, we conduct several Internet-wide scans in a one-year period. To determine which particular versions of honeypots are being run on the Internet, we logged in to them and issued a small set of shell commands. As we are the first to knowingly authenticate to honeypots, we give an extended legal analysis explaining why we did not infringe computer-misuse laws. Our analysis shows that many systems were not up-to-date. Since developers track the commands that adversaries use and continually add new features to make honeypots more covert, not updating a honeypot increases the chances that it may be fingerprinted (using traditional techniques) and thus limits its value in detecting new attack vectors. We further found that many honeypot operators are relying on standardised deployment scripts and rarely change set-up options or the honeypots' authentication configuration.
- **In Chapter 6**, we present honware, a virtual honeypot framework that processes a standard firmware image of CPE and IoT devices, customises their filesystem and runs them with a special pre-built Linux kernel. We show that our framework is better than previous approaches in providing networking functionality and in emulating the devices' firmware applications. It emulates devices much more accurately than traditional honeypots and is not susceptible to trivial fingerprinting based on timing attacks. Honware is a major contribution towards re-balancing the economics of attackers and defenders by reducing the period in which attackers can exploit zero days at Internet scale.
- **Chapter 7** summaries our work and outlines future research directions towards better honeypots in the age of universal attacks and ubiquitous networked devices.

1.2 Publications and contributions

Parts of the research described in this thesis have been published in peer reviewed conferences and workshops. I list below the contributions of my co-authors to the publications that are included in this thesis. I further list acknowledgements of funding and feedback.

1. Alexander Vetterl and Richard Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at Internet scale", *in 12th USENIX Workshop on Offensive Technologies (WOOT '18)*, Baltimore, MD: USENIX Association, 2018
   a) This publication forms the basis of Chapter 4 of this dissertation.
   b) Richard Clayton provided valuable feedback to direct this research away from manual techniques to fingerprint honeypots to a generic technique to systematically identify protocol deviations.
   c) I am grateful to Ross Anderson, Alastair R. Beresford, Alice Hutchings, Daniel R. Thomas and Sergio Pastrana for helpful comments on the WOOT paper.
   d) This work was supported by the EPSRC [grant number EP/M020320/1] and a Premium Research Studentship from the Department of Computer Science and Technology, University of Cambridge.
2. Alexander Vetterl, Richard Clayton and Ian Walden, "Counting outdated honeypots: Legal and useful", *in 4th International Workshop on Traffic Measurements for Cybersecurity (WTMC '19)*, San Francisco, CA: IEEE, 2019
   a) This publication forms the basis of Chapter 5 of this dissertation.
   b) Richard Clayton and Ian Walden (Solicitor of the Senior Courts of England and Wales, and Professor of Information and Communications Law, Queen Mary, University of London) provided the legal analysis and the statutory texts for unauthorized access.
   c) I am grateful to Alastair R. Beresford, Alice Hutchings and Daniel R. Thomas for helpful comments on the WTMC paper.
   d) This work was supported by the EPSRC [grant number EP/M020320/1].
3. Alexander Vetterl and Richard Clayton, "Honware: A virtual honeypot framework for capturing CPE and IoT zero days", *in 14th APWG Symposium on Electronic Crime Research (eCrime '19)*, Pittsburgh, PA: IEEE, 2019
   a) This publication forms the basis of Chapter 6 of this dissertation.
   b) Richard Clayton provided ideas on how the kernel can be modified and helped to better understand the attacks observed.
   c) I am grateful to Ross Anderson, Alastair R. Beresford, Alice Hutchings, Robert N. M. Watson, Daniel R. Thomas and Michael Dodson for helpful comments and discussions on the paper.
   d) This work was supported by the EPSRC [grant number EP/M020320/1] and a Premium Research Studentship from the Department of Computer Science and Technology, University of Cambridge.
4. Alexander Vetterl, Alice Hutchings, Richard Clayton, Michel Cukier, David Modic, Bertrand Sobesto, "Revisited: Restrictive deterrent effects of a warning banner in an attacked computer system", *preparing for submission*, n/a, 2019

a) Bertrand Sobesto and Michel Cukier provided and maintained the infrastructure to host the honeypots.

b) I am grateful to Ross Anderson and Daniel R. Thomas for helpful comments and discussions on the paper.

c) This work was supported by the EPSRC [grant number EP/M020320/1].

1.3 Statement on research ethics

All the experiments reported in this dissertation followed our institution's ethical research policy. This research has been approved by the Ethics Committee of the University of Cambridge, Department of Computer Science and Technology (ref: 417 and 450). Where appropriate, we discuss the particular ethical considerations and procedures in each Chapter individually.

Chapter 2

Background

In this Chapter, we introduce common characteristics of CPE and IoT devices. In particular, we discuss the system monoculture which is the result of a dominant operating system (Linux) and the market concentration of available chipsets. We focus on the devices' protocols and outline common vulnerabilities that resulted in various large-scale attacks in recent years.

We then discuss recent advances in device discovery and Internet-wide scanning, which are the basis of contemporary universal attacks – attacks that use potentially trivial exploits, but can affect millions of devices across manufacturers and geographical regions. Attackers can instantly deploy exploits without needing a complex and time-consuming search for victims. We discuss how compromised devices are used to conduct further criminal activities, such as denial of service attacks, permanently destroying devices or using them to proxy malicious traffic such as spam.

Honeypots are used in an attempt to detect such attacks at an early stage, so we discuss the different types of honeypots and how they have evolved over time. We discuss the emerging trend towards virtualisation and honeypots for the IoT, moving away from the traditional approach of honeypots attempting to provide a realistic looking shell for humans to interact with. In particular, we focus on fingerprinting, a technique whereby an attacker determines whether an apparently vulnerable machine is real or a decoy. This is a major shortcoming of honeypots in the age of universal attacks. Designing honeypots to be fingerprint resistant is not straightforward, since the increasing complexity of protocols and applications means that honeypots have to emulate ever more complex systems.

2.1 CPE and IoT devices

'Customer-premises equipment' (CPE) is a generic term for xDSL modems, routers, switches and other home networking devices that are connected to telecommunication providers' networks. Since 2001, the number of fixed-broadband subscriptions has been steadily increasing and in 2018, 14.1% of the world's population accessed the Internet through a broadband line.[1] An increasing number of everyday physical 'things' are connected to the Internet, and are commonly referred to as the Internet of Things (IoT). In particular home appliances are continually sensing and collecting vast amounts of data, so the term IoT includes devices such as security surveillance systems, thermostats, digital video recorders and smart TVs. In fact, the first Internet-connected 'thing' was a vending machine at

[1] https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

Carnegie Mellon University.[2] As of 2019, more than 25 billion IoT devices are connected to the Internet and the number of IoT devices is expected to surpass 75 billion by 2025.[3]

Besides the ubiquitousness of IoT devices, CPE is particularly important within the IoT ecosystem as they serve as an entry point into a home or office network. Furthermore, CPE ensures inter-device connectivity, manages quality of service, handles phone calls via VoIP and is a choke point for all traffic to any device. Thus, once it is compromised, attackers may be able to connect to further devices at that location and so particular emphasis should be put on its security.

2.1.1 Common characteristics

Chipset monoculture: For many consumer devices, the hardware comes in the form of a system on a chip (SoC) that integrates essential features such as CPU, memory and storage. Most of today's SoCs for CPE are manufactured by Broadcom, Qualcomm, Atheros and MediaTek and are then used by device manufacturers such as Netgear, TP-Link, D-Link and Linksys. We can further split SoCs based on their Instruction Set Architecture (ISA): the ARM and MIPS architectures are the most used ISAs for networked devices [32, 26].

Furthermore, chipset manufacturers not only provide the raw silicon, but also write drivers and applications for their devices. This lack of diversity means that once a vulnerability is found in parts which the chipset manufactures controls, the exploit will potentially affect multiple devices and vendors in various geographical regions. Furthermore, the number of versions of each chipset vendor's code makes it increasingly hard to provide updates in a timely manner as multiple players are involved in the release management process. Thus the CPE and IoT environment is having similar problems as other technology platforms such as the Android operating system: a fragmented market[4] with no clear responsibilities and no incentives for manufactures to provide regular updates [1].

Operating systems derived from Linux: The vast majority of operating systems for consumer devices are derived from Linux, typically supplemented with custom kernel modules and drivers to provide device-specific functionality [160]. An increasing number of routers are also capable of running alternative Linux-based firmware like Tomato, OpenWrt or DD-WRT. DD-WRT can be customised extensively and as of 2019, is supported by more than 80 vendors.[5] In contrast, Tomato and OpenWrt support fewer devices and claim to be more user-friendly. OpenWrt is also the only distribution which does not include non-free binaries and so fewer devices are supported. All three distributions are fairly lightweight; for example, OpenWrt recommends devices which have just 8 Mbyte of flash storage and 64 Mbyte of RAM.[6]

Tailored specifically for the IoT, Baccelli et al. developed RIOT OS, a stripped down version of Linux designed for low-power IoT devices [8]. Similar, Ubuntu offers Ubuntu Core, a minimal Linux image with 10 years of guaranteed security updates and support for the ARM and x86 architectures.[7] Lastly, Microsoft announced in 2018 that Azure Sphere, Microsoft's IoT operating system, will be based on Linux.[8]

[2] https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/
[3] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/
[4] https://developer.android.com/about/dashboards
[5] https://wiki.dd-wrt.com/wiki/index.php/Supported_Devices
[6] https://openwrt.org/toh/recommended_routers
[7] https://ubuntu.com/core
[8] https://azure.microsoft.com/en-us/services/azure-sphere/

Firmware for CPEs is almost invariably available for download and an increasing number of manufacturers also publish the source code of their firmware as GPL-Code, in particular to support ongoing projects such as OpenWrt [106].

Lack of (automated) patching capabilities: In 2014, the SANS Institute conducted a study[9] among businesses and found that 31% of participants considered the difficulty of patching 'things' as the biggest threat to the IoT over the next five years, with another 25% being concerned that IoT devices would end up distributing malware.

Hampton and Szewczyk analysed various routers including routers from the brands D-Link, Netgear and TP-LINK, and found that the firmware images use libraries and executables with an average age between three and ten years, and are often bundled together with outdated kernel versions [63] . Similar, Costin et al. analysed 32 356 firmware images for CPE and IoT devices and found that many firmware images included outdated software and kernel versions, including the use of kernels and binaries that were built up to ten years after their initial release [32].

Nakajima et al. analysed the patch management of vendors for Consumer IoT devices and found that five of six manufacturers have been releasing patches in a timely manner [99]. However, even if a patch is available on the companies' websites, customers do often not update their devices. Asking 2000 end-users, Canonical found that only 31% of consumers update their IoT devices as soon as an update becomes available, 40% of consumers have never updated their device since they purchased it, and 8% did not know what firmware was [24]. Nakajima et al. further found that vendors often implicitly announce End-of-Support for devices in certain regional areas while the same device is still supported in other regions, effectively shortening the support period [99].

To this end and to better educate consumers, Morgner et al. [95] proposed a scheme in which devices are given mandatory security update labels. In their user study they found that (timely) update guarantees for certain product categories are more important than product attributes for making purchase decisions. Similar, Emami-Naeini et al. [47] found that privacy and security attributes of products are amongst the most important in purchase decisions, but were ranked after product features and price. Thus they concluded that consumers are not prepared to pay a premium for security, especially because security remains hard to measure, particularly for consumer devices.

2.1.2 Protocols and services

In Chapter 4 and 5, we will use protocol deviations to fingerprint honeypots at Internet scale, and in Chapter 6, we will develop a honeypot framework for CPE and IoT devices. Here we briefly introduce the general characteristics of the most used communication protocols in CPE and IoT devices, which are emulated in honeypots.

Secure Shell (SSH) is a cryptographic network protocol that transparently encrypts and decrypts network connections between server and client [10]. It is the de facto standard for secure login to remote devices. The current SSH version 2.0 has been standardised by the Internet Engineering Task Force (IETF). The transport protocol (RFC4253 [158]) specifies that SSH listens on port 22 by default. After the TCP three-way handshake, the server sends a version string to the client and the client responds with its own version string. The string is sent in the form `SSH-protoversion-softwareversion SP comments \r\n` where `SP` is a space, `\r` is a carriage return and `\n` is a line feed. After the version

[9] https://www.sans.org/reading-room/whitepapers/covert/paper/34785

number exchange, the key exchange begins by sending, amongst other things, name-lists of supported encryption algorithms, MAC algorithms and compression algorithms. After the `SSH2_MSG_KEXINIT` packet exchange, the key exchange algorithm is run.

Telnet provides a bi-directional, unencrypted communication channel for interaction with remote terminals. It was first standardised by the IETF in 1983 by RFC854 [107] and subsequently updated by RFC5198 [74] in 2008. Both client and server may negotiate various terminal options specified in a number of different RFCs [108], including line mode, echo and terminal type. In general, the negotiation starts with an `IAC` (Interpret as Command) escape character followed by one of the option codes `WILL, WON'T, DO, DON'T` and the desired terminal option. The other party then accepts or rejects the requested state using the same syntax.

Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol used for accessing resources on web servers. It was initially standardised by the IETF in 1996 with RFC1945 [15] and subsequently updated; the latest 2.0 version is defined in RFC7540 [14]. In HTTP, the client uses a variety of methods such as `GET`, `HEAD`, `DELETE` and `TRACE` to request services from remote servers, each of them defined in various RFCs. Web servers interpret the incoming requests and respond dynamically. The RFCs leave much freedom as to how servers implement the methods. For example, RFC7231 [52] states that when the server receives a request that is unrecognised or not implemented, the server 'should' respond with an 501 error code ('Not implemented'). However, this is not mandatory and may differ by implementation.

Universal Plug and Play (UPnP) enables the control, data transfer and interconnection of networked devices without manual configuration [54]. It runs on top the Internet Protocol (IP) and uses HTTP to exchange messages. By default, UPnP works on port 1900 (UDP) and initially, each connected device attempts to obtain an IP address via DHCP. Subsequently, the devices will automatically advertise their services, the device type, an unique identifier, and a URL where more information can be obtained (UPnP description). The UPnP description uses the XML syntax and is typically a URL that contains a device description and one or more service descriptions [91]. The service descriptions include actions to which the service can respond. These actions can be triggered by sending a crafted message to the URL of that service, again using the XML format and HTTP requests [54].

2.1.3 Common vulnerabilities

The rapidly growing IoT market demands shortened product life cycles and shorter time-to-market [141]. Products are often immature, insecure and because of the limited diversity of available chipsets (Section 2.1.1), may share common vulnerabilities and attack vectors [4].

Internet-connected: A wide range of CPE and IoT devices have publicly accessible services. This is mainly the result of bad default configurations and the use of UPnP to reduce the configuration burden for consumers of setting up networking for each individual device.

To ensure that IoT devices can communicate with remote services, such as a mobile application, they either send data to a central server or use peer-to-peer (P2P) connections. To establish a P2P connection with the IoT device, which is typically located behind a router, the device has to overcome Network Address Translation (NAT). To do so, UPnP-enabled devices set and change port forwarding rules in the router.

Kumar et al. published the first large-scale study to use user-initiated network scans followed by an Internet-wide scan to measure how many households have publicly accessible services [80]. Using data on 16 million households, they found that 3.4% of households expose HTTP, followed by FTP (0.8%), Telnet (0.7%) and SSH (0.8%). They further found that 67.5% of IoT devices run at least one network-based service and that UPnP (46.2%) is the most popular protocol for device discovery, followed by HTTP (45.7%) for device administration.

Weak authentication: The most used protocols to administer devices are Telnet, SSH and HTTP(S). However, many devices are shipped with default passwords that can be easily guessed or found in device manuals.

In 2010, Cuin and Stolfo performed Internet-wide scans with a default credential scanner [36]. The scanner will, after having received a response from an Internet-connected networking device, attempt to authenticate using a list of default credentials. In total, they found more than 540 000 publicly accessible devices that were configured with the factory root passwords. Similar, Costin et al. analysed 32 356 firmware images for CPE and IoT devices and found hard-coded passwords in 681 of them [32]. Their Internet-wide scan revealed hard-coded credentials for Telnet affecting at least 2k devices and hard-coded web-login credentials affecting at least 101k devices. In 2019, the European Telecommunications Standards Institute (ETSI) released a technical specification which recommends banning universal default passwords for IoT devices.[10]

Backdoors are closely linked to weak authentication. For simplicity, manufactures put default credentials or cryptographic keys in their firmware so they can connect to the device easily and debug device failures. However, these access mechanisms are often undocumented and not immediately obvious. In 2013, Heffner found a backdoor in various D-Link routers which is activated by setting the user-agent header to `xmlset_roodkcableoj28840ybtide`.[11] Doing so allowed anyone to send any HTTP request, without authentication, to the device, alter the configuration, and ultimately control it remotely. Similarly, devices of various types and brands such as routers from Zyxel[12] and Cisco[13], Digital Video Recorders (DVRs) from Raysharp[14], and security cameras from Hangzhou Xiongmai[15] were using hard-coded passwords which cannot be changed by users.

Programming errors include but are not limited to improper input validation, improper use of cryptographic protocols, improper memory initialisation and ignoring the bounds of a memory buffer. These vulnerabilities are not unique to IoT devices but are a common consequence of careless software development. However, the rapidly-growing IoT market causes manufacturers to focus on shipping feature-complete devices as soon as possible rather than making them more mature and secure. As outlined in Section 2.1.1, many devices are based on (old) Linux kernels; the associated applications are often outdated, insecure and do not conform with security best practice. Recent examples include a format string vulnerability in the Broadcom UPnP software [71], a buffer overflow in appGet.cgi for ASUS routers[16], and a flaw in the algorithms to generate unique device

[10] https://www.etsi.org/deliver/etsi_ts/103600_103699/103645/01.01.01_60/ts_103645v010101p.pdf
[11] https://nvd.nist.gov/vuln/detail/CVE-2013-6026
[12] https://nvd.nist.gov/vuln/detail/CVE-2016-10401
[13] https://www.tomshardware.com/news/cisco-backdoor-hardcoded-accounts-software,37480.html
[14] https://nvd.nist.gov/vuln/detail/CVE-2015-8286
[15] https://nvd.nist.gov/vuln/detail/CVE-2018-17919
[16] https://nvd.nist.gov/vuln/detail/CVE-2018-14712

IDs.[17] The latter vulnerability allows remote attackers to predict IDs and to establish direct connections to arbitrary devices, affecting more than 2m devices of the Chinese manufacturer Shenzhen Yunni Technology.

2.2 Universal attacks

Before we introduce the concept of honeypots and their contribution to mitigating the consequences of insecure CPE and IoT devices, we first briefly explain how vulnerable devices are discovered and then discuss recent attacks.

The existence of specialised search engines to discover CPE and IoT devices and the modern ability to perform Internet-wide scans in minutes together mean that suitable exploits can be used almost instantly and leave little time for vendors to deploy appropriate defenses. This is especially problematic because, as explained in Section 2.1.1, many devices – even though they are sold by different companies – use identical chipsets and software. Thus a single exploit can affect millions of devices across brands and continents.

Once devices are compromised at scale, they typically become part of a botnet, a collection of devices that are under the control of some malicious actor.[18] Botnets are used in various ways including malware distribution, cryptocurrency mining, proxying malicious traffic and to conduct DoS attacks [113].

2.2.1 Device discovery and Internet-wide scanning

One key metric of CPE and IoT malware is its attack potency, which is largely determined by the overall number of vulnerable devices.

With open-source tools such as Zmap [45] it is possible to perform Internet-wide scans in minutes. Zmap can perform TCP SYN scans or can be used to send UDP probes. ZMap can further be extended with various modules such as ZGrab and ZBrowse. Zgrab is an application layer scanner which can, after the initial TCP handshake, send further probes to remote servers to obtain information such as the servers' application versions. These scans are repeatedly performed for various ports, including SSH, Telnet and UPnP, and the information is then made available rapidly via specialised search engines such as Shodan[19], Censys[20] and Thingful[21].

In addition to publicly available information, the propagation of IoT malware has significantly improved with the rise of the Mirai botnet. What makes the Mirai malware unique is its implementation of a fast, stateless scanning module. By default, Mirai sends 160 TCP SYN probes per second to a pseudo-randomly chosen IPv4 address on port 23 (every tenth packet is sent to port 2323) [7]. Mirai further sets the Initial Sequence number (ISN) – normally a random 32 bit integer – the same as the destination IPv4 address. After Mirai has identified a victim, it uses up to 10 username and password combinations which are randomly chosen from a list of 62 credentials [76]. In fact, Mirai uses 61 unique combinations as the list of 62 credentials includes one duplicate (guest/12345). If a login is successful, the IP address will be sent to a report server, which will then inform a loader.

[17] https://cve.mitre.org/cgi-bin/cvename.cgi?name=2019-11219
[18] https://www.cloudflare.com/learning/ddos/what-is-a-ddos-botnet/
[19] https://www.shodan.io
[20] https://censys.io
[21] https://www.thingful.net/

The loader uses the credentials and issues a sequence of commands to download a Mirai binary for the specific architecture from a malware server. Mirai will then run, start the scanning process and kill all other processes listening on port 23 and port 22 so that competing malware cannot compromise the device.

The speed of modern Internet-wide scanning and the available open-source code of Mirai with its stateless scanning module together mean that malware spreads rapidly. As a consequence, suitable exploits can be used almost instantly at Internet scale and leave little time for vendors to deploy defenses.

2.2.2 Denial of service attacks

(Distributed) denial of service attacks (DDoS) were first widely discussed and studied after such attacks hit various big companies including Amazon, Yahoo! and CNN in 2000 [81]. However, already in 1996 a flood of SYN packets resulted in the temporarily shut down of Panix, an American-based ISP [23]. DDoS attacks aim to make computer systems unresponsive and unavailable by means of excess connections or data requests. Volumetric attacks are typically measured in Gbps as their only aim is to send as much traffic as possible to the victim to overwhelm victim bandwidth capacity. DDoS attacks may also target particularly expensive parts of applications, for example by sending specifically crafted database requests or search API calls. By attempting to respond to all the incoming data, the victim machine is overloaded and cannot handle legitimate user requests. Distributed attacks are characterised in that requests are sent by multiple machines which may be in dispersed geographical locations.

To recruit a large number of machines, attackers either target the administrative ports such as Telnet and SSH across a wide range of devices, or find a vulnerable service which can be fooled to forward data. The latter can be effective if a server's response is disproportionate to the request sent, i.e. the size of the request is significantly smaller than the response triggered (the 'amplification factor'). A particularly problematic factor is that most UDP protocols have no authentication, enabling traffic to be sent to the victim on the attacker's behalf [118].

Donno et al. provide an extensive overview of thirteen IoT malware families with DDoS capabilities from 2008 to 2016 [39]. Most of the IoT malware is available for both MIPS and ARM as the most prevalent architectures. However, only for four IoT malware families has the source code become publicly available – Linux.Hydra (2008), Zendran (2012), BASHLITE/Gafgyt (2014) and Mirai (2016).

Mirai has been primarily used to conduct DDoS attacks on various websites and Internet services such as Krebs on Security, OVH and DynDNS. To carry out the attack on OVH, 145k IoT devices were used and the resulting traffic topped out at 1TBs, according to their own telemetry. The attack on Brian Krebs was significantly smaller with a peak of 623 Gbps.[22] In total, Mirai has compromised over 600k devices.[22]

In the aftermath of Mirai, we have seen the rise of the Hajime botnet. In contrast to Mirai, Hajime also targets a variety of other ports including ports 80 and 7547 (TR-064). Herwig et al. analysed it and found that it uses a peer-to-peer infrastructure instead of centralised loaders and Command & Control servers; they estimated that it infected up to 95k devices [65]. Costin and Zaddach estimate that it controls significantly more; they found that it has infected between 130k and 300k devices [31].

[22]https://elie.net/blog/security/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/

Permanent denial of service attacks (PDoS) are another form of attack which have emerged in 2016 by the name of 'Brickerbot'. It is different from DDoS attacks as the aim is to make the device permanently unusable [76]. The author of Brickerbot claims in an interview to have destroyed over 10m devices[23] before he retired the malware in December 2017. However, there is no independent evidence that supports his claim. The malware itself uses a set of credentials to login to insecure devices, primarily via Telnet. It then issues a set of commands to rewrite the device's flash storage, often including configurations necessary for the device to boot. Thus, after the devices were rebooted or powered off, they were dysfunctional.

In 2019, Akamai found related malware called 'Silexbot' which destroyed about 4,000 devices.[24] To compromise devices, Silexbot uses a set of known credentials to login via Telnet. Once compromised, it reads /dev/random and then writes the data to any mounted flash storage.

2.2.3 Proxying malicious traffic

Proxying traffic through compromised devices is useful for attackers because the proxy makes it harder to trace back their actions, counter and/or blacklist machines, and to shut down malicious hosts. The device's functionality is often retained so that its owner is not aware that their device is used as a proxy. In one of the first studies, Steding-Jessen deployed ten honeypots at volunteers' homes to capture email spam, so the IPs appeared to be standard ADSL home connections [129]. The honeypots emulated an open SMTP relay server, and in 15 months they captured more than 500m emails from more than 216k unique IP addresses. In 2017, a new malware botnet called Linux.ProxyM appeared which specifically targets IoT devices, again with a set of pre-defined credentials, and uses them as a proxy to send spam. Costin and Zaddach estimate that the botnet consists of at least 10k devices and that each device sends about 400 emails per day [31]. In 2018, the FBI issued a warning that "cyber actors are using compromised IoT devices as proxies" to, amongst other things, send spam e-mails, maintain anonymity, obfuscate network traffic and generate click-fraud activities.[25]

2.3 Honeypots

The term 'honeypot' was first used by Spitzner in 2001 who described a honeypot as a "security resource whose value lies in being probed, attacked or compromised" [127, p. 40]. However, the concept was introduced more than ten years before in 1989 when Stoll described a fake system environment with no production value that was used solely for the purpose of monitoring incoming connections and tracking attackers [132]. Shortly afterwards, in 1990/91, Cheswick described how he created a fake environment after someone had fetched the `/etc/passwd` file via ftp in order to monitor attackers' activities [28].

[23] https://www.bleepingcomputer.com/news/security/brickerbot-author-retires-claiming-to-have-bricked-over-10-million-iot-devices/

[24] https://blogs.akamai.com/sitr/2019/06/sirt-advisory-silexbot-bricking-systems-with-known-default-login-credentials.html

[25] https://www.ic3.gov/media/2018/180802.aspx

In the early 2000s, honeypots focused on deception and on increasing the attacker's workload, exhausting the attacker's resources and increasing the sophistication required to undertake an attack [29]. Since then, they have evolved and the term encompasses a variety of different concepts and types which are outlined in Section 2.3.1. Most notably, the SANS Institute defines honeypots as "fake computer systems, set up as a 'decoy', that are used to collect data on intruders" [119, p. 2]. Also, their focus has shifted from a tool of deception and frustrating attackers, to a tool that can capture and analyse attacks, act as remote sensors and provide early warning [94]. Thus, honeypots complement traditional security tools such as firewalls, intrusion detection systems and Virtual Private Networks (VPNs). Firewalls rely on rules and anti-virus software relies on updates; both primarily protect against known vulnerabilities and attack vectors. In contrast, honeypots may detect and offer a way to detect the yet unknown attacks including 'zero-days' [125].

Han et al. categorised deception techniques in four dimensions: goal, unit, layer and deployment [64]. First, the goal of deception can be to lure attackers into a sandbox where their attacks can be studied or to mitigate the harm they do by redirecting them to a sinkhole. Second, the unit of deception refers to the actual decoy. Decoys are variable, from a simple warning banner or a full emulation of an industrial control system. Third, the layer of deception refers to the layer it's applied. Lastly, the deployment of deception refers to the actual implementation technique. Deception may come built-in or be triggered by certain actions.

2.3.1 Types of honeypots

In the last 20 years, several attempts to classify honeypots have been made. First and foremost, honeypots are distinguished according to the direction of interaction [122, 100]. Throughout this thesis, we focus on server honeypots which are decoy environments that wait for attackers passively. In contrast, client honeypots actively connect to a host to evaluate whether the remote service is malicious and does not behave as intended. Most of the client honeypots are web browsers that interact with a remote web server to determine if it tries to load malicious content. Some client-side honeypots such as PhoneyC [101] can also emulate certain vulnerabilities and monitor their abuse. With the increase of malicious cryptocurrency miners [49], client honeypots such as Thug [133], which uses the Googles V8 JavaScript Engine, have been revisited and become increasingly popular.

Second, honeypots are classified as research or production [126]. Research honeypots are typically connected to the Internet and aim to get a comprehensive understanding of the threat landscape. By contrast, production honeypots are used in organisations, typically with limited network connectivity, and aim to improve the level of security of production systems directly by absorbing attack traffic. This also means that any activity within the honeypot is considered malicious. These honeypots tend to require high maintenance, and are difficult to deploy [100].

Third, honeypots are classified by whether the level of interaction is low, medium or high; this is also commonly described as 'level of fidelity' [29, 110, 50, 38]. As with the first classification (research/product), this scheme is ambiguous as there are no clear criteria for each category. At the lowest level, these honeypots may be simple scripts that serve one specific purpose, such as logging authentication attempts or emulating specific network protocols. In contrast, high-interaction honeypots should share state with the actual systems as any deviation from the real system's behaviour reduces the level and

fidelity of interaction, for example by limiting outbound traffic [110]. Most commonly, the level of fidelity relates to the layer of emulation. Honeypots can be software artefacts, a virtualisation of a system, or exhibit full system behaviour [50]. In Figure 2.1, we not only show the evolution of server honeypots, but also classify them according to their layer of emulation. As shown, we see an emerging trend towards virtualisation and the use of physical devices. This is because the increasing complexity of protocols and applications means that honeypots have to emulate ever more complex systems.

The level of fidelity has also direct implications on maintainability and associated risks. As high-interaction honeypots allow full access to a real system, they need to be tightly monitored and maintained. They have significant value, but many people are unable to accept the risk that they may be used for DDoS attacks, to distribute malware or send email spam. This is when low- and medium-interaction honeypots have proven effective as they are easy to deploy and to maintain, while their potential for harm is minimised. They are especially useful in collecting quantitative data about large-scale attacks.

More recently, Fan [50] proposed a new taxonomy based on two categories, the features of the decoy and the purpose of the security program. In this scheme, the level of fidelity (low-/medium/high-interaction) is only one feature of the decoy, next to seven other categories: scalability, adaptability, role, physicality, deployment strategy and resource type. The purpose of the security program is classified as attack monitoring, prevention, detection, response or profiling. However, even this more detailed taxonomy does not use objective factors to distinguish, for example, between low-,medium-, high interaction honeypots or at what point the honeypot is considered scalable.

2.3.2 Evolution of honeypots

As outlined in Section 2.3.1, this thesis focuses on server honeypots. The first server honeypot was *Deception Toolkit* (DTK), published by Cohen in 1997.[26] DTK emulates a Unix system with many known vulnerabilities for various services including SSH, FTP and DNS. The DTK is purely a software honeypot and is written in C and Perl. Following the success of DTK, two commercial honeypot solutions have emerged, Specter and CyberCop String. Both emulate various network services, but their level of interaction is limited and none offers full operating system interaction. *CyberCob Sting* was also the first honeypot to introduce the concept of having multiple systems bound together in a so called honeynet [94].

The first major step forward was HoneyD [127]. HoneyD was the first widely adopted open source solution and offers two major improvements. First, it listens on any TCP port and regardless of the level of emulation for each port, it logs attackers' traffic. Second, its open source code means that HoneyD is customisable. If an attack is recognised on one port that is not yet properly emulated, the honeypot operator can change the source code to implement the needed service [127].

Following the success of HoneyD, Kreibich and Crowcroft developed honeycomb, an extension to HoneyD which analyses the honeypot traffic and automatically generates intrusion detection signatures [79]. Potemkin, the first scalable honeypot platform based on virtualisation, appeared next [148]; it re-directs traffic flows to special, virtualised machines based on Xen. To set up new VMs rapidly, they use delta virtualisation and flash cloning. They demonstrated that within a 10-minute period, over 2,100 VMs could

[26] http://www.all.net/dtk/

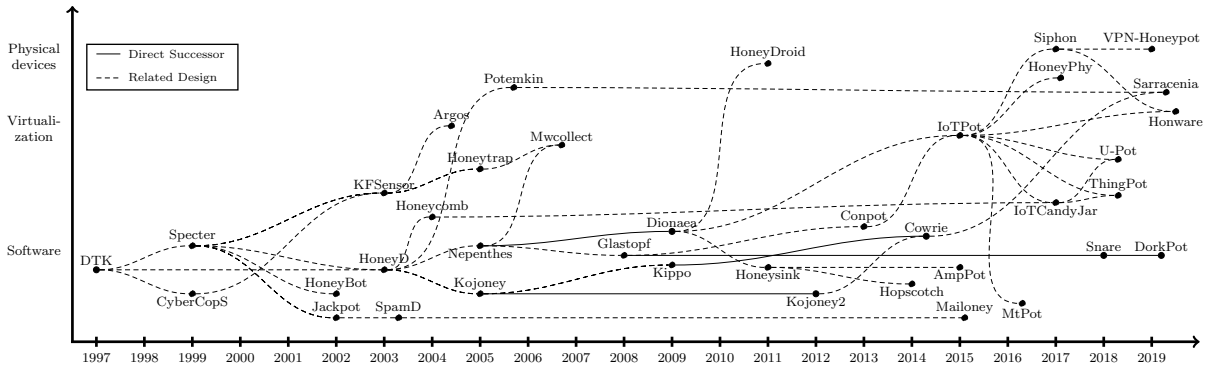


Figure 2.1: Evolution of server honeypots from 1997 to 2019. Solid lines signify a direct successor. Dashed lines signify a related design, including the choice of protocol(s), emulation of specific versus generic (vulnerable) services, and purpose (monitoring, prevention, detection, response or profiling).

be created in response to incoming packets.

Two years after HoneyD, Baecher et al. proposed Nepenthes, a new platform for rapid emulation for a wide range of vulnerabilities [9]. Their platform requires a priori knowledge of attackers and attack vectors, but was the first honeypot that specifically separates honeypot modules, i.e. the exploitation, payload analysis, de-duplication of malware URLs and downloading the respective binaries. Dionaea, the successor of Nepenthes, continued to focus on emulating vulnerabilities to capture malware samples, but supports the Session Initiation Protocol (SIP), IPv6 and TLS [25].

As protocol complexity increased, the development of honeypots changed its focus yet again. Between 2005 and 2014 a large number of protocol-specific honeypots were developed: Kojoney [75], one of the first SSH honeypots, was followed by Kippo [73] in 2009. The development of Kippo ceased in 2015 and the author now recommends a fork called Cowrie [104]. Cowrie not only supports SSH, but added support for Telnet, and is still based on the same foundation and protocol libraries of Kojoney and Kippo. Glastopf [115] was one of the first HTTP specific honeypots, followed by the successors Snare [116] and DorkPot [111]. Specific to Industrial Control systems (ICSs), Conpot has been developed to emulate a variety of protocols including Modbus, S7 and Bacnet.

As mentioned in Chapter 2.2.2, UDP-based DDoS attacks remain ubiquitous. To monitor these attacks, honeypots such as AmpPot [77] and Hopscotch [136] pretend to be vulnerable services with particularly high amplification factors. For each incoming request, Hopscotch reflects the first packets so the systems appears to be a legitimate vulnerable service. However, once the honeypot receives "more than a handful of packets with the same source IP address", it will stop forwarding attack traffic to the victim on the attacker's behalf, but capture the exact details of the attacks [136, p. 3].

A 2012 report for the European Network and Information Security Agency (ENISA) evaluated the vast majority of available honeypots, including high-interaction honeypots. The authors recommend using the medium-interaction honeypots Kippo for SSH, Glastopf for HTTP and Dionaea for the remaining protocols [59]. Honeypots that supported Telnet 'out of the box' were not widely available at the time of that study.

In 2015, Minn et al. developed one of the first generic high-interaction honeypots tailored to impersonate CPE and IoT devices. To return appropriate messages to attackers, they collected banners from the Internet for port 23. For each incoming connection, the

honeypot will return with a previously obtained conversation; if the interaction/command is unknown, it tries to run the command in a generic sandboxed environment ('IoTBOX') to infer the appropriate return string(s). IoTBOX itself uses the OpenWrt Linux operating system and QEMU, an open-source emulator that performs hardware virtualisation.

With the rise of the Mirai botnet, IoTPot has inspired many other honeypots, most notably SIPHON [60]. The platform exposes physical devices, such as networked video recorders and security cameras, on a range of IP addresses. More recently, U-Pot [62] uses the UPnP device description documents, typically a .XML file, and the gupnp library to emulate a UPnP device.

Overall, we see an emerging trend towards virtualisation, the use of physical devices, and protocol-specific honeypots. This is for two main reasons. First, as protocols become increasingly complex it is hard to develop a honeypot which can handle multiple protocols faithfully with a high level of interaction. Second, attack vectors are also increasing in their complexity. While the first honeypots were only concerned with logging brute-force attempts and file access, the increased use of smart devices had led to both vulnerabilities and their exploitation becoming ever more complex. To exploit vulnerabilities several protocol exchanges, payloads and device specific interactions may be performed.

This issue can be avoided if we use the actual physical device, or a virtualised version of it, as the original software implementation is used. However, honeypots using physical devices are limited in their scalability as there are tens of thousands of different devices to protect. Furthermore, physical devices with their original kernel and applications do not have the capability to actually log an attack, so it is only possible to intercept and monitor incoming traffic. With the use of encryption (e.g. HTTPs and SSH), this becomes an ever more complex task as it would be necessary to man-in-the-middle the traffic. But then the attacker does not interact with the vulnerable device directly, but with a different implementation that might not be vulnerable to an attack of interest. It also means that sophisticated attackers may be able to detect the MITM, perhaps from differences in protocol interactions or timing discrepancies.

2.3.3 Fingerprinting honeypots

Fingerprinting is a concern for honeypot developers and operators. Once a honeypot can be identified by attackers, its value in detecting new attack vectors and monitoring attack traffic may drastically decrease. Hosts running honeypots may be blacklisted by attackers so that their honeypots have to be moved [37]. Thus honeypots should not be easily detected. As explained in Section 2.3.1, high-interaction honeypots should ideally share state with the actual system. However, even the slightest modification to the real system, for example adding logging capabilities – which are essential for honeypots to be of any use – means that the honeypot and the real system are not identical.

Chen et al. were the first to establish a taxonomy of fingerprinting by classifying common malware anti-virtualisation techniques into hardware, environment, application and behaviour [156]. Uitto et al. categorised honeypot fingerprinting into the four dimensions temporal, operational, hardware and environment [139]. Both taxonomies are similar as the classification *behavioural* of Chen et al.'s taxonomy captures only timing differences, namely the *temporal* issues in Uitto et al.'s classification. Similarly, the category *application* captures the extent to which applications are installed and can be executed. In Chen et al.'s model, the category *operational* includes applications, but also considers

the way how applications can communicate, e.g. whether applications are allowed to send and forward traffic or even participate in attacks. Thus we will adopt the classification of Uitto et al. and focus on fingerprinting techniques specific to honeypots.

Temporal: Garfinkel et al. demonstrated that virtualisation induces anomalies, such as timing discrepancies and that these anomalies can be used to detect virtualised environments [55]. Similar, Holz and Raynal showed that the execution time of commands provides an efficient way to detect honeypots because emulation will typically result in longer execution and response time [67]. In the same vein, Mukkamala et al. demonstrate that honeypots within virtual environments respond slower to ICMP echo requests than real systems [98]. However, they also found that for Internet-connected honeypots this metric may not be useful because it depends on network load, routing and emulation technology. In Chapter 6, we evaluate if the timing of our new virtualised honeypot is indistinguishable from real devices connected to the Internet.

Operational is the extent to which the honeypot communicates. Wang et al. proposed to detect honeypots based on the assumption that honeypots cannot take part in real attacks and therefore honeypots do not allow certain applications to be run or installed [152]. To detect honeypots based on these constraints, they propose to set up sensors which act as honeypot detectors. These sensors under the botmasters' control are periodically hit to verify that the compromised machine is still actively participating in attacks. For example, in the case of proxying spam emails, emails might be send to an email address under the botmaster's control to detect honeypot sinkholes. To this end, Krawetz developed a tool called *Honeypot Hunter* which tests if emails sent through the compromised system were actually delivered [78]. Similar, Zou and Cunningham proposed fingerprinting honeypots based on their limited willingness to participate in real attacks and to execute malicious activities, such as continuously requesting content from web servers that are under the botmasters control [162].

Hardware: Honeypots often do not run on bare-metal hardware, but in some constrained environment, such as virtual machines or containers with limited resources. Inevitably, virtual machines have to create hardware devices and use specific drivers for the guest operating to function. Related to the detection of (high-interaction) honeypots are malware sandboxes. Sandboxes are used to run malware binaries in a controlled environment to gain insights how the malware operates [159].

Chen et al. analysed 6 222 malware samples and found that 95.3% of the samples have the same behaviour in a VM as running on bare-metal hardware [156]. Yokoyama et al. analysed 20 malware analysis services by submitting a custom Windows 32-bit binary with the aim of fingerprinting sandboxes based on a set of features, including display resolution, display width, RAM size, system uptime and time of last login [159]. To train their classifier and to establish 'ground truth', they used 50 Windows PCs. They found that hardware features are among the most distinctive features, as sandboxes are typically single-core and use little RAM with small disk sizes. While this is a problem for malware targeting PCs with ample resources, CPE and IoT devices themselves have limited resources and thus these features are likely to be less useful.

Environment: Environment includes the operating system, file system and the current state of the machine, such as running applications. Morishita et al. built upon our work in Chapter 4 and showed that many honeypots do not blend into the type of service they emulate and reveal themselves by a unique configuration [96]. They analysed 14 open-source honeypots and found 19 208 honeypots across 637 Autonomous Systems (ASs)

which failed to take "even the most basic precautions against detection by attackers". Similar, Sysman et al. pointed out that default configurations for a variety of honeypots, including Glastopf, Kippo and Dionaea, allow them to be fingerprinted with minimal effort [19]. For example, Glastopf serves a distinct default webpage which can be found by a basic search on search engines. We add to this body of literature in that we developed an automated technique to fingerprint honeypots based on packet-level protocol interactions, identifying honeypots at Internet scale with trivial probes (Chapter 4).

2.4 Summary

This Chapter briefly summarised the background required for the rest of the dissertation. It introduced three common characteristics of CPE and IoT devices: chipset monoculture, the use of operating systems derived from Linux and the lack of automated patching. Together with basic security failures such as weak authentication, these factors are the basis for universal attacks targeting CPE and IoT devices. The predominant attacks rely on Internet-wide scanning for device discovery. These factors mean that a single exploit can affect millions of devices across brands and continents.

We introduced honeypots as a mean of detecting such attacks quickly. We discussed how they have evolved over time to adjust to the ever-changing threat landscape. This will form the basis for Chapter 3 in which we show that warning banners have no deterrent effect and that honeypots mainly capture the behaviour of bots rather than human activities. We discussed the trend towards virtualisation and the shortcomings of the current generation of honeypots. In particular, fingerprinting was discussed in more detail to provide the necessary background for Chapter 4 which presents an automated technique to fingerprint honeypots based on packet-level protocol interactions at Internet scale.

Chapter 3

Revisiting the effect of warning messages on attackers' behaviour

The use and analysis of survey and interview data is common among criminologists and offers an avenue for better understanding crime. With the increase of cybercrime, the use of honeypot data becomes increasingly important to better understand deviant behaviour [20]. However, there have been doubts about the validity and generalisability of studies using honeypot data. In particular, concerns have been raised that honeypots do not capture individual criminals, but automated activities [134] and that "incorrect assumptions about the actors" are made as honeypots do not collect data on demographics, motivations and rationales [66, p. 740].

We are the first to empirically confirm with two independent studies that warning banners have no deterrent effect and that previous research has wrongly assumed that they would affect human behaviour. We find that the number of human trespassers is orders of magnitude lower than previously assumed, and that previous research measured the behaviour of automated scripts. These scripts are only programmed to evaluate the systems' usefulness by looking for a pre-defined set of characters such as indications of a shell prompt. In the presence of a (lengthy) warning banner, they either fail to correctly parse the unexpected message or completely ignore it.

3.1 Introduction

Accessing computer systems without authorisation, also referred to as 'system trespassing' in Criminology, is one of the most prevalent forms of cybercrime. In an attempt to deter attackers, the use of warning banners has been proposed.[1] These banners aim to convey to trespassers that the use of the system is monitored, unauthorised and subject to criminal and civil penalties. In general, deterrence theory argues that individuals do not engage in criminal activities if the punishment is severe and costly. Thus when the costs of deviant behaviour outweigh the benefits, humans are less likely to proceed with their action [57].

Examining the effects of such warning banners in an attacked computer system has received extensive attention over the last couple of years. In 2014, Maimon et al. [90] conducted two experiments in which target computers, i.e. honeypots, were set up either to display or not display a warning banner after attackers gained access to the system via SSH. The authors found that displaying warning messages that the system is under

[1] https://nvd.nist.gov/800-53/Rev4/control/AC-8

surveillance after attackers successfully logged in significantly reduces the probability that attackers will issue commands and shortens the duration of sessions.

Several studies repeated their experiment and support their findings. First, Stockman et al. [131] found that the mean duration of system trespassing incidents for incidents on systems showing a warning banners is significantly shorter (15.29 seconds) than of those without (23.45 seconds). Second, Testa et al. [134] found that system trespassers with non-administrative privileges navigate through the filesystem and change file system permissions significantly less often on computers with warning banners than on computers without. Similar, Wilson et al. [153] found evidence that the presence of a warning banner significantly reduces the probability that commands are typed in the system. Third, Jones [70] argues that both the display of a standard legal threat and an ambiguous threat that explicitly mentions that there will not be any consequences, increases the early use of *reconnaissance commands* – commands that are used to gain more information about the system such as `uname` and `update`. Their results were not statistically significant, but in the anticipated direction, i.e. displaying either warning message results in the earlier examination of the targeted computer than on the control group.

Steinmetz [130] was one of the first scholars to point out several methodological concerns in the research design of Testa et al. [134]. He criticises that in the 30 days the target computers were deployed, the computers might have been comprised by completely different criminals with the same cracking tools, each of whom could respond differently to warning banners, and the research design fails to take account of that. Second, he argues that the study is unable to distinguish between human actors and bots. Therefore, we cannot be confident in any inferences we make about user behaviour. Recently, Udhani et al. [138] used 423 days of SSH honeypot data with more than 500 million connection attempts to identify common traits of attackers which can be used to differentiate between humans and bots. Not surprisingly, they find that attackers are more likely to be a human if the number of password attempts is less than ten per minute and if less than three characters per second are typed. More importantly, they find that the vast majority of observations are the results of bots interacting with the SSH honeypot and not humans. Although Wilson et al. [153] found that showing warning banners significantly reduces the probability that commands are typed in the system, they also concluded that warning banners do not lead to a reduction in the volume and probability of repeated system trespassing. Adding to the criticisms, Bossler [20, p. 685] argues that data from honeypots may have severe limitations and biases: "the reality is that most attacks are automated" so honeypots are not measuring humans, but automated scripts.

We add to the debate [89, 20] by adapting the study from Maimon et al. [90] by including a third treatment option with 'nonsense' text of the same length as the warning message. We further implemented additional logging mechanisms, so we were able to record which (SSH) options were requested. Options include a shell, purposely not requesting a shell, and the execution of a single command. We ran our study for 365 days on the same University network that was used for Maimon et al.'s research (except for the additional treatment, the experimental set up was identical).

In a second experiment that also ran for 365 days, we deployed a CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) [147]. The CAPTCHA was displayed after login credentials have been successfully guessed, but before any message was shown. This allowed us to reliably distinguish between humans and automated scripts as the latter would fail to solve the CAPTCHA.

We could not find any support for the deterrent effects of warning banners and conclude that previous research has misinterpreted honeypot data. First, we find that displaying a warning banner in an attacked computer system is not positively associated with the hazard of first system trespassing incident termination and therefore does not result in shorter duration of first trespassing incidents. Consistent with the original work, the term 'first system trespassing incident' describes the first recorded event of unauthorised access ('trespassing incident') on each target computer. Second, a warning banner does not increase the probability of trespassing session termination and therefore does not reduce the duration of system trespassing incidents. Third, the majority of requested sessions were non-interactive. This means that the assumptions of previous research was wrong – what previous research has measured was the behaviour of automated scripts rather than the behaviour of humans. The second experiment confirmed this observation and we find that the number of humans is magnitudes lower than previously assumed. Thus, if we want to measure the effects of a warning banner, we need to carefully design experiments which solely target humans and do not capture the behaviour of automated scripts.

We further believe that inter-disciplinary research including experts from the field of information security, psychology and criminology is necessary to understand the complex nature of deception in cyber space and cybercrime more generally. It is important to understand how honeypot data is collected and to have expert knowledge about the current cybercriminal landscape, including technical capabilities, attack vectors and tools to draw reasonable conclusions. By not doing so, we risk drawing at best inconclusive or, at worst, wrong conclusions which may directly impact (deterrence) policy making in cyberspace.

3.2 Maimon et al.'s original study

To examine the effects of a warning banner on the frequency and duration of system trespassing incidents, Maimon et al. [90] used 80 public IP addresses that belong to an American university. The honeypots are high-interaction in that attackers are given full access to the compromised machines. The experimental set up is done with a combination of Sebek [109], an established tool to log user interactions, and OpenVZ hosts and containers to quickly re-deploy target computers after 30 days have passed. The experiment ran for a period of 2 months from April 1 to May 30, 2011. In this period, 86 target computers were successfully compromised, and 971 trespass incidents were logged. The authors also evaluated the effects of different system configurations, i.e. they explored whether RAM size and bandwidth capacity influence the duration or frequency of trespass incidents. For this experiment, 300 public IP addresses were used and 502 target computers were successfully compromised in a six-month period from October 4, 2011 to April 3, 2012.

In both of their experiments, most of the target computers recorded repeated trespass incidents. Looking at the effects of warning banners, they found that "a warning banner reduces the duration of system trespassing incidents [...] but that the hazard of first system trespassing incident termination is not conditioned by the computing configurations of the target computer" (p. 16). The second experiment confirmed that "a warning banner produces more deterrence and shorter duration of system trespassing incidents on target computers with a low bandwidth" (p. 18) and that RAM size has no effect on the hazard of system trespassing incident termination.

3.3 Experiment 1: The deterrent effects of warning banners

Based on the original work, we replicated Maimon et al.'s. [90] original study, but we added a second treatment option, the presentation of Lorem Ipsum text. To this end, we present the findings and results in a comparable way and where appropriate, used the identical statistical methods and terminology.

We hypothesise that the effects seen in the original work when presenting the warning banner were due to the interaction with automated scripts rather than the semantics of the warning banner or anything else. In particular, the hypothesis is that there is no difference between the effectiveness of the warning message and the Lorem Ipsum text.

These scripts do not understand the semantics of the messages, but read all responses solely to look for the shell prompt in order to evaluate the 'usefulness' of the compromised computer. This evaluation may further fail because these scripts only read responses up to a certain length, typically the length of a buffer.

This behaviour has been observed repeatedly in the past. The Mirai malware, one of the few malware types for which the source code has been leaked, is an indicative example of the challenges to evaluate the usefulness of the compromised system. Mirai automatically determines the underlying system environment with a combination of protocol exchanges, issued commands, and subsequently examining the responses [7]. It stores these responses in a buffer that can hold up to 256 bytes. If the buffer does not contain the desired response, the malware will not proceed, for example with trying passwords or downloading a binary to infect the device. To check the content of the buffer, Mirai implements a two-step process as indicated exemplary in Listing 3.1 for verifying the password prompts: initially, it looks for standard password prompts with the characters :, >, $, and # which often indicate a shell environment, identical to SSH. If one of these characters is found, it returns to the position after the prompt and continues to brute-force passwords.

The second part handles cases in which the routine could not find a standard prompt in the buffer. In this case, the function `util_memsearch` sequentially looks for "assword" in the buffer (the character P/p is skipped intentionally to ignore lower and upper case). If any 7 sequential characters of the buffer are equal to "assword", the function will return a match. However, it is important to note that the buffer is limited to 256 bytes. If it is full, Mirai drops the first 64 characters, moves the remaining content along, reads 64 additional bytes from the queue and adds them to the end of the buffer. This procedure is implemented in a loop so that all bytes of the incoming responses are consumed.

So Mirai is solely focused on finding the password prompt and simply ignores any other characters sent. This has three implications. First, any character or string in the buffer that is not of interest will be ignored. Second, their hard-coded characters and strings mean that if Mirai sees any use of these character sequences before the prompt, Mirai will wrongly believe that it has found a password prompt, return the wrong position to input the password, and be unable to infect the host. Third, any code that does not correctly drain the buffer, but for simplicity, just reads responses into the fixed length buffer, will inevitably cut off responses and thus not find the relevant information. This example demonstrates that reading inputs is not trivial and that it is often done in a way convenient for specific use-cases.

```
static int consume_pass_prompt(struct scanner_connection *conn)
{
    char *pch;
    int i, prompt_ending = -1;

    for (i = conn->rdbuf_pos - 1; i > 0; i--)
    {
        if (conn->rdbuf[i] == ':' || conn->rdbuf[i] == '>' || conn->rdbuf[i] == '$' || conn->rdbuf[i] == '#')
        {
            prompt_ending = i + 1;
            break;
        }
    }

    if (prompt_ending == -1)
    {
        int tmp;

        if ((tmp = util_memsearch(conn->rdbuf, conn->rdbuf_pos, "assword", 7)) != -1)
            prompt_ending = tmp;
    }

    if (prompt_ending == -1)
        return 0;
    else
        return prompt_ending;
}
```

Listing 3.1: Original Mirai source code to check the clients' responses for the password prompt. Mirai determines standard password prompts with the characters :, >, $, and # and if not found, looks for 'assword' in the buffer.

```
while (TRUE)
    {
    int ret;
        [...]
    if (conn->rdbuf_pos == SCANNER_RDBUF_SIZE)
    {
        memmove(conn->rdbuf, conn->rdbuf + SCANNER_HACK_DRAIN, SCANNER_RDBUF_SIZE - SCANNER_HACK_DRAIN);
        conn->rdbuf_pos -= SCANNER_HACK_DRAIN;
    }
    [...]
    ret = recv_strip_null(conn->fd, conn->rdbuf + conn->rdbuf_pos, SCANNER_RDBUF_SIZE - conn->rdbuf_pos,
         MSG_NOSIGNAL);
    conn->rdbuf_pos += ret;
    [...]
}
```

Listing 3.2: Original Mirai source code to read clients' responses into a fixed length buffer of 256 bytes (`SCANNER_RDBUF_SIZE`). If the buffer is full, Mirai drops the first 64 characters of the buffer, moves the remaining 192 bytes along and adds 64 *new* bytes from the incoming stream of data to the end of the buffer.

3.3.1 Design

In our experiment, we used 100 public IP addresses that belong to the same American university as for the original study. The target computers were set up with a standard Linux Ubuntu operating system. To gain access to them, trespassers had to break into them successfully through Internet-wide scanning and guessing the authentication credentials.

3.3.2 Procedures

In line with the original work, we did not advertise our honeypots or actively recruit subjects to participate in our experiment. Instead, we deployed our target computers on the identical university network for a period of 365 days (June 6, 2018 to June 5, 2019) and waited for system trespassers to find them. The targets were modified to reject login attempts by system trespassers until a predefined number of attempts. As Maimon's experimental design, this predefined threshold was a random number between 150 and 200. While we are aware that this high number means that successful attacks are likely

Table 3.1: Overview: Original work and this study

	Maimon et al. 2014 [90]	This study
Result 1	A warning banner in the target computer does not lead to immediate termination of a system trespassing incident i.e. the proportion of first system trespassing incidents that were terminated on the warning target computers up to 5 seconds after a trespassing incident had started is almost identical to the proportion of incidents that were terminated in the same period on the no-warning computers.	Result confirmed – The warning banner and the Lorem Ipsum text do not lead to immediate termination of a system trespassing incident.
Result 2 - Cox model 1	A warning banner in the target computer is positively associated with the hazard of first system trespassing incident termination, i.e. a warning banner more than doubles the rate of first system trespassing incident termination and results in shorter duration of first trespassing incidents.	Result not confirmed – A warning banner and the Lorem Ipsum text in an attacked computer system are *not* positively associated with the hazard of first system trespassing incident termination.
Result 3 - Cox model 2	A warning banner increases the probability of trespassing session termination by 29 percent and demonstrates that a warning banner reduces the duration of system trespassing incidents on the attacked systems.	Result not confirmed – A warning banner does *not* increase the probability of trespassing session termination. We further find no evidence that a warning banner reduces the duration of system trespassing incidents.
Result 4	A warning banner does not reduce the volume of repeated system trespassing incidents on the target computer.	Result confirmed – The warning banner and the Lorem Ipsum text are not associated with an reduction in the volume of repeated systems trespassing incidents.

performed by automated scripts, we did not want to change the original design to ensure comparability between the original study and ours. When the threshold of 150 to 200 attempts was reached, the target computer was 'successfully' compromised in that it allowed the attacker access to the system by creating a new user with the latest credentials attempted by the system trespasser.[2]

Once access to our target computer had been granted, trespassers were randomly assigned to either a no-warning, a warning or a computer displaying the *Lorem Ipsum* message and initiated a system trespassing incident. When assigned to a warning condition, the following message appeared on the screen immediately after the intruder gained access:

> *The actual or attempted unauthorized access, use, or modification of this system is strictly prohibited. Unauthorized users are subject to institutional disciplinary proceedings and/or criminal and civil penalties under state, federal, or other applicable domestic and foreign laws. The use of this system is monitored and recorded for administrative and security reasons. Anyone accessing this system expressly consents to such monitoring and is advised that if monitoring reveals possible evidence of criminal activity, the Institution may provide the evidence of such activity to law*

[2]One IP address was not allowed to compromise more than one target in any 30-day period.

enforcement officials.

When assigned to the Lorem Ipsum computer, the following message appeared on the screen of the intruder:

> *Aliquam ut odio sapien. Morbi in est sed lectus consequat mollis. Fusce id risus eros. Ut eget lacinia elit. Aliquam condimentum libero iaculis viverra imperdiet. Nulla sit amet purus leo. Praesent auctor ac nunc non laoreet. Morbi justo purus, volutpat non velit a, dictum mollis tellus. Cras eget mi risus. Nam pulvinar ut odio sit amet venenatis. Aenean feugiat tincidunt ante, ac sollicitudin lacus convallis quis. Sed ut consectetur diam, ut condimentum libero. Nunc pulvinar, elit hendrerit commodo suscipit, orci mi luctus purus, eget viverra mauris enim eget orci. Curabitur sed condimentum orci amet.*

Both messages have 609 characters including spaces so that reading the characters into buffers and processing the information should take identical time. When assigned to a no-warning target computer, no message appeared on the screen of the intruder.

In line with the original work, we allowed system trespassers to use the target computers and initiate repeated system trespassing incidents for a period of 30 days, which might include sharing credentials with third parties. To ensure that the honeypots are not a threat to our own computer networks, all honeypots were actively monitored. Using Sebek as keylogger, we recorded each trespassing incident.

In an extension to the original work, we added further logging to differentiate between interactive and non-interactive sessions[3], and which commands were issued. This allows us to infer whether the observed activities are more likely to originate from automated scripts or humans. Typically users would request a shell to interact with the remote machine using a text interface. However, SSH also supports the execution of commands without requesting a shell. In this case, the server will only return the outcome of the executed command, but no further information, i.e. no shell prompt or any other additional text will be displayed. Thus the remote party has no means of seeing the warning message or the Lorem Ipsum text. We further log every command that is sent to the shell as well as every keystroke pressed. Both additional logging mechanisms allow us to potentially differentiate between automated scripts and humans based on the nature of the session and the associated timings of activities within a session. After 30 days have passed, the access to the target computer was revoked and a new target computer was redeployed.

During the 365 days of the experimental period, 410 target computers were compromised and infiltrated by system trespassers (144 of the computers had a warning banner installed, 133 showed the Lorem Ipsum text and 133 had no warning message, but the standard operating systems welcome message), and 3 795 system trespassing incidents were recorded; 1 024 (27%) of the system trespassing incidents were recorded on the no-warning computers, 1 102 (29%) sessions were recorded on the warning computers and 1 669 (44%) sessions were recorded on the computers which showed the Lorem Ipsum text. Importantly, most of the target computers experienced repeated system trespassing incidents.

To explore our research hypotheses, we first run our analyses using data on the first system trespassing incidents only (n = 410 trespassing sessions), and then we employ data on the entire poll of trespassing incidents recorded during the experimental period (n = 3 795 sessions).

An overview of the original work and its findings is presented in Table 3.1.

[3]The RFC4254 defines the term *session* as "a remote execution of a program. The program may be a shell, an application, a system command, or some built-in subsystem."

3.3.3 Outcome measures

As in the original work, we measured the time between the start and end of each trespassing session and calculated its duration. We then created two dependent measures. The first measure, immediate incident cessation, is a dummy measure (1 = immediate incident cessation) indicating the termination of a trespassing incident within a period of five seconds from its start.[4] The second measure, incident duration, is a continuous measure of the elapsed time (in seconds) between the beginning and the end of a trespassing incident.

3.3.4 Results

3.3.4.1 First trespassing incidents

We begin with analysing the first trespassing incidents recorded on each target computer, and we test for an association between the text message (no warning, warning banner, Lorem Ipsum) and immediate incident cessation as a dependent variable (1 or 0) using a Chi-Square Test. As shown in Figure 3.1, the results revealed no association between the type of text message and immediate incident cessation ($X^2(2) = 1.549, p = .461$).

Second, we test if a warning banners influences the survival time of system trespassing incidents. Identical to the original work, we employ event history analysis techniques because of the right-skewed distribution of the survival time of trespassing incidents. This analysis allows for estimating and comparing the proportion of sessions surviving an event, as well as a prediction of the rate at which duration end [90, 21].

To this end, we use standard life table methods to examine the effects of the warning and the Lorem Ipsum text on the time until system trespassing incidents terminate. To determine whether either treatment condition influences the time until termination, we compare the survival distribution of first trespassing incidents observed on all these target computers with the corresponding survival distribution of first trespassing incidents recorded on computers with no warning message. As indicated in Figure 3.2, overall the proportion of first trespassing incidents that survived *longer* periods of time is higher on the no-warning and Lorem Ipsum text computers than on the warning computers. Sessions that display the warning message are terminated faster within the first 5 seconds of the session, but after 5 seconds, the proportion of terminated sessions is similar across all target computers.

Consistent with the original work, we test whether the effect of a warning on the duration of system trespassing incidents is significant, by generating a dummy variable indicating whether a system trespassing incident was recorded on a warning, Lorem Ipsum or a no-warning target computer and estimating a Cox proportional-hazard regression [21]. The Cox model aims to explore the relationships between dependent and independent variables, but focuses on the independent measures that may be associated with the quantity of time (Box-Steffensmeier and Jones, 2004). The results from the estimated Cox model are presented in Table 3.2, model 1.

Contrary to the original work, we find that displaying a warning banner in the target computer is not positively associated with the hazard of first system trespassing incident termination. Similarly, displaying the Lorem Ipsum text is also not positively associated with the hazard of first system trespassing incident termination. Specifically,

[4]In line with the original work, we chose 5 seconds as a cut-off threshold so that attackers have sufficient time to see and read the banner.

Table 3.2: System trespassing incident duration regressed over Warning configurations

Variables	Model 1			Model 2		
	First Observed Incidents > 0 seconds			All Observed Incidents > 0 seconds		
	Cox Regression, x = 410			Frailty Model, n = 3795		
	Coefficient (SE)	P-Value	Hazard Ratio (HR) (95% CI for HR)	Coefficient (SE)	P-Value	Hazard Ratio (HR) (95% CI for HR)
Warning banner	0.149	0.217	1.161	0.321	0.27	1.379
Lorem Ipsum text	0.056	0.652	1.057	0.407	0.15	1.502
Log likelihood				−24 833.13		

ABBREVIATION: SE = standard error.

the results are statistically insignificant with p values of .652 for the Lorem Ipsum text and .217 for the warning banner. The finding is consistent with results obtained from log-rank tests for comparing the differences between the survival curves of warning vs. no-warning computers ($Z = 1.266, p = 0.206$) and Lorem Ipsum vs. no warning computers ($Z = -0.465, p = 0.642$).

3.3.4.2 All trespassing incidents recorded

To analyse the effect of a warning banner on the volume of repeated trespassing incidents, we use all 3 795 recorded trespassing incidents, and we estimate whether the mean number of repeated trespassing incidents recorded on the warning or Lorem Ipsum computers is significantly different than the mean number of repeated trespassing incidents observed on the no-warning computers. The results from an ANOVA indicate a non significant effect of the type of text message on the volume of repeated trespassing incidents at the $p < .05$ level for the three conditions $[F(2, 407) = .740, p = .478]$. Post hoc comparisons using the LSD test indicated that the mean score for the Lorem Ipsum condition ($M = 12.6, SD = 56.7$) was not significantly different than the no warning text ($M = 7.7, SD = 26.2$) and the warning text ($M = 7.7, SD = 22.2$).

We then compare the survival distributions of all system trespassing incidents recorded on the warning, the no-warning and the Lorem Ipsum computers. Figure 3.3 presents results from this comparison. We find that the proportion of trespassing incidents that survived longer periods of time is smaller on the Lorem Ipsum and warning computers than on the no-warning computers. To quantify the effect of a warning banner, we use shared-frailty models (or random-effect models). These models are used because they account for the heterogeneity and dependence issues generated by repeated observations [21, 85]. The results from the random-effect model are reported in Table 3.2, model 2. We find that the effect of a warning banner in the target computers is insignificant on the hazard of trespassing session termination (p value of .27). The same observation can be made for the effect of the Lorem Ipsum text with a p-value of .15. These findings confirm that neither the warning banner nor the Lorem Ipsum text positively or negatively affects the probability of trespassing session termination and that we find no evidence that a warning banner reduces the duration of system trespassing incidents.

3.3.4.3 Session types recorded and commands issued

In addition to the re-evaluation of the original work, we further analysed the requested session types (Section 3.3.2). In line with our previous findings, we found that 3 619

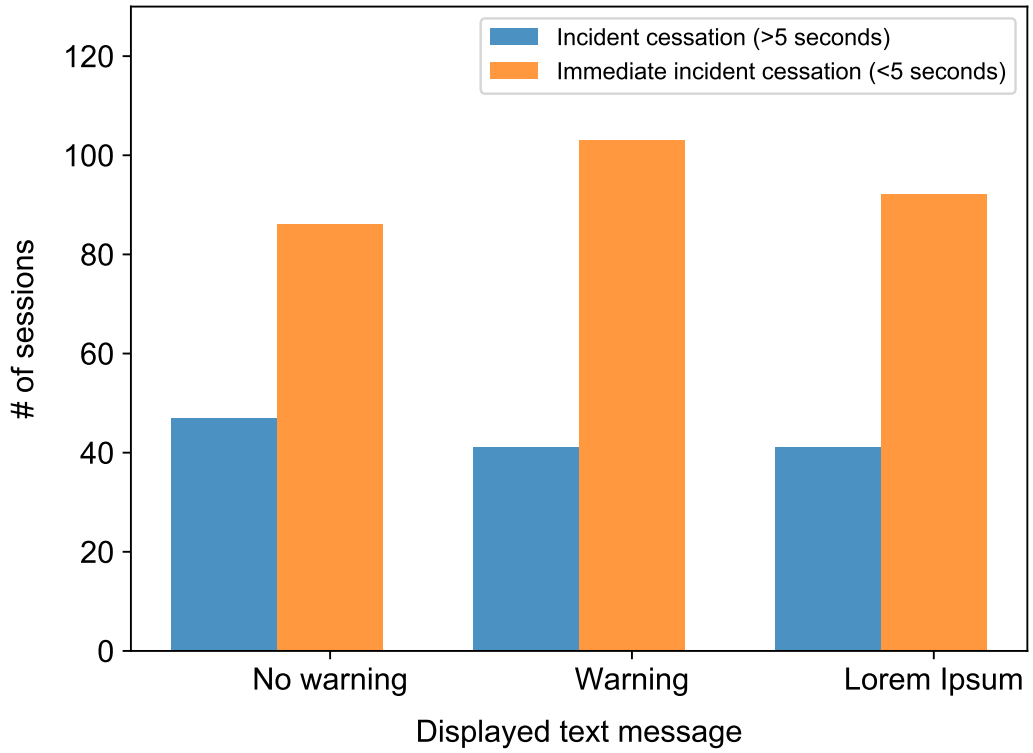
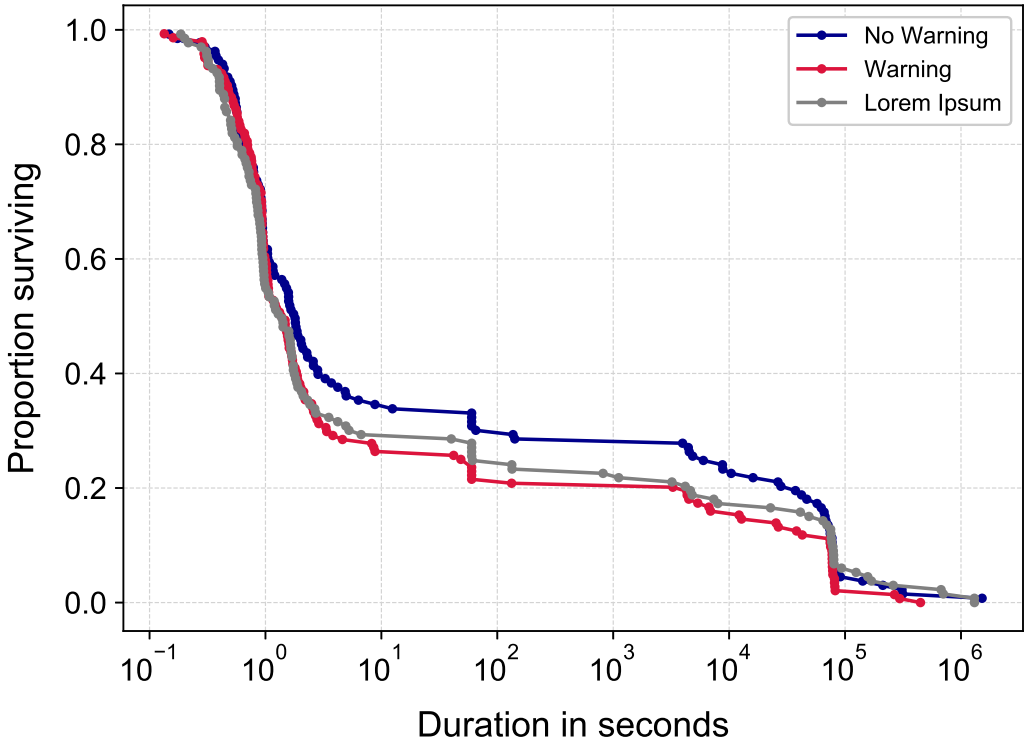


Figure 3.1: First trespassing incidents – Immediate incident cessation

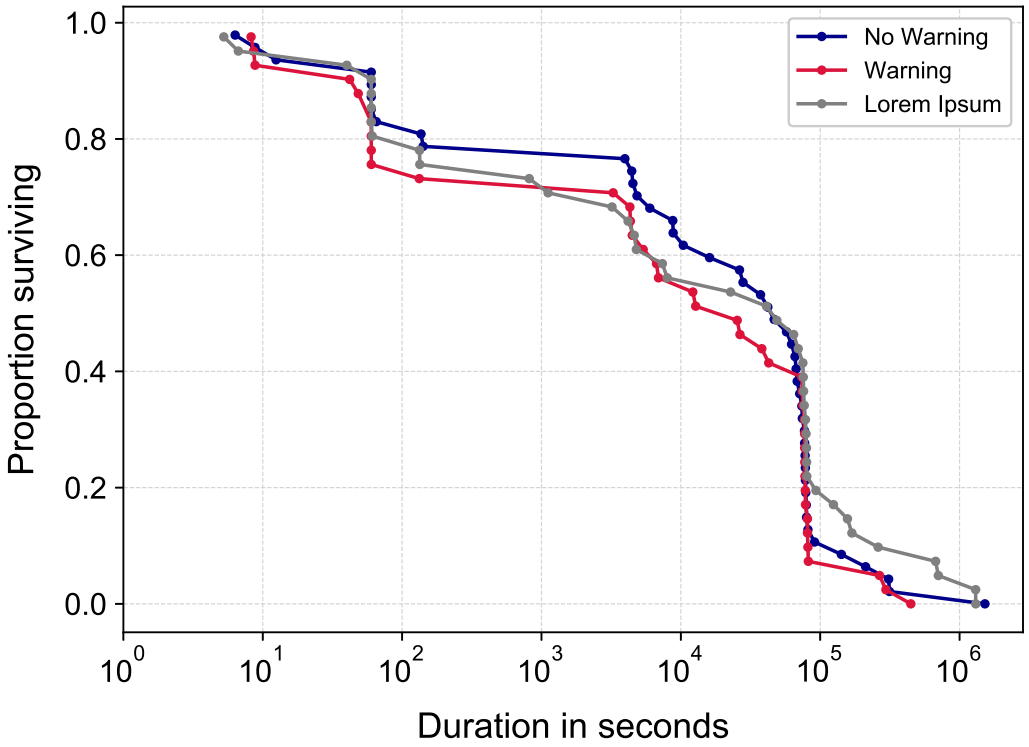
of 3 795 sessions were non-interactive and the remaining 176 sessions were interactive, i.e. keystrokes have been recorded. We further find that in 109 of 3 619 non-interactive sessions commands had been issued to download additional software so that we cannot be certain that the intruders were completely unaware of the warning banner or Lorem Ipsum text. The additional scripts and/or software could have been used to transmit further information about the systems status and its configuration once the machines have been compromised. However, as we have not recorded any keystroke information, we are confident that in these cases our machines did not interact with humans, but with automated scripts.

Overall, 1 690 commands have been issued and in 308 cases additional software was downloaded. The top three commands were `uname -a`, `wget` and `curl`. The command `uname -a` returns additional information of the comprised machine such as its hostname, operating system and current patch level. The latter two commands `wget` and `curl` are used to download additional software which is subsequently executed. We further found 999 sessions in which commands to delete the shell's history were issued. This is in line with previous research which has found that in about one third of attacks the shell history is deleted, in an attempt to cover the intrusion [2].

We further find, that the 176 interactive sessions were initiated from only 25 different IP addresses. This strongly suggests that, despite having the experiment designed in a way to ensure that any login attempt from an IP address that had already taken over a target computer in the previous 30 days was rejected, the design could not block distributed scanning and did not block subsequent logins to different target computers from the same IP address. This means that we are potentially dealing with fewer attackers/individuals than anticipated. Unfortunately, this also means that we cannot repeat the statistical

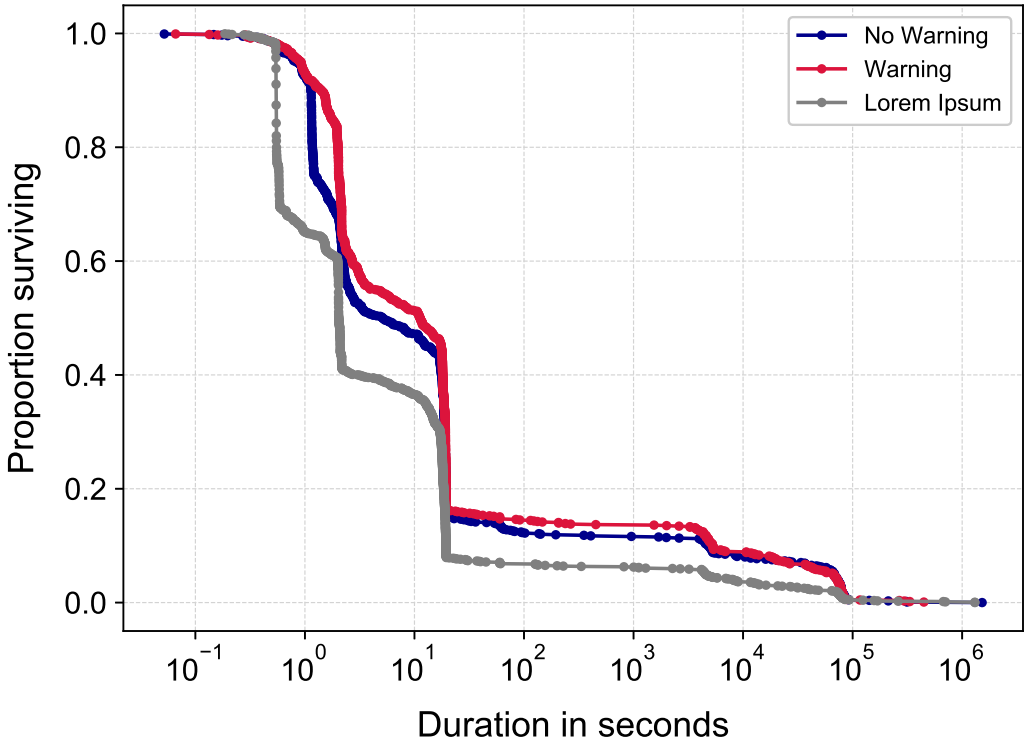


(a) Time to system trespassing incident termination – First trespassing incidents (n=410)

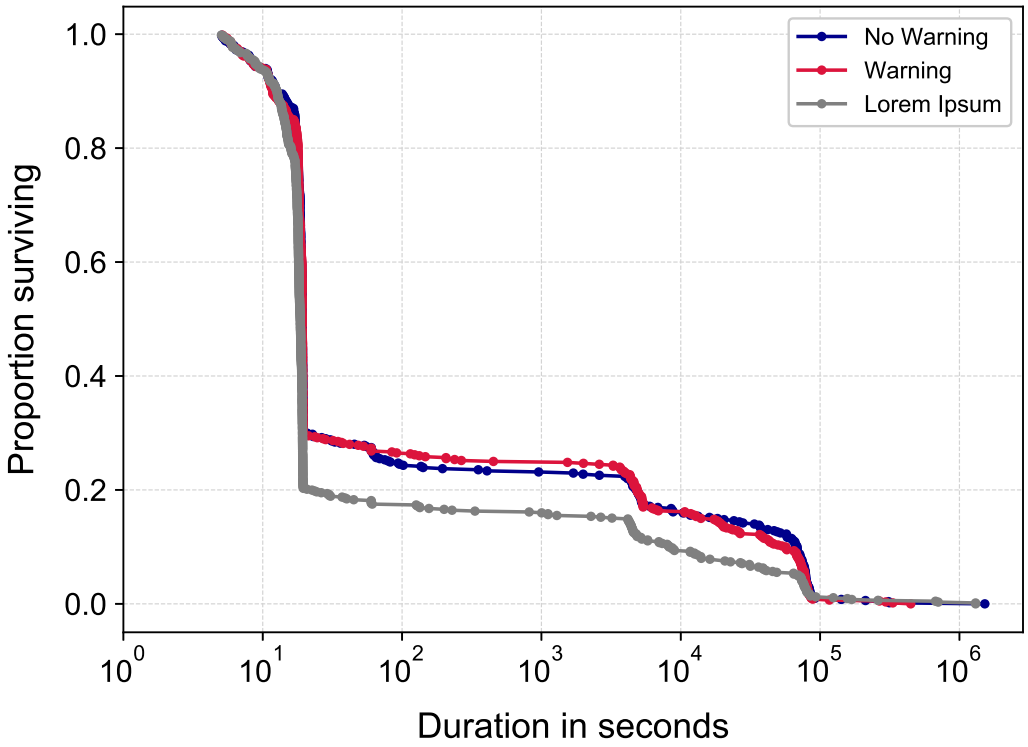


(b) Time to system trespassing incident termination – First trespassing incidents > 5 seconds (n=129)

Figure 3.2: First trespassing incidents



(a) Time to system trespassing incident termination – All trespassing incidents (N=3 795)



(b) Time to system trespassing incident termination – All trespassing incidents > 5 seconds (N=1 765)

Figure 3.3: All trespassing incidents

analysis for interactive sessions only. Cox regressions are best used with a minimum sample size of 10 events per variable [146], but as we only have 25 unique observations (in this case IP addresses) for 3 variables, this requirement is not met.

Table 3.3: Descriptive statistic by experiment condition

Target Computer Type	Target Computers with System Trespassing Incidents	Trespassing Incidents	Trespassing Incidents with Commands	Average Trespassing Incidents per Target Computer	Average Trespassing Incidents with Commands per Target Computer
No warning banner	131	695	276	5.31 (10.56)	2.11 (2.38)
Warning banner	144	1102	324	7.65 (26.22)	2.25 (2.34)
Lorem Ipsum	132	1145	237	8.67 (35.02)	1.80 (1.92)
Total	407	2942	837	7.23 (25.99)	2.06 (2.19)

Note: In the analysis shown, three target computers with 524, 178 and 151 observed commands are removed (see Section 3.3.4.3 for a detailed explanation). Standard deviations are presented in parentheses where appropriate.

For three computers we observed that the same command was entered hundreds of times: for the first instance, we recorded 524 commands from the same IP address in about 10 minute intervals with a median session length of 2.21 seconds. We observed similar behaviour for two computers for which we recorded 151 and 178 commands with a median session length of 0.548 and 1.158, respectively. These observations strongly suggest that these computers have been abused by automated scripts beyond what possibly could be the result of human interaction. As the fourth most popular honeypot deployment saw 16 commands entered, we consider these three deployments to be outliers for the next analysis.

Turning to an investigation of the effect of a warning banner on the probability of commands being typed in the target computers during system trespassing events, we estimate whether the mean number of entered commands on the warning or Lorem Ipsum computers is significantly different from the mean number of entered commands on the no-warning computers. The results from an ANOVA indicate a non significant effect of the type of text message on the number of entered commands at the $p < .05$ level for the three conditions $[F(2, 404) = 1.537, p = .216]$. Post hoc comparisons using the LSD test indicated that the mean score for the Lorem Ipsum condition ($M = 1.8, SD = 1.9$) was not significantly different than the no warning text ($M = 2.1, SD = 2.4$) and the warning text ($M = 2.3, SD = 2.3$).

3.4 Experiment 2: Surveying system trespassers

The goals of the second experiment are twofold. First, we wanted to study the prevalence of human trespassers by supplementing the SSH password authentication with a CAPTCHA. The CAPTCHA allows us to distinguish between humans and bots as the latter should not be able to solve it. Second, to better understand the motivation and psychological traits of system trespassers, we designed a survey which is aimed at those that attempt to access University computers without legitimate access.

We intended to get some indications about the size and nature of the market, explore participants' notions relating to the legality of the actions undertaken and obtain some understanding of who accesses computer systems without authorisation. In particular, we aimed to understand why they have chosen to access our computer systems, and their views about unauthorised access more generally.

This research was approved by Ethics Committee of the University of Cambridge, Department of Computer Science and Technology (ref: 450). The main risks associated with this experiment relate to the confidentiality and anonymity of the participants, particularly as in some jurisdictions, there may be legal issues relating to the unauthorised access of computer systems. Therefore, all survey responses were de-identified upon receipt, and no information that may identify participants, such as the participants' IP address or email address, was retained with the survey responses.

3.4.1 Recruitment

To undertake this research, we ran a modified version of OpenSSH version 7.2 on 50 public IPv4 addresses that belong to the University of Cambridge. We deployed the honeypots for a period of 365 days (October 26, 2017 to October 25, 2018). We did not advertise the deployed computers or their assigned IP addresses, but waited for attackers to find and attempt to compromise them.

To gain access to our computers, the attackers must guess credentials of 'legitimate' users – typically with the help of toolkits. We set up ten credentials for legitimate users with common username/password combinations such as admin/admin, root/123456 and raspberry/pi. After the correct credentials have been entered and a SSH2_MSG_-USERAUTH_SUCCESS packet has been sent, indicating the credentials are valid, a CAPTCHA is sent.

The CAPTCHA stated that "if you want to access this host, please type the following one digit number: 5" and thus is easily solved by humans. After the CAPTCHA is solved, we presented a welcome message and told the attacker that they have logged in to a honeypot and asked them to take part in our survey. The complete sequence of events is shown in Figure 3.4.

The displayed welcome message assured potential participants that the survey is for research purposes only. It further stated that, if compelled to disclose information such as by subpoena, the data cannot be linked to participants as it will be de-identified.

3.4.2 Survey design

The survey used LimeSurvey and contained 44 questions and takes approximately 30 minutes to complete. Due to the nature of the activities and the notion of automation, we did not anticipate a large response rate. Therefore, the survey was a mix of open and closed response options and was structured in two parts.

The first part was aimed to ask questions about their background, methods to locate and to gain access to computer systems, views about law enforcement, selection of targets, and income. In the second part, we offered them the possibility to respond to three psychometric tests – HEXACO personality scale, Susceptibility to Persuasion (StP-II) and Dark Triad (DT). For all tests, we provided results at the end of the survey and all results were automatically generated based on their answers.

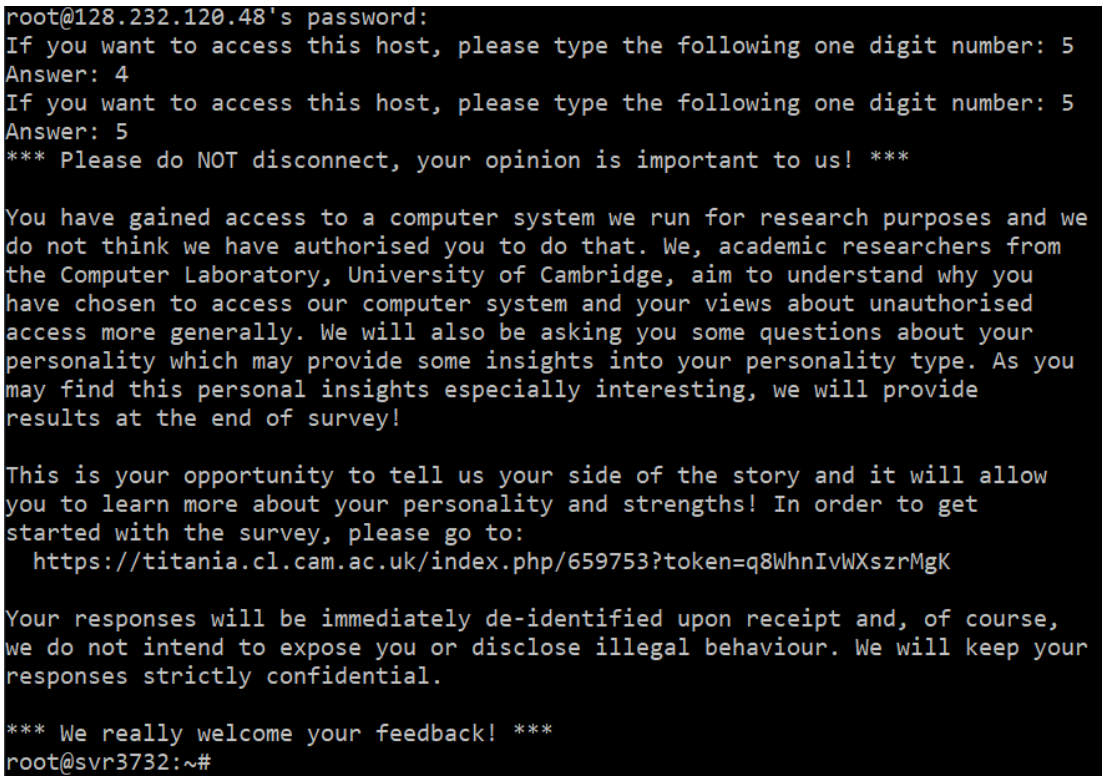


Figure 3.4: Surveying system trespassers – Sequence of events

All survey responses were de-identified upon receipt, and no information that may identify participants was retained with the survey responses. Participants were advised that they were free to withdraw from the study at any time, and can opt not to answer any of the questions they are asked.

3.4.3 Results

In the 365 days study period, we logged 23.3m connections, more than 64k a day. As discussed in Section 3.4.1, we ignored automated scripts that interacted with our honeypots and we would only ask humans to complete the survey. For 78 493 of the 23.3m connections, one of the ten valid credentials was used and the CAPTCHA was displayed. In 99 cases (0.13%) the CAPTCHA had been solved and as expected, all such sessions were interactive, i.e. our honeypots encountered less than 100 human interactions in a period of 365 days.

Unfortunately, only 4 of the 99 individuals to whom we showed the welcome message, used the URL to view the survey and none of them completed the survey. As we de-identified all information such as IP addresses upon receipt, we do not have any information who accessed our computer systems.

3.5 Discussion

Our study is the first to use empirical data to challenge the findings of Maimon et al. [90]. We find that the display of a warning banner in an attacked computer system does not increase the probability of trespassing session termination and does not result in shorter session duration. The vast majority of observed interactions in both our experiments were

automated scripts and only two dozen IP addresses initiated all interactive sessions in the one year study period (experiment 1). We confirmed this hypothesis with a second experiment in which we used a CAPTCHA to tell humans and bots apart, and found that human interactions are magnitudes lower than assumed by Maimon et al.

Maimon et al. assumed that "system trespassers [to] find [out] systems and employ special software cracking tools to break into them" [90, p. 40] and subsequently humans read the shell prompt, including the warning banner. However, we find that the whole process – from finding computer systems, trying user credentials and evaluating the systems usefulness – is mainly automated and performed by automated scripts. Thus, warning banners are not seen by any human and what has been measured is the deterrent effect of warning banners for automated scripts, which evaluate the system's usefulness by reading characters into buffers and comparing the content with a pre-defined set of features.

Since the research first appeared in 2014, two additional studies appeared which support Maimon et al.'s findings.

First, Stockman et al. [131] developed a custom honeypot solution in which they used bridge servers to forward SSH traffic to 27 honeypot hosts. To do so, they used HonSSH, a man-in-the-middle proxy for SSH. HonSSH accepts incoming connections from attackers and transparently forwards them to the target machines. In a period of 25 days, the authors collected data of 238 sessions which showed a warning banner and 284 for which no warning banner was displayed. The mean duration for those sessions with a warning banner was 15.29 seconds and without a warning banner 24.35. Interestingly, sessions without a warning banner have an eightfold higher variance (535.28 to 4767.59 seconds). This huge variance indicates, just as with the results we observed, that the scripts were puzzled about the usefulness of the system and either immediately determined that the system is useless (or in fact a honeypot) and therefore disconnected immediately, or were waiting for the shell prompt. Furthermore, their honeypots were set up so that logins with the username admin had a ten percent chance to log in to the system, regardless of what password was used. Unfortunately they did not give any justification why this arbitrary number was chosen. However, their set-up means that automated scripts are favoured as a substantial number of attempts is necessary to break into the system.

Second, Testa et al. [134] used the identical University network and experimental set up as Maimon et al., including the identical warning message and found that system trespassers with non-administrative privileges navigate through the filesystem and change file system permissions significantly less often on computers with warning banners than on normal computers. Interestingly, they acknowledge that they "could not fully distinguish between human-driven and bot-based attacks" [134, p. 718]. In fact, they do not discuss and did not make any attempt to distinguish between both. Furthermore, they argue that the high number of 150 to 200 attempts "limits the ability of human users to break into the system [and that] it prioritises the use of brute force toolkits by system trespassers" [134, p. 699]. Similar to Maimon et al. and Stockman et al., the arbitrary high number of 150 probes inevitably means that this task is likely performed by scripts. Testa et al. and Maimon et al. acknowledged the possibility that this might happen, but they assumed that a human would evaluate the shell prompt and take action while in reality, this process is completely automated.

One might argue that the level of automation has increased since the original data was collected in 2012. However, even if the level of automation in 2012 was lower than it is in 2019, one should not assume that all system trespassing incidents in 2012 were interactive

human sessions. Already in 2011, Nicomette et al. [102] deployed a high-interaction honeypot for 419 days and logged 552 333 connection attempts and found that only 153 sessions are likely to be human. To distinguish between humans and bots they considered (1) typos and (2) the way how data is transmitted between the user and the honeypot, i.e. character by character or in blocks. In 2016, Barron and Nikiforakis [11, p. 395] deployed 102 SSH honeypots over a four-month period and conclude that "one must expect that the majority of break-ins will originate from bots which will not necessarily be followed by a human attacker [and] that if one is interested in studying the attack patterns of humans (what they do once they break into a machine) they will have to deploy a large number of honeypots and use filters to remove automated bot-related activity".

In the attempt to distinguish between human behaviour and bots, we further find a small number of malicious actors (IP addresses) who logged in to our target computers using interactive sessions. In our first experiment, we logged 176 interactive sessions which were initiated from only 25 different IP addresses. Similarly, in our second experiment, we logged 99 interactive sessions in a one year study period. This indicates that only a very small number of interactions are performed by humans and that vulnerable machines are primarily found through distributed scanning, i.e. the initial system trespassing is done from a different machine than subsequent logins with interactive sessions. Overall, the numbers of potential human trespassers are magnitudes lower than previously assumed.

We further find that about one quarter of connections are long-lived connections with session durations of 80 000 seconds or more. While for first trespassing incidents, even though not reaching statistical significance, the warning banner results in shorter session duration, we cannot conclude that this is because of the semantic properties of the warning banner, but it is more likely the effect of pre-defined connection timeouts. In particular, we see an increase of dropped connection around 80 000 seconds, about 24 hours after the connection has been established. Looking at all trespassing incidents, we find that the Lorem Ipsum text results in shorter session durations, but again, not reaching statistical significance. In both instances however, the standard operating system (Ubuntu) message resulted in the longest sessions.

Our results further showed that the experimental design needs to be adjusted: many criminals share the same tools or obtain tools from specialised marketplaces. Thus simply rejecting login attempts until a random number between 150 and 200 does not "present a more genuine environment" as argued by Maimon et al. [90], but means that the scripts are likely to choose a password which appears between the 150th and 200th place on their brute-force lists. In fact, we find that in our first experiment 154 honeypots (37.6%) were deployed with a password either starting with the number `1` or with the letter `a`. First, this indicates that the dictionaries used by the attackers are sorted alphanumerically. Second, the dictionaries seemed to be fairly big as the number 1 and the letter a are at least within the first 150 passwords. Some might argue that for future studies a higher number of random attempts should be used to represent a more genuine environment so that the number of passwords starting with characters on top of a sorted alphanumerical dictionary are not that frequently used. However, attackers with the same dictionary will still share honeypots. Consequently, the current experimental set-up may not be successful at all in measuring the deterrent effects of warning banners.

One avenue might be to deliberately make log-in details of honeypots with different treatment conditions publicly available, for example, by posting them on appropriate forums and marketplaces. This will further increase the likelihood that honeypots interact

with humans and not with bots. Of course, one might have to think about the ethical and legal implications of this work, but advertising honeypots on various underground forums has been done in the past. Most recently, Onaolapo et al. [103] posted GoogleMail credentials on underground forums to better understand how compromised email accounts are used and monetised.

3.6 Conclusion

We challenge the findings of Maimon et al. and two subsequent studies and find that the display of warning banners has no deterrent effect in an attacked computer system. Our third treatment condition, a Lorem Ipsum text, led to similar results and no significant difference between computers that showed the warning banner and those that showed the Lorem Ipsum text can be found. The vast majority of observed interactions with our honeypots were automated scripts and only two dozen IP addresses initiated all interactive sessions in the one year study period. The original study completely ignored automated scripts and believed that all observed session interactions were humans. However, we show that only a tiny fraction of sessions were interactive.

We will need a better experimental design with more focus on the technical aspects of honeypot deployment and the threat landscape. Mechanisms such as CAPTCHAS instead of a predefined number of authentication attempts or the advertisement of honeypots on underground forums could help to study human behaviour only and thus measure if warning have a deterrent effect. However, even if warning messages are found to have a deterrent effect, they will only affect a small number of trespassing incidents as most trespassing incidents are fully automated. We further believe that more interdisciplinary research is necessary to better understand the technological implications of certain methodical choices as well the current threat landscape including criminal actors, tools and their motivations.

Chapter 4

Fingerprinting honeypots at Internet scale

The current generation of low- and medium-interaction honeypots uses off-the-shelf libraries to provide the transport layer. We show that this architecture is fatally flawed because the protocols are implemented subtly differently from the systems being impersonated.

We present a generic technique for systematically fingerprinting such honeypots at Internet scale with just one packet (in addition to the initial TCP three-way handshake) and an ERR (Equal Error Rate) of 0.0183. We conduct Internet-wide scans and identify 7 605 honeypot instances across nine different honeypot implementations for the most important network protocols SSH, Telnet, and HTTP. The nature of our techniques means that the logs kept by these honeypots will not show indisputable evidence that adversaries are actively fingerprinting the honeypot. We believe our findings to be a *class break* in that trivial patches cannot address the issue.

The work presented in this Chapter was published in the 12th USENIX Workshop on Offensive Technologies (WOOT '18) [143], and is in collaboration with Richard Clayton. This work was also presented at the 31st Annual FIRST Conference on Computer Security Incident Handling (FIRST '19).

4.1 Introduction

Attackers have a strong motivation to detect honeypots at an early stage as they do not want to disclose their methods, exploits and tools [67]. In Chapter 2, we discussed various methods of fingerprinting, including timing discrepancies and differences in application behaviour. Attackers have attempted to distinguish Telnet and SSH honeypots by executing commands within the login shell (or the impersonation of the login shell) and examining the responses. This has led to an arms race as attackers develop new distinguishers and honeypot authors improve the verisimilitude of their system.

However, if a honeypot can be detected at the transport level, for example without completing the SSH handshake or Telnet options negotiation, the honeypot's value will be minimal and efforts to impersonate the service will be in vain [78]. This aspect of the detection arms race is especially challenging because modern protocols such as SSH must handle a variety of versions, key exchange mechanisms, ciphers and service requests. Similarly, in Telnet the client and server can negotiate numerous settings such as line mode, echo and terminal type. As the RFCs do not mandate every aspect of a network

protocol, two implementations of a complex protocol may deal with ambiguities differently, and this may reveal the presence of a honeypot.

We are the first to observe that there is a generic weakness in the current generation of low- and medium-interaction honeypots because of their reliance on off-the-shelf libraries to implement large parts of the transport layer. These libraries are used for their convenience, but they were never intended to provide identical behaviour to 'real' servers. We systematically identify these differences by constructing distinguishing probes and show that they allow us to locate a large variety of honeypots by Internet-scale scanning. From the servers' responses we are further able to determine which implementation it is and also the exact software version.

We believe this to be a *class break* in that patches to the current generation of honeypots cannot solve the problem. Until these honeypots are given a new architecture, anyone with moderate capabilities has a lot of extremely quick and simple methods of identifying that a honeypot is running on a particular IPv4 address.

Overall, we make three main contributions:

- We present a generic and accurate technique for systematically fingerprinting low- and medium interaction honeypots by constructing distinguishing probes at the transport layer. We identified thousands of deviations between honeypots and the services they are impersonating.
- We use this technique to perform Internet-wide scans for 9 different honeypots for the most important network protocols SSH, Telnet and HTTP. We find 7 605 honeypot instances residing on 6 125 IPv4 addresses: 2 779 honeypot instances for SSH, 1 166 for Telnet and 3 660 for HTTP.
- We provide insights into how honeypots are configured and deployed in practice. We discover that only 39% of the honeypots were updated within the previous 7 months. Furthermore, we find that 546 (72%) of the 758 Kippo honeypots are still (44 months after our last scan) vulnerable to a known fingerprinting technique that was first disclosed in 2014.

4.2 Background

Table 4.1 provides an overview of the honeypots that we considered in our study, which we now discuss in detail.

Table 4.1: Honeypots in this study

	Updated	Language	Library
SSH			
Kippo	May 15	Python	TwistedConch
Cowrie	May 18	Python	TwistedConch
Telnet			
TPwd	Feb 16	C	custom
MTPot	Mar 17	Python	telnetsrv
TIoT	May 17	Python	custom
Cowrie	May 18	Python	TwistedConch
HTTP/Web			
Dionaea	Sep 16	Python	custom
Glastopf	Oct 16	Python	BaseHTTPServer
Conpot	Mar 18	Python	BaseHTTPServer

SSH Honeypots: In this work we consider SSH honeypots emulating generic servers running OpenSSH whose login credentials can be guessed and commands issued within an operating system shell such as `bash`. OpenSSH is the most widely used SSH protocol suite, and is installed on approximately 77% of all SSH servers listening on port 22 [46].

Many SSH honeypots have been developed over the years and one of the first SSH honeypots was Kojoney [75] but active development ceased around 2006. Kojoney uses the TwistedConch library which dates back to 2002 and is the de facto standard implementation of SSHv2 for Python2/3. Kojoney inspired Kippo [73] which was developed from 2009 to 2015 but the Kippo author now recommends people use a forked project called Cowrie [104]. Cowrie has added more extensive logging and support for Telnet, and it remains under active development. The project's philosophy is to only implement shell commands that are being used by attackers and so as of January 2018, Cowrie (partly) emulated 34 commands [35]. In 2015, Deutsche Telekom included Kippo in T-Pot, a multi-honeypot platform "based on well-established honeypots" [40]. T-Pot combines different honeypots for network services with an intrusion detection system and a monitoring and reporting engine. As of March 2016, Kippo was replaced by Cowrie "since it offers huge improvements over Kippo" [41].

Telnet Honeypots: Since mid-2016, and the rise of the Mirai botnet, there has been an increasing interest in Telnet honeypots. In this work we consider four recent systems: MTPot, Telnet-IoT-Honeypot (TIoT), Telnet-Password-Honeypot (TPwd) and Cowrie (which can act as a Telnet honeypot as well as an SSH honeypot).

MTPot, developed by Cymmetria Research, a company focusing on cyber deception, is designed to catch Mirai binaries. It is written in Python 2.x and uses the telnetsrv library for its Telnet protocol implementation. TIoT also aims to catch IoT malware and is also written in Python 2.x, but with a custom implementation of the Telnet protocol. TPwd is written in C and also has its own implementation of the Telnet protocol.

HTTP/Web Honeypots: There are many web application honeypots available – some focusing on emulating WordPress or various login interfaces to obtain credential guesses, while others are full web application honeypots. We focus on three honeypots of this second type: Conpot, Dionaea and Glastopf – all implemented in Python.

Conpot is a honeypot designed to emulate industrial control systems and by default it listens on ports 80, 102, 161 and 502. Dionaea supports almost all protocols and provides templates for each of them, including HTTP. Glastopf specifically focuses on HTTP and uses the BaseHTTPServer library to implement the protocol. Glastopf is the most highly recommended honeypot for HTTP, including in the ENISA report we mentioned in Chapter 2. Glastopf is also included within the latest versions of T-Pot.

4.3 Systematically fingerprinting honeypots

We present a new generic technique to systematically identify low- and medium-interaction honeypots based on protocol deviations and before any authentication takes place, i.e. our technique does not require log-in credentials. We find probes that result in distinctive protocol responses. This has resulted in the identification of thousands of deviations between honeypots and the services they are impersonating.

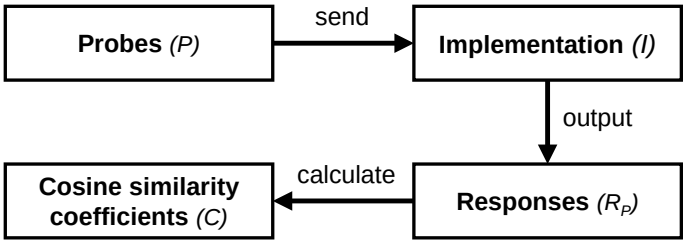


Figure 4.1: Steps to identify probes with distinctive responses, which can then be used for Internet-wide scans.

4.3.1 Efficient detection of deviations

Given a set of implementations of a network protocol $I = \{I_1, I_2, ..., I_x\}$, we send a set of probes $P = \{P_1, P_2, ..., P_n\}$ to I and record the set of responses R_P. We then calculate the cosine similarity coefficients C for all responses R_{P_i}. The aim is to find the 'best' P_i where the sum of C is the lowest, i.e. the responses R_{P_i} for I_j are overall the least similar. In other words, we try to find *the* probe that results in the most distinctive response across all the protocol implementations.

Cosine similarity is an established technique for comparing sets of information, commonly used to measure text semantic similarity. It has also proven useful in traffic analysis to find abnormalities [157] and to measure domain similarity [88, 154]. We represent our responses R_P as a vector of features appropriate to the network protocol. For example, in the case of Telnet, each individual terminal option character is treated as a feature. The resulting cosine similarity coefficient is a normalized value between 0 and 1. The higher the coefficient, the more similar the two items under comparison. The overall approach is outlined in Figure 4.1.

We fingerprinted the responses from widely deployed systems that support SSH, Telnet or HTTP along with the protocol libraries commonly used in building honeypots for these protocols. To obtain the 'ground truth' for each of the honeypots and reference implementations, we compiled and ran them locally. To trigger responses, the probe generation followed the syntax of the respective protocol RFCs, but we altered individual parts as subsequently outlined in each Section. As honeypot developers supplement their chosen protocol library with custom code to emulate reference implementations, it is reasonable to expect to be able to generate *unique* fingerprints and so it proved. We then used the probe that resulted in the most distinctive responses for each protocol and sent it to every host on the Internet. From the responses we are able to determine the implementation (possibly a honeypot) and also the software version that is running on each host.

The key point of our technique is that we are able to rapidly identify honeypots because of the way in which they have been implemented (using a particular protocol library) rather than having to consider how they respond when interacted with at length.

4.3.2 Protocol 1: SSH

We look for deviations in responses to a client version string and a `SSH2_MSG_KEXINIT` packet. For this comparison, we use five OpenSSH versions (6.6, 6.7, 6.8, 7.2, 7.5), the SSH

server example supplied with TwistedConch and eight versions of the honeypots Kippo[1] and Cowrie[2], four for each honeypot. We include multiple versions of Kippo and Cowrie as both honeypots have undergone substantial modifications over the years.

First, we create a set of client version strings: `SSH-protoversion-swversion SP comment crlf`. Our probes follow this syntax, but we alter individual parts – expecting this to result in differing responses. We start with 'ssh' and 'SSH', we use 12 different protoversions ranging from 0.0 to 3.2, swversion (which identifies the software) we set OpenSSH or an empty string, comment we set to FreeBSD or an empty string, and the terminating crlf we set to either \r\n or an empty string. In short, we construct 192 client version strings: `[SSH, ssh]-[0.0, 0.1, 0.2, 1.0, 1.1, 1.2, 2.0, 2.1, 2.2, 3.0, 3.1, 3.2]-[OpenSSH, ""] SP[FreeBSD, ""][\r\n, ""]`.

Second, we create `SSH2_MSG_KEXINIT` packets using the algorithms defined in RFC 4250 [83] and its intended update [12]. Together these give 16 key-exchange algorithms, 2 host key algorithms, 15 encryption algorithms, 5 MAC algorithms and 3 compression algorithms. We supplement each of them with an empty string and do not include any supported languages in our packets. We populate the 16 byte cookie with random bytes and correctly pad the packet. This leads to the generation of 19 584 correctly formed packets where each packet offers just one algorithm of each type. In addition to these correctly formed packets, we create a variant with incorrect padding (mod 13 instead of mod 8) and another variant for which we omit the packet and padding length.

In total, we generate 58 752 different packets; in combination with the client versions, we issue 157 925 376 probes, 11 280 384 to determine how each of our 14 implementations will respond.

We record every character the servers send in response, including random content such as the cookie or the padding. Thus for SSH, we do not expect a cosine similarity of 1.0. In fact, including *random* parts has proven very valuable as we find that OpenSSH uses NULL characters to pad packets, but the honeypots use random bytes for padding (see Section 4.5.1).

Table 4.2: Average similarity measures across all tested SSH implementations based on their responses (n=157 925 376)

		A	B	C	D	E	F	G	H	I	J	K	L	M	N
OpenSSH 6.6	A	X	0.98	0.98	0.94	0.94	0.42	0.75	0.75	0.75	0.66	0.78	0.79	0.79	0.79
OpenSSH 6.7	B		X	0.98	0.98	0.98	0.41	0.76	0.76	0.76	0.68	0.80	0.81	0.81	0.80
OpenSSH 6.8	C			X	0.96	0.96	0.42	0.76	0.76	0.76	0.76	0.78	0.79	0.79	0.79
OpenSSH 7.2	D				X	0.98	0.42	0.76	0.76	0.76	0.68	0.80	0.80	0.80	0.80
OpenSSH 7.5	E					X	0.41	0.80	0.80	0.80	0.71	0.78	0.79	0.79	0.79
Twisted 15.2.1	F						X	0.46	0.46	0.46	0.45	0.50	0.51	0.51	0.52
Kippo 0d03635	G							X	0.99	0.99	0.92	0.88	0.88	0.88	0.88
Kippo 40b6527	H								X	0.99	0.92	0.88	0.88	0.88	0.88
Kippo 4999618	I									X	0.92	0.88	0.88	0.88	0.88
Kippo 9645e50	J										X	0.83	0.83	0.83	0.83
Cowrie 96ca2ba	K											X	0.98	0.98	0.98
Cowrie dc45961	L												X	0.99	0.99
Cowrie dbe88ed	M													X	0.99
Cowrie fd801d1	N														X

Table 4.2 gives the average similarity scores across all of the implementations. Overall, Kippo and Cowrie respond similarly to OpenSSH, with an average cosine similarity measure

[1]Commits 0d03635, 40b6527, 4999618 and 9645e50 on https://github.com/desaster/kippo

[2]Commits 96ca2ba, dc45961, dbe88ed and fd801d1 on https://github.com/micheloosterhof/cowrie

ranging from 0.66 to 0.81, but in no case do they manage to be identical across all probes. After calculating all the cosine similarity coefficients, as outlined in Section 4.3.1, we find that `SSH-2.2-OpenSSH \r\n` as version string and the `SSH2_MSG_KEXINIT` packet including ecdh-sha2-nistp521 as key-exchange algorithm, ssh-dss as host key algorithm, blowfish-cbc as encryption algorithm, hmac-sha1 as mac algorithm and zlib@openssh.com as compression algorithm, with the wrong padding, is the probe with the lowest cosine similarity coefficient C and will be used in our scans because it is the best distinguisher between honeypots and non-honeypot SSH servers.

4.3.3 Protocol 2: Telnet

For Telnet we look for deviations in responses to our negotiation requests. For this comparison, we use Busybox versions 1.6.1, 1.7.2, 1.8.0, 1.9.0, 2.0.0, 2.1.1, 2.4.0, 2.6.2, Ubuntu-telnetd 0.17.4, FreeBSD 11.1 telnetd and the honeypots MTPot[3], Cowrie[4], TPwd[5] and TIoT[6].

Given the `IAC` escape character, four option codes `WILL, WON'T, DO, DON'T` and 40 Telnet options[7], we create 160 different negotiation requests (n).

The Telnet protocol specifies that an arbitrary number of requests (r) can be sent at any time. To get the most exhaustive coverage we test all 160 negotiation requests ($n = 160$), but limit the maximum number of negotiation requests per connection to two ($r = 2$). As we do not want to send the same requests twice, we generate $\frac{160!}{(160-2)!} = 25\,440$ probes for each Telnet implementation. In total we generate 356 160 responses, 25 440 for each of our 14 implementations. The responses will contain the negotiation options that the server sends initially, along with the response it makes to our probe.

Table 4.3: Average similarity measures across all tested Telnet implementations based on their responses (n=356 160)

		A	B	C	D	E	F	G	H	I	J	K	L	M	N
Busybox 1.6.1	A	X	1	1	1	1	1	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 1.7.2	B		X	1	1	1	1	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 1.8.0	C			X	1	1	1	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 1.9.0	D				X	1	1	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 2.0.0	E					X	1	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 2.1.1	F						X	1	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 2.4.0	G							X	0.99	0.89	0.83	0.89	0.85	0.85	0.88
Busybox 2.6.2	H								X	0.89	0.83	0.89	0.85	0.85	0.88
MTPot	I									X	0.90	0.99	0.84	0.89	0.86
Cowrie	J										X	0.88	0.95	0.97	0.94
TPwd	K											X	0.83	0.87	0.85
TIoT	L												X	0.94	0.96
FreeBSD 11.1 telnetd	M													X	0.96
Ubuntu-telnetd 0.17.4	N														X

Table 4.3 gives the average similarity scores across all of the implementations. Again, the honeypots respond in a similar way to other systems, but in no case do they manage to be identical across all probes. Cowrie responds most similarly to Ubuntu telnetd with

[3]https://github.com/Cymmetria/MTPot/commit/c32d433e
[4]https://github.com/micheloosterhof/cowrie/commit/ffe669f
[5]https://git.zx2c4.com/telnet-password-honeypot/commit/0f9b0c
[6]https://github.com/Phype/telnet-iot-honeypot/commit/15343df9
[7]We only use the main options from 0 to 39 [68]

an average cosine similarity measure of 0.94 followed by MTPot with 0.89. Interestingly, Busybox versions 1.6.0 to 2.4.0 have identical behaviour. After calculating all the coefficients, we find that `\xff\xfb\x00\xff\xfb\x12`, i.e. `IAC WILL BINARY IAC WILL LOGOUT` is the probe with the lowest cosine similarity coefficient C and will be used in our Internet-wide scan to find honeypot implementations.

4.3.4 Protocol 3: HTTP/Web

For HTTP we look for deviations in responses to HTTP method requests. For this comparison, we use Apache versions 2.0.50, 2.2.34, 2.4.27, nginx versions 1.0.15, 1.4.7, 1.12.1, python3.5.2-aiohttp version 2.2.0, python2.7-simplehttpserver, python2.7-basehttpserver, Glastopf[8], Conpot[9], and Dionaea 0.6[10].

Our probes follow the syntax of HTTP requests and are formed as follows: `method char version.` When considering the responses we omitted the semantics of the date and time information that is included in the header, but not the syntax. This prevents region/language configuration differences from affecting our results.

We use the 43 different request methods defined in RFC2616 [53] and RFC2518 [58] (including Webdav methods), the 123 non-alphanumeric ASCII characters with a preceding `/` as the path and 9 different HTTP versions ranging from version `HTTP/0.0` to `HTTP/2.2` to create our set of probes. In total, we sent 571 212 probes, 47 601 for each of our 12 implementations.

Table 4.4: Average similarity measures across all tested HTTP implementations based on their responses (n=571 212)

		A	B	C	D	E	F	G	H	I	J	K	L
Apache 2.0.50	A	X	0.74	0.74	0.23	0.23	0.03	0.39	0.27	0.29	0.02	0.10	0.19
Apache 2.2.34	B		X	0.97	0.26	0.26	0.04	0.50	0.31	0.32	0.01	0.09	0.20
Apache 2.4.27	C			X	0.26	0.26	0.04	0.51	0.31	0.32	<0.01	0.09	0.20
python2-basehttpsevr	D				X	0.64	0.01	0.17	0.08	0.08	0.11	0.02	0.08
python2-simplehttpsvr	E					X	0.01	0.17	0.08	0.08	0.11	0.02	0.08
python3-aiohttp	F						X	0.04	0.02	0.02	<0.01	<0.01	0.01
nginx 1.12.1	G							X	0.57	0.57	<0.01	0.04	0.17
nginx 1.4.7	H								X	0.57	<0.01	0.02	0.10
nginx 1.0.15	I									X	<0.01	0.02	0.11
Glastopf	J										X	<0.01	<0.01
Conpot	K											X	0.02
Dionaea 0.6.0	L												X

Table 4.4 gives the average similarity scores. Again, no honeypot behaves identically to any of the systems we tested. Compared with our SSH and Telnet results, the similarity measures are much lower and the web server implementations respond far more distinctively. We find that Dionaea outperforms all the other honeypots we tested and is the most identical to Apache and nginx with average similarity measures ranging from 0.10 for nginx 1.4.7 to 0.20 for Apache 2.4.27. As expected, Glastopf most resembles python2-basehttpserver and python2-simplehttpserver – the underlying libraries used to provide its transport layer. We then identified the probe with the lowest coefficient C and we use `GET /. HTTP/0.0\r\n\r\n` for the Internet-wide scan.

[8]https://github.com/mushorg/glastopf/commit/bcbcebe
[9]https://github.com/mushorg/conpot/commit/74699fc
[10]https://github.com/DinoTools/dionaea/commit/02492e2b

4.4 Internet-wide scanning

We use the probes we identified to perform six scans, two scans each for SSH, Telnet and HTTP honeypots, to find honeypots at Internet scale. Table 4.5 summarises these scans and gives the number of detected honeypots. All our scanning is performed from a dedicated host within our University network and in accordance with the ethical considerations outlined in Section 4.7.

First, we use ZMap and perform a one-packet scan sending TCP SYN packets to the respective ports 22, 23 and 80 using the exclusion list maintained by DNS-OARC [43]. In total we scanned 3 336m IPv4 addresses, 78% of the IPv4 address space. We configured ZMap to scan at 30mbps and determined which IPv4 addresses responded successfully with a SYN-ACK packet.

Second, responsive IPv4 addresses were visited by a custom scanner which connects on the appropriate port and sends probes to identify honeypots. For each responsive IPv4 address we only try to connect once, with a socket timeout of six seconds.

For SSH, we only consider servers which appear to be running OpenSSH configured for SSHv2, i.e. when we connect to them on port 22 they send the server version string `SSH-2.0-OpenSSH_*`, where `*` is the OpenSSH version (number). We then determine whether the server behaves identically to OpenSSH.

Table 4.5: Results of the Internet-wide scan. *Scan 1 (SSH) was performed with the techniques outlined in Section 4.5

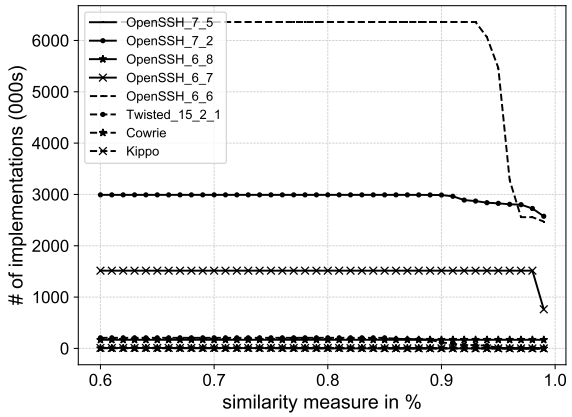
	Date	#ACKs	Sum	Kippo	Cowrie
Scan 1 (SSH)*	2017-09	18,196k	2844	906	1938
Scan 2 (SSH)	2018-01	20,586k	2779	758	2021

	Date	#ACKs	Sum	Dionaea	Glastopf	Conpot
Scan 1 (HTTP)	2017-10	58,775k	2616	139	2390	87
Scan 2 (HTTP)	2018-01	67,615k	3660	202	3371	87

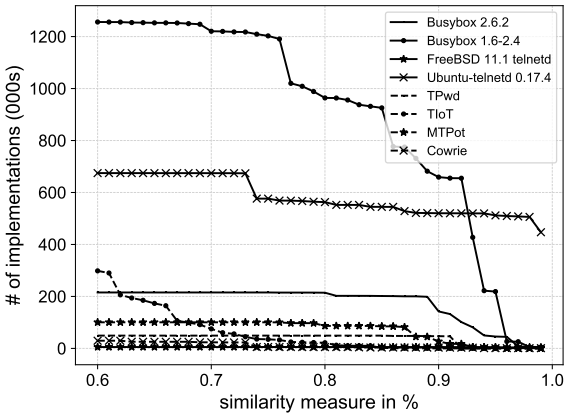
	Date	#ACKs	Sum	TPwd	MTPot	TIoT	Cowrie
Scan 1 (Telnet)	2017-09	8,290k	1430	1	388	22	1019
Scan 2 (Telnet)	2018-01	8,169k	1166	1	216	11	938

4.4.1 Results

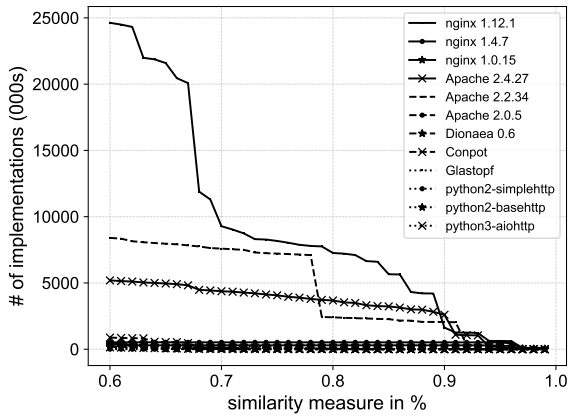
As shown in Figure 4.2a, we find that most SSH implementations are similar to OpenSSH 6.6 and 7.2. Only when we look for a cosine similarity score of 0.9 and higher does the number of hosts classified as OpenSSH significantly decrease. As we do not exclude the 'random' parts of the servers' responses (Section 4.3.2) we have no easy way of defining a threshold of cosine similarity at which we should accept the hypothesis that responses originate from SSH honeypots. We discuss how we overcome this with a detailed analysis of false positive and false negative rates in Section 4.4.2 below. Doing so, we classify 758 honeypots as Kippo and the remaining 2 021 are Cowrie honeypots (Scan 2). As Kippo and Cowrie respond very similarly to our probes, we differentiate them based on the disconnection messages (see Section 4.5.2).



(a) SSH implementations



(b) Telnet implementations



(c) HTTP implementations

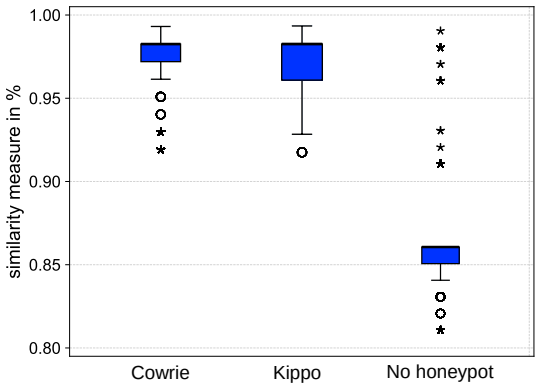
Figure 4.2: Similarity of SSH, Telnet and HTTP implementations in the Internet-wide scan based on their responses to our probes. Results are based on the latest scans for each protocol.

The very first scan predated our systematic method of fingerprinting honeypots and was performed by sending a non-compliant `SSH2_MSG_KEXINIT` packet to each responsive IPv4 address (Section 4.5.2). In that first scan, we found 2 844 honeypot instances, 1 938 instances of Cowrie and 906 of Kippo.

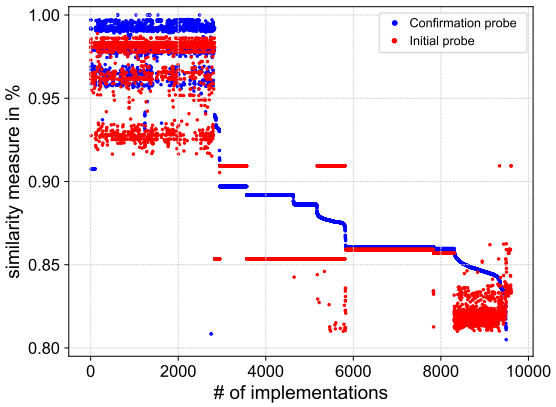
By design, Telnet servers do not advertise their implementation or version, so there are no deployment statistics available; we are the first to fill this gap. As shown in Figure 4.2b, a significant number of hosts are similar to Busybox versions 1.6 to 2.4, but as the similarity measure increases, the number of hosts we can definitively identify significantly decreases. We also find about 400k Telnet servers that are identical to Ubuntu telnetd. When identifying honeypots we only consider exact matches (a cosine similarity score of 1.0) and we find that there are 1 430 Telnet honeypots deployed. The vast majority of these are Cowrie (1 019) followed by MTPot (388), ToIT (22) and TPwd (1). In our second scan three months later we find 1 116 Telnet honeypots and again, the vast majority are Cowrie honeypots (938) followed by MTpot (216), TIoT (11) and TPwd (1).

As shown in Figure 4.2c, most HTTP implementations resemble nginx 1.12.1 and Apache 2.2.34. We further observe that the number of implementations that are at all

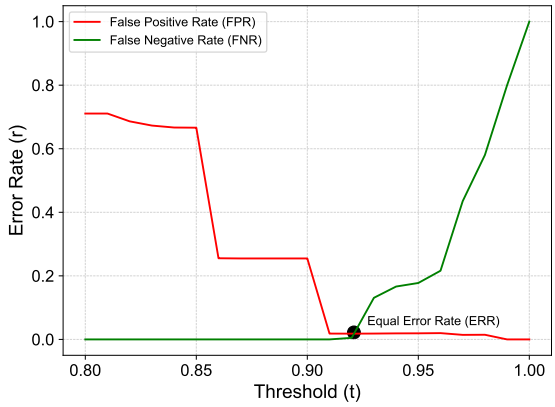
similar to one of the honeypots or the plain library implementations is minimal. In total, we find 2 616 instances of HTTP honeypots in our first scan and 3 660 in the second scan. Glastopf is the most widely used honeypot with 3 371 instances followed by Dionaea (202) and Conpot (87). Differences between scans not only reflect changes to the honeypot population but also of course whether temporary Internet glitches meant that SYN-ACKs were not returned to ZMap.



(a) We find a significant difference between the cosine similarity scores and the classified groups – Cowrie honeypots, Kippo honeypots and hosts not classified as honeypots.



(b) Result confirmation: For all of the fingerprinted honeypots the cosine similarity score for both, the initial probe and the confirmation probe is 0.90 or higher.



(c) Our methods achieves an ERR of 0.0183 with a threshold of 0.9235, i.e. responses with a cosine similarity score of 0.9235 and higher are classified as honeypots.

Figure 4.3: We validate our method by removing the random parts in the servers' responses and sending the second-best distinguishing probe to each potential honeypot.

4.4.2 Validation

The nature of honeypots means that there is no publicly available list of honeypot IP addresses so we looked for ways to cross-validate our method.

The Telnet and HTTP Honeypots in our study do not include randomized content in their responses. Thus we can classify hosts as honeypots only when there is an exact match (cosine similarity score 1.0). We then used the second-best distinguishing probe for these honeypots to confirm the initial hypothesis that the servers' response is unique to the specific honeypot implementation. Doing so, we find that 1 136 of 1 166 (97.4%) Telnet honeypots, and 3 549 of 3 660 HTTP (97.0%) honeypots respond to these probes as expected (cosine similarity score of 1.0 for both probes). Manual inspection shows that all the discrepancies are caused by incomplete responses.

For SSH we first considered all responses classified as honeypots with a cosine similarity of 0.80 or more (9 607). We then remove the packet length, padding length, cookie and the random padding. This results in the identification of 2 779 instances of SSH honeypots which now had a cosine similarity score of 1.0. We find a significant difference in the original cosine similarity scores between what we now know to be Cowrie honeypots (2 021), Kippo honeypots (758), and all the other responses which we initially classified as not being a honeypot (6 828); Kruskal-Wallis, $p = 0.001$ with a mean rank of 8 245 ($median = 0.981$) for *Cowrie* honeypots, 8 057 ($median = 0,972$) for *Kippo* and 3 424 ($median = 0.857$) for *Non-honeypots* (see Figure 4.3a).

We further send the second-best probe to all of the 9 607 implementations that were initially been classified as SSH honeypots. As can be seen in Figure 4.3b, for all of the 2 779 fingerprinted honeypots the cosine similarity score for both, the initial probe and the confirmation probe (including the random parts), is 0.90 or higher.

In summary, the cosine similarity of 1.0 for payloads makes us certain that we have found 2 779 SSH honeypots. Furthermore, the resulting cosine similarity values of the second-best probe are effectively identical to the initial probe, 0.90 and higher.

4.4.3 Accuracy

Evaluating our method further, it is essential to report false positive rates (FPR) and false negative rate (FNR). The FPR indicates how often our method identifies hosts as honeypots when they aren't and the FNR is the likelihood that our method fails to identify hosts as honeypots when they are. Assuming that the ground truth is 2 779 honeypots, we get an Equal Error Rate (ERR) of 0.0183 using the threshold (t) of 0.9235 and the *best* probe (see Figure 4.3c). In other words, at this point we falsely accept and at the same time fail to identify 51 honeypots.

When using both the best probe and the second-best probe, we achieve a slightly better ERR of 0.0132. This is a minor improvement and in most situations not worth the additional overhead of sending twice the number of packets. Arguably a real attacker would choose a higher FPR so they can be certain not to touch a honeypot at the expense of excluding potential targets.

4.4.4 Honeypot deployment

Update behaviour of SSH honeypots: Based on the honeypots' responses, we split the SSH honeypots into four groups according to their patch level. The results are shown in Table 4.6. In the first scan we found that 695 (24%) of the Kippo honeypots were more than 40 months out of date and hence would fall to a well-known fingerprinting technique first disclosed on 2014-05-28. Even by the second scan three months later, 546 had still not been updated (72.0% of a reduced population).

Kippo has not been actively developed since 2015, but the people running Cowrie are also failing to keep their deployments up to date. Our figures from the second scan show that only 1 071 (53%) of these honeypots had incorporated improvements from 7 months earlier. Since developers track the commands that adversaries use and continually add new features to make honeypots more covert, not updating a honeypot increases the chances that it may be fingerprinted (using traditional techniques) and thus limits its value in detecting new attack vectors.

Mass deployment of honeypots: We scanned for all three types of honeypots independently, but we now consider what we learn by linking them by IPv4 address. The 7 605 honeypot instances of our second scan reside on 6 125 IPv4 addresses. Thus a significant number of honeypot operators deploy several honeypots on a single host. We find 714 IPv4 addresses run Cowrie on port 22 (SSH) and simultaneously Glastopf on port 80 (HTTP); there are also 550 instances of Cowrie being run on both port 22 (SSH) and port 23 (Telnet). The risk here is that fingerprinting one honeypot instance may reveal the presence of other honeypots on the same host.

We also fetched the SSH host keys, which are intended to be unique, of all the honeypots that we had identified. We find that only 1 838 of the 2 844 SSH honeypots (65%) in the first scan have a unique host key. The remaining 1 006 honeypots have 71 host keys between them with a median of 5 honeypots per host key. For the second scan only 2 193 of the 2 779 honeypots (78.9%) have unique host keys. It follows that a substantial number of honeypot operators deploy more than one honeypot and while doing so exercise little caution. It is likely that honeypot operators use deployment scripts, docker containers or simply copy and paste the source files, including the same host key, to all their honeypots.

We determine which hosting companies honeypot operators use; we list the Top 10 ASs for all the honeypots that our scans identified in Table 4.7. We find that the honeypots are hosted in 82 countries, with the majority being located at well-known cloud providers in the USA. As low- and medium-interaction honeypots are not very resource-intensive the main criteria for honeypot operators will be reliability and cheap hosting with the ability to quickly re-set or re-deploy a (compromised) honeypot. Thus, it is not surprising that honeypot operators mainly locate their honeypots at well-known cloud providers.

4.5 SSH: Implementation flaws

We now explore some sources of divergence for the SSH protocol and show that there are numerous differences 'on-the-wire' between entirely standards compliant implementations; each of which gives attackers a quick and easy way of identifying honeypots at Internet scale.

4.5.1 SSH Binary Packet Protocol

We find that Kippo and Cowrie use random bytes for the `SSH2_MSG_KEXINIT` packet, but OpenSSH uses `NULL` characters for padding. The Binary Packet Protocol (BPP) of SSH is defined in RFC4253. Each packet consists of the packet length itself, the padding length, a payload, random padding and the Message Authentication Code (MAC). The BPP uses random padding to ensure that the total length of the packet TL_p is a multiple of the cipher block size (or of 8, whichever is larger). Section 6 of the RFC further states that the padding MUST consist of at least 4 bytes and these bytes SHOULD be random.

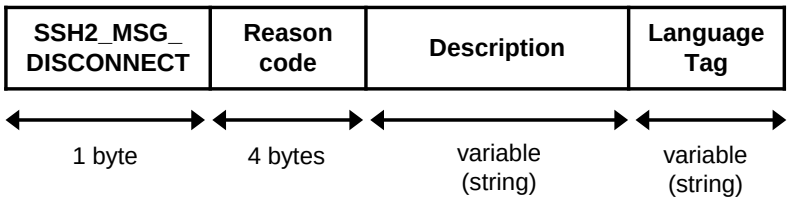


Figure 4.4: Structure of SSH2 Disconnection Message

We corresponded with the OpenSSH authors who told us that long ago they used random values but have changed to null padding bytes because this has no security implications either way [105]. This difference means that observing just one `SSH2_MSG_KEXINIT` packet is enough to distinguish OpenSSH from TwistedConch and hence determine if Kippo or Cowrie is responding.

4.5.2 Disconnection messages

A large source of divergences are the result of disconnection messages at various stages of the protocol exchange. We list six differences, each of which can serve as a distinguisher, and explain its origin.

The general structure of an SSH2 disconnection message is shown in Figure 4.4 and is defined in RFC4253 Section 11.1. It consists of one byte indicating the message type, a reason code ranging from 1-15 followed by descriptive text and a tag indicating the language being used. OpenSSH 7.5 implements more than 51 different disconnection messages and many of them are dynamically generated. Earlier versions of SSH are essentially similar.

Bad versions 1.0: After the TCP three-way handshake, the client sends its SSH version string. We find that changing the protocol version from 2.0 to 1.0, e.g. sending `SSH-1.0-OpenSSH FreeBSD\r\n` as the client version, results in TwistedConch terminating the connection with the disconnection message "bad version 1.0". In contrast, OpenSSH terminates the connection with "Protocol major versions differ.\n".

Bad versions 2.2: We observe that changing the protocol version from 2.0 to 2.2 results in the disconnection message "bad version 2.2" for TwistedConch, but OpenSSH does not raise an error and continues the protocol sequence by sending its `SSH2_MSG_KEXINIT` packet.

Missing mandatory key exchange algorithm: We find that if we complete the `KEXINIT` negotiation by sending our own (client) `SSH2_MSG_KEXINIT` packet offering only `diffie-hellman-group14-sha1` as a key exchange method that the earlier versions of TwistedConch disconnect with the message "couldn't match all kex parts". In contrast, OpenSSH servers proceed with the key exchange, the connection is not closed and no error message is sent. According to RFC4253 Section 6.5 `diffie-hellman-group14-sha1` is mandatory and OpenSSH is compliant. However, it has only been supported by TwistedConch since version 15.5 (2015-11).

Non-compliant SSH2_MSG_KEXINIT packet: We find that we can trigger disconnection messages by sending `SSH2_MSG_KEXINIT` packets that are not compliant with the SSH transport layer protocol. By omitting the packet and padding length, we force the SSH server to close the connection. Based on the disconnection message, we differentiate between Kippo ("bad packet length *", "Protocol mismatch.\n"), Cowrie

Table 4.6: Update Statistics for Kippo and Cowrie

		Scan 1 (SSH)		Scan 2 (SSH)	
Kippo	<2014-05-28	695	(24.4%)	546	(19.6%)
Kippo	<2015-05-24	211	(7.4%)	212	(7.6%)
Cowrie	<2017-06-06	1228	(43.2%)	950	(34.2%)
Cowrie	≤date of scan	710	(25.0%)	1071	(38.6%)

("Packet corrupt\n") and OpenSSH ("Packet corrupt\x00"). OpenSSH does have a similar error message to Kippo, but with a capital `B`, i.e. ("Bad packet length `*`"), where `*` is the packet length.

This intersects with a previously reported issue where eight carriage returns are sent as the client version string. Kippo returned the disconnection message "bad packet length 168430090" instead of replying with "Protocol mismatch" [97], but this was patched on 2014-05-28. However, the implementation of the patch catches all *bad packet length* errors, regardless of the current stage of the protocol exchange and thus also our non-compliant packet which has been sent after the version exchange.

Non-compliant packet: If we violate the specification of the BPP and construct a `SSH2_MSG_KEXINIT` packet so that $TL_p \pmod 8 \neq 0$, Kippo and Cowrie terminate the connection with, for example, the error message "bad packet mod (244%8 = 4)" where $244 \equiv TL_p$. OpenSSH provides no detail but terminates the connection with the generic reason "Packet corrupt".

Authentication failures: Lastly, we find a previously undescribed deviation at the authentication process. By default, OpenSSH limits the number of authentication attempts permitted per connection to six. Once this number is reached, the connection is closed. To do so, OpenSSH sends a disconnection message with the reason code 2 (`SSH2_DISCONNECT_PROTOCOL_ERROR`) and the description "Too many authentication failures". Kippo and Cowrie also send a disconnection message, but with reason code 14 and the description "too many bad auths". This divergence is particularly odd because reason code 14 is not used in any of the OpenSSH versions we examined (3.6, 4.4, 4.9, 5.4, 6.6, 6.7, 6.8, 6.9, 7.5) and OpenSSH has never used the "too many bad auths" string.

4.6 Discussion

Previously, especially for SSH honeypots, the main goal was to provide a realistic-looking shell for humans to interact with. But with the rise of botnets probing random servers and fast Internet-wide scanning, almost everything a honeypot observes is generated by automated scripts (see Chapter 3). Meanwhile, honeypot operators and developers have put little emphasis on the underlying protocols, but have relied on off-the-shelf libraries. More emphasis needs to be placed on identically implementing the lower layers of the networking stack. We have shown that no honeypot developer has implemented a protocol the same way as the server they impersonate. The RFCs that define protocols do not mandate every protocol detail and hence there are numerous differences 'on-the-wire' between entirely standards-compliant implementations. Ambiguities in RFCs are not the only source of divergence because code also evolves over time. For example, OpenSSH used random padding for its `SSH2_MSG_KEXINIT` in version 3.6p1 and earlier, but now uses clear padding with `NULL` characters. The developers of OpenSSH argue that the `SSH2_-`

Table 4.7: Top 10 ASNs used to host honeypots (latest scans)

CO	ASN	Organisation	Telnet	SSH	HTTP	Total
US	16509	Amazon.com	140	520	506	1166
JP	2500	WIDE Project	–	–	490	490
US	14061	Digital Ocean	162	189	139	490
FR	16276	OVH SAS	117	202	122	441
TW	4662	GCNet	15	2	254	271
TW	18182	Sony Network	2	–	256	258
US	15169	Google LLC	45	139	46	230
TW	9924	Taiwan Fixed	1	74	146	221
US	14618	Amazon.com	12	70	110	192
RO	43443	DDNET Sol.	30	–	155	185

`MSG_KEXINIT` packet is unencrypted and so random padding does not offer cryptographic benefits – but this change was missed by honeypot developers.

In the same vein honeypot operators need to carefully consider what Telnet terminal options, HTTP response codes, SSH version and authentication settings are sent – and far more care is needed with SSH host keys. Copying the same key to multiple machines not only links honeypots together, but attackers need only see the same key returned by multiple locations for suspicions to be raised.

4.6.1 Practical impact

The generic technique presented above allows the (automatic) generation of thousands of probes, any of which could be used to identify that a honeypot is running on a particular IPv4 address and thereby treat it differently than otherwise. Furthermore, we demonstrated that by identifying a probe with the maximum discriminatory power we can then rapidly scan the Internet to find thousands of honeypot instances. It will be easy to add scripts using these techniques into tools such as Metasploit. Since the probes do not leave meaningful log entries in any of our tested honeypots, operators will not be aware that their honeypot has been detected.

Once a honeypot is identified, it must be expected that it will be blacklisted by adversaries. Its value may drastically decrease, in particular in collecting data about large-scale attacks. The honeypot will need to be moved to a new IP address, perhaps a new hosting provider. But so long as it can be trivially spotted, the only real remedy will be to fix the flaw.

4.6.2 Countermeasures

Short term, honeypot operators will want to assess whether attackers have started to identify them by carefully inspecting their logs and looking for incomplete connections and repetitive disconnection messages. However, these messages commonly arise for other reasons and without recourse to packet level logging this will not be unambiguous evidence of fingerprinting. While not our current aim, our technique can be adapted to find and subsequently filter probes that induce no or even less logging than the probes we used.

Medium term, the developers of honeypots and libraries such as TwistedConch (where the SSH distinguishers we have identified reside) may mitigate some of the issues we have identified. Cowrie has already implemented a fix to use `NULL` characters for padding.

Long term, the only robust fix is to develop a new generation of honeypots that implement protocols using exactly the same code as the systems they set out to impersonate. Otherwise, as attackers include our methods in their scripts, low- and medium-interaction honeypots will have minimal value. This is undesirable because low- and medium-interaction honeypots are an extremely useful source of information, and not everyone is prepared to run high-interaction honeypots as they need to be carefully operated and maintained.

This new generation is not especially difficult to implement and we have already developed a modified OpenSSH daemon (sshd-honeypot) to be used in conjunction with Cowrie so there will be no difference in responses [142]. The daemon forwards all shell interactions to Cowrie where all commands are interpreted. The respective outputs are then returned to the client, appearing as if they originated from the daemon server itself. The honeypot requires little changes to both the OpenSSH daemon and Cowrie, and can be easily integrated in existing sensor networks. Cowrie remains the primary source where all brute force attacks, shell interactions and malware samples are collected.

4.7 Ethical considerations

We followed our institution's ethical research policy throughout, with appropriate authorisation at every stage.

We followed a strict responsible disclosure process and notified the relevant honeypot developers of our findings. We initially notified the developer of Cowrie on 2017-03-01 and subsequently the developers of the TwistedConch library used by Cowrie on 2017-03-14. Development of Kippo has ceased. Once we could fingerprint Telnet and HTTP honeypots, we also disclosed our results to the developers of TPwd, MTpot, TIoT, Cowrie, Dionaea, Glastopf and Conpot on 2017-10-16. As of June 2018, only one of the issues we have identified in Cowrie has been resolved. TwistedConch acknowledges that honeypots are an important use-case for their library, but they never promised byte-for-byte parity with OpenSSH. Based on the responses from the developers of Cowrie and TwistedConch, we are pessimistic that any further issues will be resolved.

The developer of Glastopf and Conpot agrees that fixes require a new architecture. Cymetria Research, the maintainer of MTPot, classified our findings as vulnerabilities as it is a "critical aspect of any honeypot" even though they say that some would argue otherwise. However, their view is that MTPot was intended to capture Mirai binaries and that it achieves that goal as Mirai does not try to fingerprint honeypots. The author of TIoT is not concerned about transport layer issues as his honeypot can be identified solely by its delivered content.

Before performing the Internet-wide scans to identify honeypot deployments, we thoroughly tested our scanner. For all scans we used the exclusion list maintained by DNS-OARC [43]. The host used for scanning runs a web page on port 80 so that people who are scanned can determine the nature of our experiment and learn our contact details. We also added reverse DNS entries to clarify the nature of the host. We ensured that local CERTs were fully aware of our activity. We received two complaints and respected their request to be excluded from further scanning.

4.8 Related work

Closest to this work is Bethencourt et al., who sent probes to ranges of IP addresses and observed changes of activity within published reports of sensor networks such as the SANS Internet Storm Center [17]. Thus, over time, they could enumerate all sensors for particular systems. However, this approach requires that sensors make their data publicly available; our technique does not require this.

Brumley et al. showed that by automatically building symbolic formulas from binaries they could find deviations in the protocol implementations of HTTP and NTP [22]. Similarly, AUTOPROBE generates fingerprints of malicious C&C servers through binary analysis [155]. Focusing on protocol reverse engineering, Comparetti et al. developed Prospex, a system to extract protocol specific information, but it may also be used to identify protocol deviations [30]. Our approach differs from all of them in that it is scalable to a variety of implementations, that we do not rely on binary analysis, and that it works at Internet scale.

Identifying vulnerabilities and characterizing network services by sending specifically crafted packets to network hosts and analysing their response is a long established practice. Popular tools include Nmap [87] and more recently ZMap [45]. Characterizing network hosts based on Internet-wide scanning has been previously performed for Industrial Control Systems (ICSs) and continuously for SSH. Censys.io performs weekly scans for all major protocols and grabs all publicly available information [46]. Similarly. Feng et al. performed an Internet-wide scan to characterize ICSs and their usage [51]. To get more accurate results, they trained a probability model and use a heuristic algorithm to exclude ICS honeypots. To do so, they use four characteristics including the number of open ports and HTTP configuration.

Focusing on SSH, Althouse et al. developed JA3, a technique for creating SSL/TLS client fingerprints based on SSL negotiation packets [5]. This allows JA3 to fingerprint known malware and client applications such as Trickbot, Emotet and the standard Tor client. To this end, Albrecht et al. present multiple vulnerabilities in SSH and perform an Internet-wide scan to obtain deployment statistics and estimate the impact of their new attacks [3]. Similarly, Gasser et al. showed that the rate of software updates for SSH is slow and that many SSH keys are reused on different hosts [56]. While our results will not explain all the hundred thousand duplicate keys they found, some belong to honeypots and not 'real' systems.

There has been a long arms race between finding ways to detect honeypots and camouflaging their presence [44, 120]. Successful attempts to fingerprint Kippo have been made by sending eight carriage returns as the SSH client version as previous versions of Kippo return the error message "bad packet length 168430090" instead of replying with "Protocol mismatch" [97]. In 2014, the SANS Technology Institute [120] reported that attackers issue the command `file /sbin/init` that returns dynamic content to fingerprint Kippo. In 2015, Cymmetria Research summarised such known cases for a variety of honeypots and outlined a list of recommendations [19]. However, unlike our automated and generic approach, these are individual findings affecting single honeypot implementations.

More recently, *Shodan* provides an online tool [123] that allows anyone to check whether a host is running a honeypot or a real industrial control system (ICS). While Shodan's effort is still work-in-progress and mainly targets ICSs, we find that all of the honeypots' IPs that we identified are believed by *Shodan* to be real systems.

4.9 Conclusion

We have presented a generic approach for systematically generating probes that can be used to find low and medium-interaction honeypots with just one or two packets, leaving minimal clues in the logging. Our technique is not only applicable to the SSH, Telnet and HTTP honeypots we discussed, but to a wide range of other protocols including SMTP, FTP, Modbus, S7 and SIP. We start by identifying a number of distinguishing probes and subsequently select the 'best' probe that minimises the effort to identify honeypots at Internet scale. We then present a number of techniques to determine the patch level of Kippo and Cowrie, the leading examples of medium interaction SSH honeypots.

The distinguishers we have identified result from design decisions to use off-the-shelf libraries (or in some cases newly developed code) to handle the protocol implementation – even though the libraries never promised byte-for-byte parity. This is a *class break* in that we do not believe that patching the current generation of honeypots can fully solve the problems we have identified. The potential damage is worrying. We will need a new generation of honeypots with new architectures – and those new honeypots will need to be installed on new IP addresses with new settings and with far more attention paid to the mechanics of deployment so that honeypot collections cannot be linked together.

Our impression is that honeypot authors believe that they are dealing with naïve human adversaries, but with the rise of fast Internet-wide scanning and the dominance of automated scripts probing servers, much more emphasis has to be put on ensuring that there is complete accuracy in the lower levels of the networking stack rather than just ensuring that humans can be fooled. This is, however, not an argument for high-interaction honeypots as many operators are unable to accept the risk that they pose or the effort required to monitor them to prevent them from doing harm. We need low- and medium-interaction honeypots too; we just need a new generation that is far less easy to identify.

Chapter 5

Counting outdated honeypots: legal and useful

Honeypots are intended to be covert, so little is known about how many are deployed or who is using them. We conducted several Internet-wide scans over a one year period to determine which particular versions of Kippo and Cowrie honeypots are being run on the Internet. By logging in to these SSH honeypots and sending specific commands, we not only revealed their patch status, but also show that many systems were not up to date: a quarter or more were not fully updated and by the time of our last scan 20% of honeypots were still running Kippo, which had last been updated several years earlier.

We further provide a detailed legal analysis and an extended ethical justification for our research to show why we did not infringe computer-misuse laws by accessing and logging in to honeypots.

The work presented in this Chapter was published in the 4th International Workshop on Traffic Measurements for Cybersecurity (WTMC '19) [145], and is in collaboration with Richard Clayton and Ian Walden.

5.1 Introduction

In the previous Chapter 4 we showed that low- and medium-interaction honeypots are almost invariably implemented as Python programs with the relevant Internet protocol layer being provided by an off-the-shelf Python library. This architecture is fatally flawed because the Python libraries have a large number of minor differences in their implementation of the Internet protocols compared to a 'real' system – particularly when it comes to the handling of errors [143]. This allowed us to develop an automated technique to identify the most valuable differences and then send a small number of malformed packets to a system and determine from the responses whether it was a real system or merely a honeypot. By sending our malformed packets to every host on the Internet, we were able to count how many medium-interaction honeypots are currently deployed for the SSH, Telnet, and HTTP protocols.

We extend the work presented in Chapter 4 as not only did we count the SSH honeypots that are deployed but we also 'logged into' them and issued commands to determine what version of software they were running. Even though the honeypots are generally operated by security professionals we found that a high proportion of them were significantly out of date and many had been deployed in a way that would leak that they were a honeypot.

Failing to run the latest version means that a proportion of attackers will rapidly determine that they are interacting with a honeypot, so its value will drastically decrease.

The reason we have not previously reported the configuration and updating issues was that in order to obtain detailed information it was necessary to login and issue a small set of shell commands. Our original work explaining our methodology and results was rejected by a leading conference on the basis that our interaction with the honeypots was illegal and hence our research was unethical. Leaving aside whether the one necessarily follows from the other, in this Chapter we start by exploring in detail the legal situation when accessing honeypot machines, where we firmly believe we have not broken the law, and then we set out why performing our research was also ethical. Having done that, we report our results.

5.2 Unauthorised access to computers

There is significant uniformity when it comes to legislation about 'cybercrime' because the statutes are less than 35 years old, there has been a tendency to replicate the wording of statutes from certain leading jurisdictions and there have been significant international harmonisation initiatives, primarily the Council of Europe Convention on Cybercrime (2001) [34]. So the usual caveats that 'your jurisdiction may vary' tend to be less relevant in this area.

5.2.1 Statutory text

Our conduct described in this Chapter can be broadly divided into two kinds: scanning for honeypots, and interacting with them. Scanning for identification purposes is a subset of the latter, but should be distinguished because it involves interaction with a broad range of systems, not only honeypots. In both cases, our conduct might appear to be 'unauthorised' or 'illegal' access – a form of computer integrity offence [149].

Under UK law, unauthorised access is an offence under the Computer Misuse Act 1990 (as amended). S. 1(1) states that *a person is guilty of an offence if–*

a) *he causes a computer to perform any function with intent to secure access to any program or data held in any computer, or to enable any such access to be secured;*
b) *the access he intends to secure, or to enable to be secured, is unauthorised; and*
c) *he knows at the time when he causes the computer to perform the function that that is the case.*

It is clear that element (a) is made out, so the issues to be determined are whether the access we intend to secure is 'unauthorised' and whether we have the requisite knowledge that our access is unauthorised. Element (b) primarily concerns the conduct of the person operating the honeypot (as 'victim'); while element (c) looks to our state of mind when accessing the honeypot (as 'perpetrator'). The statute sets out in s17(5) the meaning of unauthorised access: *access of any kind by any person to any program or data held in a computer is unauthorised if–*

a) *he is not himself entitled to control access of the kind in question to the program or data; and*
b) *he does not have consent to access by him of the kind in question to the program or data from any person who is so entitled.*

This interpretive subsection has been considered by the courts, focusing on the controller of the 'victim' system. In *Bow Street Magistrate and Allison (AP), ex parte US Government (HL(E)) [1999] 4 All ER 1*, it was held that access of the 'kind in question' can be refined considerably by the system controller, such that authority to view data may not extend to authority to alter data. Additionally, in *DPP v Lennon [2006] EWHC 1201 (Admin)*, the court held that a system controller's consent can "be implied from his conduct in relation to the computer".

In the USA the federal offence comes under '18 U.S. Code §1030 – Fraud and related activity in connection with computers'. This was originally enacted as the Computer Fraud and Abuse Act in 1986 but has been amended several times:

(a) Whoever . . . (2) intentionally accesses a computer without authorization or exceeds authorized access, and thereby obtains . . . (C) information from any protected computer;

So again, the issue is authorisation, but the question of whether the person obtaining access knew their access was unauthorised is implicit rather than explicit as in the UK. Even in Mexico and Taiwan – relevant because they appear to host many honeypots [96] – the applicable legislation follows a similar pattern to that of the UK and US. Under Mexico's Federal Penal Code, Article 211 bis 1, unauthorised access is only criminalised where the system is "protected by a security mechanism", which is itself an arguable assertion when operating a honeypot. Likewise, in Taiwan, Article 358 of the Criminal Code, provides that it is an offence to access a person's computer 'without reason' by 'breaking his computer protection'.

The 2001 Convention on Cybercrime (sometimes known as the Budapest Convention) is intended to harmonise cybercrime laws across the world [34]. Article 2 – "Illegal access" requires:

Each Party shall adopt such legislative and other measures as may be necessary to establish as criminal offences under its domestic law, when committed intentionally, the access to the whole or any part of a computer system without right. A Party may require that the offence be committed by infringing security measures, with the intent of obtaining computer data or other dishonest intent, or in relation to a computer system that is connected to another computer system.

Thus signatory states must have laws that forbid access 'without right', a concept which is seen as either referring to a person having the positive authority to engage in the conduct (e.g. granted by consent, contract or legislation) or a negative defence or justification that is recognised in law. Without right is not a term of art in UK or USA law, but "authorisation" is seen as being a comparable concept. Currently 62 states have ratified the convention with 4 more having signed but not yet ratified.

There is a further obligation within the European Union under Directive 2013/40/EU (Article 3: Illegal access to information systems): *Member States shall take the necessary measures to ensure that, when committed intentionally, the access without right, to the whole or to any part of an information system, is punishable as a criminal offence where committed by infringing a security measure, at least for cases which are not minor.* Again the terminology 'without right' is used, but the Directive permits member states to criminalise behaviour only when it is "not minor" and where a "security measure" was infringed.

To summarise the statutes, unauthorised access to computer systems is an offence in many jurisdictions although it may be in some parts of the world that minor infringements are not criminalised.

5.2.2 Implicit authorisation

Quite clearly, much authorisation of access to computer systems is implicit – web servers expect visitors to fetch pages and fill in forms to customise the material shown. Anonymous FTP servers provide access to files once a user has specified a username of '`anonymous`' and provided (by convention only) their email address as a password. No-one suggests that every visitor must first correspond with the system owner before viewing the front page of a website.

We argue that, first, the identification stage of our analysis, i.e. scanning IPv4 addresses, is lawful, primarily because the scanned systems implicitly authorise interaction by being connected to the Internet; while the absence of intent on our behalf to access all but the identified honeypots also avoids our conduct being considered illegal. Second, we think that someone who places a medium-interaction honeypot on the Internet has done so specifically because they wish people to send commands to it. In fact, they would be most disappointed if no-one was to interact with their honeypot at all. Since, by using the techniques outlined in Chapter 4, we can be certain that we are interacting with particular software implementations of medium interaction honeypots we are not accessing systems without authorisation.

To labour the point, we are not arguing that we can attempt to log into any old machine on the off-chance it might be a honeypot, but that our sure and certain knowledge that we are communicating with a honeypot changes everything: we are no longer guessing credentials in order to impersonate a legitimate user of the system but when presented with the password prompt we are sending a standard value in the sure and certain knowledge that the honeypot we are interacting with will present us with an impersonation of a shell prompt in a pretence that we have 'logged in'. Thus it makes no difference whether we successfully gain access to the system with the first attempt or perhaps after a number of iterations.

In terms of the statutory requirements outlined above, it would seem clear that our access is not unauthorised because the controller of the honeypot has intentionally made available a vulnerable system, implicitly permitting access of the 'kind in question', which we know at the time we access the system. Taken together, our conduct cannot constitute an offence of unauthorised access.

We have been unable to identify any earlier discussion of the exact issue we are concerned with, but two decades ago when honeypots were first being widely employed there was a belief (not borne out in practice) that people who accessed honeypots would be routinely prosecuted. One question that arose was whether the use of honeypots might be 'entrapment' – whether the deployment of a honeypot by a public law enforcement agency would have caused an otherwise innocent person to commit a crime.

In 2003 Walden and Flanagan considered the entrapment issue in a comparative law approach [150]. To summarise a lengthy analysis they concluded that operating honeypots would not be entrapment because anyone who broke into them had not been inveigled into doing so by anything more than being presented with the opportunity. This earlier work did not consider the legal position of a person who accesses a honeypot knowing that it was specifically designed to be so accessed.

5.3 Ethical analysis

Having established that our research was not illegal the question as to whether illegal research is always unethical becomes moot. We would disagree, but we would accept that there is a very high bar in such situations and that it would be essential to provide a detailed analysis as to how the law was unethical before starting to consider the research itself.

We followed our institution's ethics policy at all times and addressed some particular concerns that were raised about data storage. Nonetheless, for completeness, and given the history of trying to publish our results, we now set out the ethical justification for our research at longer length than might normally be expected.

Our overall view was that the research was in the public interest because demonstrating that it is possible to identify medium interaction honeypots rapidly at Internet scale should serve as a wake-up call for the people who deploy these honeypots – they need to upgrade to new implementations without the flaws. Accordingly we followed a responsible disclosure policy to ensure that the authors of the various honeypots learnt of our discoveries well before we made them public.

The first part of our research involved an Internet-wide scan to identify honeypot deployments. Before doing this we thoroughly tested our scanner to ensure it was working exactly as intended. For all of our scans we used the DNS-OARC exclusion list [43]. The host used for scanning ran a web page on port 80 so that people who were scanned can determine the nature of our experiment and learn our contact details. We also added reverse DNS entries for this host to clarify its purpose. Whilst using our scanner we ensured that local CERTs were fully aware of the nature of our activity. We received one complaint and respected their request to be excluded from further scanning.

Honeypot software is updated all the time, to make it more secure in the sense of removing distinguishing traits that criminals have identified in their efforts to determine whether they are interacting with a honeypot. As these distinguishers are identified the honeypot is extended so as not to be identifiable in that manner.

Our initial scan, sending very small numbers of packets, had already shown that some honeypots were not being kept up to date. To refine our understanding we decided to interact with the honeypot programs and issue a small number of shell commands, carefully chosen after examination of the revision history, to tell us rather more exactly how out of date the honeypot might be.

We were careful not to issue any commands that had the potential to impair the operation of the honeypot, which could be a separate offence to that of illegal access. We also took the ethical stance that we should not hide who we were – so we used a University IP address, gave it an appropriate reverse DNS entry and ensured that local CERTs would promptly pass on any reports to us so that we could explain what we were doing.

We believe that our contribution to the honeypots' overall traffic is negligible. After all, we send a minimum number of packets and honeypots are built to be probed, attacked and tested. At no point did we download files or try to find or use remote code exploitation.

We were also concerned that honeypot operators might consider our interaction with their systems to be worth their time investigating. So every successful SSH session was started and ended with the 'command' "Cowrie fingerprinting experiment, please ignore this session/connection". We intended that, by looking at the honeypots' logs, it would become evident to the honeypot owner that an experiment was occurring and that they need not view the activity as some new form of attack that they should spend effort on understanding.

Table 5.1: Update statistics for Cowrie and Kippo

	Scan 1: 2017-03		Scan 2: 2017-06		Scan 3: 2017-09		Scan 4: 2018-01	
Kippo < 2014-05-28	1384	(42.5%)	1519	(42.8%)	695	(24.4%)	546	(19.6%)
Kippo < 2015-05-24	659	(20.3%)	285	(8.0%)	211	(7.4%)	212	(7.6%)
Cowrie < 2016-09-05	385	(11.8%)	392	(11.0%)	134	(4.7%)	147	(5.3%)
Cowrie < 2016-11-02	—	—	556	(15.7%)	360	(12.7%)	422	(15.2%)
Cowrie < 2017-06-06	—	—	—	—	734	(25.8%)	381	(13.7%)
Cowrie ≤ date of scan	827	(25.4%)	799	(22.5%)	710	(25.0%)	1071	(38.6%)
Total	3255		3551		2844		2779	

In retrospect this message was unnecessarily cryptic, but since the responsible disclosure process was still in progress we did not want to explain to every honeypot operator that we had an easy way of identifying their system as a honeypot.

5.4 Fingerprinting SSH honeypots

Having disposed of the legal and ethical issues we can now move on to discussing what we learnt by interacting with the SSH honeypots that we identified.

The previous Chapter sets out our observation that the architecture of medium-interaction honeypots is invariably that of a specially written emulation program which uses a general purpose library to provide the relevant protocol layer. We developed a method of identifying the best 'probe' we could send to a server which would reveal whether we were interacting with a library or a real system – and we reported on the number of honeypots that we were able to identify in a series of scans of the Internet.

5.4.1 Identifying SSH honeypot patch levels

Note that for clarity of exposition within this section we will use terms such as login, authentication and passwords, although when dealing with honeypots, as explained in Section 5.2.2 above, we are merely just sending strings to a Python program that is impersonating a vulnerable system with a view to having it change internal state and start executing its impersonation of the 'bash' shell.

In Chapter 4, we showed that without authenticating to the honeypots, solely based on the protocol interactions, we are able to get a estimate of the honeypots' patch level. However, to get more insights on how honeypots are actually configured, e.g. what authorisation settings are used or what hostname is configured, the only viable option is to login to the honeypots and issue commands.

After the careful ethical considerations described in Section 5.3, we decided to authenticate with a list of passwords to all of the Cowrie honeypots and to issue three simple shell commands in order to better estimate the patch level of the host system (Table 5.1). These commands were: `uname -a`, to get more information about the host system, `ls -d`, added on 2016-09-05, and `tftp`, added on 2016-11-02. The command `tftp` is issued without any arguments and solely to check if the command is implemented. After hearing from us the developer of Cowrie implemented a fix to null pad packets on 2017-06-06 which is a useful intermediate date for determining the age of Cowrie honeypots. The analysis of the version information we obtained is given in Section 5.5.

5.4.2 Measurement setup

We aimed to visit all SSH servers on the Internet with a custom scanner written in Python3 which would send probes to determine whether we had found an instance of Kippo or Cowrie and if so which version was being run. We first used ZMap to perform a one-packet scan at 30mbps sending TCP SYN packets to port 22 (the well-known port number for SSH) using the IP address exclusion list maintained by DNS-OARC [43]. In total we scanned 3 336m IPv4 addresses, 78% of the IPv4 address space. We determined which IPv4 addresses responded successfully with a SYN-ACK packet and thereby efficiently identified the presence of SSH servers.

We connected to the SSH servers and checked the version to determine if it was claiming to run OpenSSH. We then checked whether the server behaved identically to OpenSSH using the method outlined in Chapter 4. We did not only rely on a single probe, but also sent the second-best probe, and removed the random padding, packet length and padding length for both responses so that we unambiguously knew when we had identified an instance of Kippo or Cowrie. As explained in further detail in Chapter 4, both probes are sent before any authentication credentials are used. To ensure that we only authenticated to machines that we had classified as honeypot, we checked the SSH host keys ('host fingerprint'), which are intended to be unique, before each authentication attempt. This ensures that we were not unintendedly authenticating to real systems, for example, because of IP address churn. For the Cowrie machines (and only these machines), we used a custom script written in Python3 to try to authenticate. Having done so we issued the commands `uname -a`, `ls -d` and `tftp`. Note that Kippo does not implement these commands.

5.5 Results

5.5.1 Authentication configuration

The results of our authentication attempts are summarised in Table 5.2.

For the first run, conducted on 2017-03, we used the username `root` and just 6 passwords: `123456, root, admin, password, PASSWORD, cisco`. For 859 of 1 212 Cowrie honeypots (70.9%.) the authentication was successful and we received a `SSH2_MSG_USERAUTH_SUCCESS` packet indicating that the password was deemed correct.

We re-ran our script three weeks later, but instead of 6 passwords, we used the 500 most common passwords seen in authentication attempts to our own research SSH honeypots. For each connection, we kept trying passwords until we received a "too many bad auths" disconnection message with reason code 14. We immediately reconnected and continued down the list until we were logged in, had tried all 500 passwords, or received an error message. In our second run, we successfully logged in to 794 of 1 212 Cowrie honeypots

Table 5.2: Authentication attempts for Cowrie honeypots

Outcome	Scan 1: 2017-03		Scan 2: 2017-06	Scan 3: 2017-09	Scan 4: 2018-01
	6 passwords	500 passwords	500 passwords	500 passwords	500 passwords
successful login	859 (70.9%)	794 (65.5%)	1165 (66.7%)	1347 (69.5%)	1578 (78.1%)
all passwords failed	110 (9.1%)	136 (11.2%)	187 (10.7%)	195 (10.1%)	223 (11.0%)
connection timed out	49 (4.0%)	110 (9.1%)	41 (2.4%)	43 (2.2%)	7 (0.3%)
other errors	194 (16.0%)	172 (14.2%)	354 (20.2%)	353 (18.2%)	213 (10.6%)

(65.5%). For 136 (11.2%) of the honeypots, all 500 passwords were successfully attempted, but failed. Table 5.2 reports the other outcomes. We repeated this measurement for the Cowrie instances we identified in the second, third and fourth scans, but only using the 500 password approach. We observed that the number of successful logins remains fairly stable as we were able to successfully login to 1 165 (66.7%) honeypots in the second scan, 1 347 (69.5%) honeypots in our third scan and to 1 578 (78.1%) in our fourth scan.

In all four scans, a significant number of honeypots rejected all the passwords we tried. We conclude that around 10% of the honeypot operators are only interested in obtaining password and username combinations, but not in providing a shell and in letting adversaries execute commands; though it is possible that they are just being more selective about the credentials they will accept. For a few honeypots (0.3% to 9.1%), the connection timed out (after 6 seconds). For the remaining honeypots we received various error messages including "Connection refused" and "Connection reset by peer".

We cannot be sure why we managed to login to some honeypots but got no further. It may be that firewalls or hosting providers interfered in the process of establishing a connection. A more likely explanation is that, because one or more honeypots report 'malicious' activities they see to central databases this caused other honeypots to refuse to communicate with us. In particular we found that the IP we used for scanning and logging in to honeypots was added to various blocklists including `blocklist.de` which is used by the intrusion prevention system `fail2ban`. This might also explain why trying 500 passwords yielded fewer successful logins than 6 passwords. The attempt using 500 password was made three weeks after the initial attempt with 6 passwords, at which time blocklists and intrusion prevention systems could have been updated to include our IPv4 address and blocked subsequent connections.

5.5.2 Set-up options of SSH honeypots

The first two options of the configuration file of both Kippo and Cowrie are the SSH server version string and hostname. We find that honeypot operators seldom change default configuration options.

Our first scan found 61 different SSH version strings, but in 83% of the cases it was a default. The Kippo/Cowrie default value accounts for 2 046 (72%) of the honeypots, but additionally, 312 (11%) Kippo honeypots report the version string `SSH-2.0-OpenSSH_-5.5p1 Debian-4ubuntu5`, which is the default set by the deployment script of the Modern Honey Network [6], an open source honeynet management platform. We observed some change in the second scan where 1 839 (66%) honeypots had default SSH version strings. In particular advertising versions equal or greater than OpenSSH 7.2 are becoming more popular.

In an attempt to further confirm our hypothesis that honeypot operators often use standardised configurations we issued the command `uname -a` on all the Cowrie honeypots to which we could successfully login in our first scan (see Section 5.5.1). By default, Cowrie will return `Linux [hostname] 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86_64 GNU/Linux\r\n`.

The default hostname Cowrie places into this string is `svr04`, but that hostname is only configured for 64 (3.3%) of the honeypots and we find 171 different hostnames. However, although the Cowrie default hostname is not being widely used, our hypothesis about not changing defaults is in fact confirmed because we find that many hostnames

have been set in a default manner by deployment scripts. The most common hostname is `debnfwmgmt-02` (14.6%), followed by `router` (5.5%) and `pos01` (5.3%).

What has occurred is that `debnfwmgmt-02` was the default hostname for Cowrie when it is used in T-Pot 16.03 whereas `debnfwmgmt-01` was used for Kippo in T-Pot until it was replaced by Cowrie.[1] It follows that 296 of Cowrie honeypots are extremely likely to be part of T-Pot (and hence T-Pot has a significant 'market share') – which in turn means that it is quite likely that other servers hosted on the same IPv4 address are also (T-Pot installed) honeypot instances.

5.6 Discussion

We found that many honeypot operators are relying on standardised deployment scripts, docker containers or public configuration files. Since various aspects of the configuration reveal that the system is a honeypot, this is clearly suboptimal. The fix is not, however, to eschew the use of standardised deployment systems, but for those systems to be far better engineered so that they do not allow honeypots to be fingerprinted or for IPs to be linked together. In response to our findings, Deutsche Telekom acknowledged this issue and rapidly changed how they configured the honeypots in their T-Pot collection.

We operate our own research SSH honeypots which are not based on Kippo or Cowrie [142]. As of publication, we have not observed attackers using the techniques we have developed to determine that they are interacting with a honeypot. However, we and others do observe that the command `uname -a` is one of the most popular commands the attackers' scripts issue, presumably because they need information about the host system and its architecture [112]. From that point of view T-Pot made an unwise choice because any search engine will immediately reveal that `debnfwmgmt` is part of the hostname T-Pot uses for Kippo and Cowrie honeypots.[2]

5.7 Conclusion

In Chapter 4, we showed how the use of off-the-shelf libraries in medium interaction honeypots allows us to fingerprint SSH servers and unambiguously identify instances of Kippo and Cowrie. In this Chapter, we have extended this analysis by determining which versions of Kippo and Cowrie are being run at the time of our scans.

We were surprised to see how out-of-date many of the honeypot deployments were. It is well-known that 'ordinary users' find it a challenge to keep their systems patched up-to-date, but we would expect that the majority of honeypots are deployed by security professionals and hence would be being actively looked after. In particular, most of the reason for updates to these honeypots has been to counteract fingerprinting tricks by criminals who wish to avoid interaction with honeypots. Failing to update makes the honeypots much less useful.

Finally, we hope that our detailed account of some aspects of the legal and ethical framework of interactions with honeypots will enable more research in the future.

[1] `https://github.com/dtag-dev-sec/tpotce_archive`

[2] When we first saw this string we found that the *only* results returned to us by a Google search were within the T-Pot source files.

Chapter 6

Honware: A virtual honeypot framework for capturing CPE and IoT zero days

Existing solutions are ineffective in detecting zero day exploits targeting CPE and IoT devices. In this chapter, we present honware, a high-interaction honeypot framework which can emulate a wide range of devices without any access to the manufacturers' hardware.

Honware automatically processes a standard firmware image (as is commonly provided for updates), customises the filesystem and runs the system with a special pre-built Linux kernel. It then logs attacker traffic and records which of their actions led to a compromise. We show that our framework is better than existing emulation strategies which are limited in their scalability, and that it is significantly better both in providing network functionality and in emulating the devices' firmware applications – a crucial aspect as vulnerabilities are frequently exploited by attackers in 'front-end' functionalities such as web interfaces. Honware's design precludes most honeypot fingerprinting attacks, and as its performance is comparable to that of real devices, fingerprinting with timing attacks can be made far from trivial.

We provide four case studies in which we demonstrate that honware is capable of rapid deployment to emulate devices much better than traditional honeypots, and thus capture the exact details of attacks along with malware samples. In particular we identified a previously unknown attack in which the default DNS for an ipTIME N604R wireless router was changed. We believe that honware is a major contribution towards re-balancing the economics of attackers and defenders by reducing the period in which attackers can exploit zero days at Internet scale.

The work presented in this Chapter will appear in the 14th APWG Symposium on Electronic Crime Research (eCrime '19) [144], and is in collaboration with Richard Clayton.

6.1 Introduction

The emphasis on automation and the nature of CPE and IoT devices themselves together mean that vulnerabilities and exploits not affecting device functionality are likely to remain unnoticed by their owners. Recent DDoS attacks which used inadequately secured devices [7, 137, 71] further highlight that existing defences are slow to detect zero day exploits and capture attack traffic. This means that attackers have considerable periods

of time to find and compromise vulnerable devices before the attack vectors are well understood and mitigation is in place [16].

Now that the Mirai source code has been leaked and is well understood, it is straightforward to build a honeypot that emulates a vulnerable device by sending appropriate strings back to scanners. But without source code, or reverse engineering the malware binaries, this type of honeypot is hard to construct. It is a significant challenge to monitor large numbers of attackers who are going after a wide range of devices using different attack techniques, some of which may be previously unknown 'zero days'.

Meanwhile, it has become feasible to scan the whole IPv4 address space for vulnerable devices with modest investment (Chapter 2). In 2003 Spitzner argued that honeypots "get little traffic" and "collect small amounts of high-value data" [128]. However, this observation was from a time when attacks were generally performed by humans. As shown in Chapter 3, since the rise of botnets almost all activities that honeypots observe are performed by automated scripts.

Previous research includes Firmadyne [26], an analysis system that runs embedded firmware and subsequently provides dynamic analysis capabilities, and IoTPOT [92], one of the first generic high-interaction honeypot tailored to impersonate IoT devices. It supports eight architectures including ARM, MIPS and X86 and aims to return appropriate strings to connections on port 23 (Telnet). If the command is unknown, it tries to run the command in a generic sandboxed environment to infer the appropriate return string(s). In 2017, Guarnizo et al. [60] presented a "scalable high-interaction" honeypot platform called SIPHON which is based on physical devices. They exposed six security cameras, one networked video recorder and one networked printer through a distributed architecture on a range of IPv4 addresses.

All these approaches have critical shortcomings: Firmadyne is built for dynamic analysis, but not to monitor a large number of attackers and thus not to be connected to the Internet. IoTPot does not use firmware images of real devices and thus it is a generic representation of a vulnerable platform which will fail to detect new attack patterns; SIPHON needs physical devices connected to the Internet to capture attack traffic – an expensive endeavour limited in its scalability.

We present honware, the first flexible and generic framework to efficiently and effectively deploy honeypots for networked devices on the Internet to log attacker traffic and their actions. Instead of buying CPE or IoT devices and running them as honeypots, honware utilises device firmware images (which are widely available for download) and a special pre-built Linux kernel to emulate device behaviour within a virtual environment. In particular, it is easy to deploy all the available firmware versions for a particular device so as to understand which are vulnerable to a particular attack.

Overall, we make four main contributions:

- We present new techniques (and improve on existing work) to allow honware to run standard firmware images without needing custom hardware.
- We show that honware is superior to existing emulation strategies, making significant improvements in scalability, in the provision of network functionality, and in emulating firmware applications.
- We perform extensive measurements to show that the performance of honware is comparable to real devices and that honware is not susceptible to trivial fingerprinting based on timing attacks.
- We present four examples to show the success of honware in identifying real-world

attacks which had been hard to capture with the traditional approach of low-/medium-/high- interaction honeypots.

6.2 Virtual honeypot framework

Honware uses the firmware images provided by CPE and IoT manufacturers, employing Quick Emulator (QEMU) to run code for different CPU architectures (x86-64 PCs, ARM, MIPS and PowerPC) on a single host machine. Although parts of the firmware will be closely linked to the hardware of a particular device, the Linux kernel and the device driver APIs are substantially the same across many different devices. Honware decouples the execution of the firmware from the underlying hardware by the use of a custom kernel.

Figure 6.1 shows the four main parts of honware: a host operating system and kernel, QEMU, a custom kernel, and the firmware filesystem (extracted from the firmware image) which contains applications such as Telnet and web servers.

QEMU is the de facto standard for full machine emulation and it provides the necessary functionality to connect the honeypot to the Internet. However, QEMU cannot be used for off-the-shelf emulation of CPE and IoT firmware images. The firmware, with its own kernel, will try to communicate with the hardware, but its absence means that this communication will fail. For example, any access to non-volatile memory (nvram) to read configuration files or an attempt to retrieve the MAC address from hardware will fail. Thus we run QEMU with our own customised kernel and the extracted filesystem on top of an host operating system such as FreeBSD or Linux Ubuntu.

Honware currently uses three custom kernels since these are sufficient for handling very large numbers of firmware images. These are the Linux kernels 2.6.32.70 for MIPS little endian (mipsel) and MIPS big endian (mipseb), and 4.1.52 for ARM. We purposely used older kernels as these are most prevalent across CPE and IoT firmware images and support for newer kernels is rarely required. The MIPS kernels are compiled with MIPS Malta and with the MIPS32 CPU release version 2. MIPS Malta is particularly useful as it supports the emulation of PCI and LAN functionalities that are inherently required for the emulation of networked equipment. The ARM kernel is compiled with the architecture Versatile Express and `VIRT/MULTI_V6_V7` as a *dummy* CPU architecture. As the Versatile Express platform is meant for development and rapid prototyping, it provides a wide range of hardware support. We further configure the kernel to support various networking features such as VLAN and WLAN.

The emulation of firmware images is performed in three stages and is fully automated: A) extracting the filesystem from the firmware image, B) modifying the filesystem to allow for virtualisation and C) running it in QEMU with one of our pre-compiled kernels. Honware will typically process a firmware image (typically a .zip or .rar file) and have the honeypot ready to run within one minute.

The focus of honware is not to advance state-of-the-art sandbox anti-evasion techniques, but to develop a tool that enables the rapid construction of honeypots for a very wide range of devices – and for those honeypots not to be susceptible to remote fingerprinting attacks based on protocol deviations [142] or self-revealing properties [96]. It is not attempting to prevent attackers with local access from determining whether the system is a honeypot or a real device.

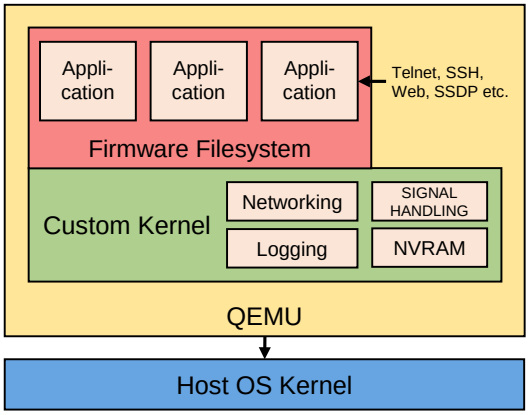


Figure 6.1: Honware architecture overview: Honware consists of four main parts, a host operating system and kernel, Quick Emulator (QEMU), a custom kernel, and the firmware filesystem itself which contains specific applications such as Telnet and web servers.

6.2.1 Automated firmware extraction

Firmware images for routers, ADSL modems and IoT devices are widely available online and many manufacturers provide regular updates on their websites. Honware uses these images as input, usually in a compressed format, and extracts the firmware filesystem. Any supplied kernel is ignored as running it within QEMU would not allow us to interfere with the execution of the filesystems applications and services – a necessity to run the firmware image as a honeypot and in fact, to decouple the firmware image from the underlying hardware.

To extract the filesystem, we use binwalk [18] and recursively look for the Linux filesystem structure. We identify Linux filesystems by determining whether a folder contains the necessary root structure including `bin`, `usr` and `proc`. This has proven to be challenging as not all firmware images are packed in the same way. Occasionally they include multiple suitable filesystems such as one firmware image for the upgrade, one for the previous version and one for a factory reset. We also found instances in which a secondary filesystem is mounted during the boot process, for example to provide scratch space. In such a case, we only consider the filesystem where the init process resides as without this process, the firmware will not boot at all. Although extraction is intended to be completely automatic, honware does allow manual selection of which filesystem to use should this be needed.

We then use `qemu-img` to create a 2GB raw file, subsequently create an ext2 filesystem and copy the root folder structure including all files and binaries. We infer the CPU architecture by reading the ELF header of the biggest binary of the filesystem, typically Busybox. We further extract all certificate (`.pem`) files from the firmware images so that we can use tools such as Wireshark to decrypt, for example, HTTPS traffic to the web server.

It is more challenging to decrypt SSH traffic as the Diffie-Hellman key exchange uses not only a static key, but also a session key. Retrieving the relevant session key is not straightforward and requires locating a particular memory structure in guest memory. We decided not to tackle this particular issue as we can log executed commands in our custom kernel. However honeypot operators could consider SSLSNOOP [69] or related techniques to collect the raw traffic.

6.2.2 Customised pre-built kernel

We now consider the various components of our customised Linux kernel and explain how we have managed to improve on earlier systems such as Firmadyne. In particular, previous work has ignored the kernel's signal handler as way of ensuring that applications continue running and do not terminate silently, and no previous attempts have been made to support out-of-tree kernel modules. Furthermore, our kernel does not solely rely on the default configuration of the guest system to ensure network connectivity, but can force a particular network configuration to be used. In addition, honware supports up to four ethernet device for the ARM little-endian platform whereas Firmadyne only supports a single ethernet device.

6.2.2.1 Honeypot logging and module loading

In order for honeypots to provide insights into how systems are attacked and to monitor subsequent actions, it is essential to have extensive logging capabilities. To get details of all executed programs and commands with the appropriate time stamps, process ids and contexts, we modified the kernel function `do_execve`. For each connection, we create a new session id under which we log the invoked programs and their details. Commands which are not associated with network connections are logged with the default session id 0. This means that the boot process, cron jobs and other processes that are executed by the firmware image itself can be clearly distinguished from actions triggered from the outside.

6.2.2.2 Signal interception

One central problem we encountered is that the init process and various applications frequently terminate silently or with generic error messages. Applications may terminate because of wrong or missing nvram values, incorrect hardware emulation or missing kernel features. To mitigate the effects of this problem, we modified the signal handling in the kernel to 1) not allow the kernel to terminate the process, for example by means of the default signal handler and 2) not allow the program to terminate itself with, for example, a SIGABRT signal. This means that applications and kernel modules continue their execution irrespective of what signals are sent. To achieve this we modified the kernel function `get_signal()` which is called by `do_signal()` and is responsible for signal handling in the kernel.

It has proven particularly useful to intercept SIGNAL 6 (SIGABRT). We find that SIGABRT is used to detect broken constraints, for example, to indicate missing license keys or the absence of hardware modules. In particular, devices with Broadcom chips look for a variety of settings in nvram to differentiate between hardware versions. If these settings cannot be found, the init process or the calling application wants to terminate itself to prevent further execution. In addition to SIGNAL 6, we also intercept SIGNAL 11 (SIGSEGV) and SIGNAL 7 (SIGFPE). The latter two mitigate the problems caused by missing nvram values that are not absolutely necessary for running applications. SIGPFE is typically sent for floating point errors such as when the application attempts to divide by 0. We believe that the absence of nvram leads to the use of zero values and various mathematical operations fail.

For a small number of firmware images' ignoring signals leads to indefinite loops and very high CPU usage, however in many cases the programs appear to continue running

successfully. Thus intercepting signals is a trade-off between ignoring (some) values, while at the same time making the emulation useful and viable.

6.2.2.3 Module loading

A number of firmware images are shipped with custom modules to implement device specific functionality such as cryptographic operations or support for custom hardware. To avoid undesired behaviour and kernel crashes, our customised kernel will not load incompatible modules, even if instructed to (`modprobe -f`). However, the kernel will accept modules with different `vermagic` strings and does not require exact matches. Vermagic strings are typically used to ensure that the modules were built and configured for the same kernel version, but as the vermagic strings greatly differ between manufacturers and firmware versions, we ignore them. For example, if a module has the vermagic string `2.6.22-routeros` our newer kernel with the version 2.6.32.71 will attempt to load it. This problem could be avoided by re-compiling the kernel for every individual firmware image with the correct vermagic string, but this would be a significant barrier to deploying honeypots in a timely manner and we find that, in practice, our approach works very well.

6.2.2.4 NVRAM

To store and later retrieve device-specific configuration parameters, device manufacturers often use non-volatile memory (nvram). Chen et al. [26] looked at 23 035 firmware images and found that more than half of them accessed nvram, for example to handle configuration information during the boot process. Typically firmware images set a variety of nvram values during the boot process and subsequently read these nvram values with `nvram_get`. To emulate nvram, we use the approach first mentioned by Heffner and later developed by Firmadyne: we set the environmental variable `LD_PRELOAD` to the path of our own nvram implementation, so that our file will be loaded before any other (firmware) library [42]. This means that we reliably intercept calls to `nvram_get` and `nvram_set`.

To extend Firmadyne's approach, we implement a script that automatically reads the kernel logs, detects missing nvram values, re-compiles the necessary shared library and updates the filesystem for the next time the firmware is run. This can be an iterative process as setting certain nvram values may cause other nvram values to be accessed and those could also be missing. In our evaluation (in Section 6.3) which compares honware with Firmadyne, we used static values and did not iteratively look for missing values. However, as shown in Table 6.1 and 6.2, we achieve far better results than Firmadyne in terms of correctly emulating the firmware, its applications and in inferring the network configuration.

6.2.2.5 Network configuration

Providing network functionality is a fundamental prerequisite for honware since without it the emulation cannot be connected to the Internet and probed by attackers. To infer the network configuration, honware parses the kernel logs for the initial configuration of the bridge device, typically named `br0` or `ra0`. It then looks for `ifconfig` commands which configure the bridge, and for any `addif` command which adds one or multiple network interfaces to that bridge. Subsequently, it extracts the IP address and creates a

tap interface in QEMU, and sets the associated route and iptables rules on the host to forward all traffic to the inferred network interface.

If the network does not work with the inferred configuration, we re-run the emulation and overwrite the firmware's default network configuration with a custom configuration. This is achieved by placing the configuration into `/sbin/boot.sh` which will be executed by the kernel during the boot process. The custom configuration starts by shutting down the network interfaces `(eth_x)` and any wrongly configured bridges. Then it assigns appropriate local static IP addresses and adds the network interfaces back on to the bridge, setting up a route so that the guest network interface and the host's network interface can exchange packets. Finally, the script removes all firewall rules so that we can be certain that traffic is not interfered with. If the network still does not respond to ICMP echo request packets, we consider the firmware image not *network reachable* (and our attempt to create a useful honeypot has been unsuccessful).

To fully test network reachability we use `ping` and `nmap`. In particular we use nmap[1] to do a port scan of the most common ports including port 22 (SSH), 23 (Telnet), 80 (HTTP), 443 (HTTPS) and 1900 (UPnP). The results of this evaluation with 8 387 firmware images can be found in Section 6.3.1.

6.2.3 Filesystem modifications

After extracting the filesystem, we have to modify it for emulation. First, we add the module for nvram emulation (Section 6.2.2.4). Second, we modify `do_execve` to execute, if present, `/sbin/boot.sh` through the kernel function `call_usermodehelper`. This allows us to execute custom scripts and commands for a particular firmware image without having to change the pre-built kernel or perform complex modifications to the firmware filesystem. This gives honeypot operators flexibility to, for example, specify additional network interfaces, execute applications with particular configuration options or customise the firewall to their needs.

Unsurprisingly, many emulated firmware images do not configure static IP addresses, but use DHCP. As we choose not to emulate access to a DHCP server, attempts to obtain an IP address would fail and the emulated firmware would not be reachable over the network. To address this, we use busybox-static, a statically linked version of Busybox which is available for mipseb, mipsel and ARM as well as for many other architectures. We copy this Busybox binary into the guest filesystem and use it to set up a bridge, attach one network interface to it, configure both appropriately, and set up a default route (see Section 6.2.2.5). This approach yields considerably better results than relying on the default configuration of the guest system itself, as was done, for example, by Firmadyne.

6.2.4 Emulation

After the extraction and preparation of the filesystem, honware invokes QEMU to start the emulation. The honeypot is tested to see if it responds to ICMP echo request packets over the local network. If this is successful the necessary interfaces, routes and host firewall rules for connecting the honeypot to the Internet are created. We pre-route incoming packets on the host ethernet interface to the QEMU tap interface and post-route packets

[1] `nmap -F -St -Su ipaddress`

back to the host. By specifying which ports are handled this way, operators are able to customise their honeypots and only expose ports to the Internet that are of interest.

Honware can be configured so that the firmware emulation only runs for a certain period of time or *forever*, i.e. until stopped by the user. While honware is running it outputs kernel log information, details of incoming network connections, and all invoked commands with the relevant time information and writes this all to log files.

6.3 Evaluation

The evaluation of our framework is threefold. First, we compare honware with Firmadyne [26] in terms of extracted firmware images, network reachability and number of emulated services. We do this by obtaining and running the identical firmware images that they used. Second, we provide four case studies where we demonstrate that honware is capable of rapidly emulating devices to capture not only malware samples, but to emulate advanced device behaviour which is not feasible with traditional honeypots. Third, we perform extensive measurements to show that the performance of honware is comparable to real devices and that honware is not susceptible to trivial fingerprinting based on timing attacks.

6.3.1 Extraction, network reachability and services

To measure how well honware extracts firmware images, configures the network and emulates services, we obtained the list of firmware images used in the evaluation of Firmadyne and downloaded all images that are still accessible on the URLs provided. As of March 2019, 8 387 of 23 035 images (36.4%) can still be downloaded. The list includes a variety of firmware images for CPE and IoT devices such as ADSL modems, routers, NAS systems, web cameras and smart power plugs. Unfortunately the authors of Firmadyne lost their database that would have allowed us to map the individual images to the outcomes *network reachable* and *listening services emulated*, so we ran Firmadyne ourselves over the 8 387 remaining firmware images to be able to compare results.

6.3.1.1 Extraction

As shown in Table 6.1, honware appears better in extracting firmware images than Firmadyne. In total, we successfully extracted 4 650 of 8 387 available images (55.4%) compared to 2 920 for Firmadyne (34.8%). We both use binwalk (the de-facto standard tool for firmware extraction) and for most manufacturers our results are very similar. Our significantly better results for Synology are probably because we allow filesystems up to 2GB in size (1GB for Firmadyne) – Synology firmware images are quite large, compressed to an average size of 133MB. That is, we suspect that Firmadyne failed to populate the filesystem correctly because of space constraints. The firmware images for which both Firmadyne and honware were unable to extract a filesystem are encrypted, have a proprietary way of packaging images or are simply updates – and therefore important folders and binaries are missing.

Table 6.1: Comparison between honware and Firmadyne: We obtained the list of firmware images (23 035) used in the evaluation of Firmadyne (2016-02) and downloaded all that remained accessible (8 387 in 2019-03). We used Firmadyne and honware to extract these images and test their network reachability by sending them ICMP echo request packets.

#	Brand	Available (2019-03/2016-02/Δ)		Extracted (honw./firm./Δ)		Network reach. (honw./firm./Δ)	
1	Actiontec	0/14	14↓	–	–	–	–
2	Airlink101	0/15	15↓	–	–	–	–
3	Apple	0/9	9↓	–	–	–	–
4	Asus	1/3	2↓	1/1	←	1/0	1↑
5	AT&T	3/25	22↓	0/2	2↓	–	–
6	AVM	0/132	132↓	–	–	–	–
7	Belkin	123/140	17↓	49/49	←	9/0	9↑
8	Buffalo	97/143	46↓	6/7	1↓	2/1	1↑
9	CenturyLink	13/31	18↓	7/4	3↑	7/0	7↑
10	Cerowrt	0/14	14↓	–	–	–	–
11	Cisco	0/61	61↓	–	–	–	–
12	D-Link	1443/4688	3245↓	537/498	39↑	272/115	157↑
13	Forceware	0/2	2↓	–	–	–	–
14	Foscam	44/56	12↓	5/5	←	–	–
15	Haxorware	0/7	7↓	–	–	–	–
16	Huawei	13/29	16↓	0/3	3↓	–	–
17	Inmarsat	0/47	47↓	–	–	–	–
18	Iridium	0/17	17↓	–	–	–	–
19	Linksys	32/126	94↓	26/26	←	15/1	14↑
20	MikroTik	4/13	9↓	–	–	–	–
21	Netgear	1396/5280	3884↓	639/629	10↑	384/187	197↑
22	On Networks	0/28	28↓	–	–	–	–
23	Open Wir.	0/1	1↓	–	–	–	–
24	OpenWrt	756/1498	742↓	714/705	9↑	674/0	674↑
25	pfSense	214/256	42↓	–	–	–	–
26	Polycom	612/644	32↓	0/24	24↓	–	–
27	QNAP	8/464	456↓	–	–	–	–
28	RouterTech	0/12	12↓	–	–	–	–
29	Seiki	0/16	16↓	–	–	–	–
30	Supermicro	0/150	150↓	–	–	–	–
31	Synology	1977/2094	117↓	1866/239	1627↑	–	–
32	Tenda	6/244	238↓	4/3	1↑	2/0	2↑
33	Tenvis	9/49	40↓	6/6	←	6/4	2↑
34	Thuraya	0/18	18↓	–	–	–	–
35	Tomato	362/2942	2580↓	362/362	←	217/0	217↑
36	TP-Link	463/1072	609↓	171/171	←	147/95	52↑
37	TRENDnet	336/822	486↓	134/100	34↑	87/37	50↑
38	Ubiquiti	26/51	25↓	20/19	1↑	11/0	11↑
39	u-blox	0/16	16↓	–	–	–	–
40	Verizon	0/37	37↓	–	–	–	–
41	Western Dig.	0/1	1↓	–	–	–	–
42	ZyXEL	449/1768	1319↓	103/67	36↑	69/20	49↑
	Total	8387/23035	14648↓	4650/2920	1730↑	1903/460	1443↑

6.3.1.2 Network reachability

To measure network reachability, we prepared and ran all the successfully extracted firmware images as outlined in Section 6.2.4 and sent them ICMP echo request packets. As shown in Table 6.1, we achieve significantly better results than Firmadyne. From the 4 650 successfully extracted firmware images, 1 903 images (40.9%) respond to the ICMP packets, compared to 460 images for Firmadyne (15.8% of the extracted images). For OpenWrt we were able to ping 674 devices whereas for Firmadyne no firmware image was reachable and this clearly increases our score. However, we find similar results for Zyxel (69 to 20), TP-Link (147 to 95) and Netgear (384 to 187).

We believe honware performs better for two reasons. First, Firmadyne supports only one ethernet device for their ARM little-endian platform whereas honware supports up to four. This is particularly important as a network bridge has to have at least one device attached (e.g. eth0) so that services are network reachable. Second, Firmadyne has no (fallback) mechanisms to correct missing network configuration settings, for example, because nvram could not be loaded in the absence of physical hardware. In contrast, as outlined in Section 6.2.2.5, honware will automatically detect an initial failure and will set up a bridge and an associated ethernet device.

6.3.1.3 Services

The execution of firmware applications is critical for honeypots since most exploits target these applications. If they do not function, then connecting firmware images to the Internet is of limited value. As shown in Table 6.2, significantly more applications running under honware respond on their listening ports than it is the case for Firmadyne. We find that for Telnet, 879 firmware images respond to our nmap scan compared to 149 for Firmadyne. HTTP on port 80 is the second most observed service with 676 firmware images responding to our nmap scan, followed by port 67 (316) and port 1900 (239).

We attribute our significantly better results to our instrumented kernel which has placed great emphasis on improved signal handling and on constructing a working network configuration, by executing our custom `/sbin/boot.sh` script. This technique allows us to very simply change default configurations, which is particularly important for firmware images that try to obtain IP addresses with DHCP.

6.3.2 Honeypot deployments in the wild

To evaluate the effectiveness of honware, we deployed multiple honeypots on the Internet including four brands of ADSL modems, TP-Link, D-Link, Eminent and ipTIME. We now discuss four case studies which show that devices can be rapidly emulated, very much faster than with previous approaches, and that honware can detect both known and previously unknown attacks. In particular, whilst emulating a router from ipTIME, we observed an unknown attack in which the default DNS setting in the router is changed to a rogue IP address – which we subsequently found to affect not only ipTIME, but also other brands.

6.3.2.1 Broadcom UPnPHunter

UPnPHunter is the name of a format string vulnerability in the Broadcom software for Universal Plug and Play (UPnP) and affects various brands, including TP-Link, D-Link and Netgear [71]. The vulnerability allows a remote adversary to cause the UPnP service

Table 6.2: Comparing honware and Firmadyne: Top 15 listening services.

Prot.	Port/Service	Honware	Firmadyne	Δ
TCP	23/telnet	879	149	730↑
TCP	80/http	676	293	383↑
UDP	67/dhcp	316	160	156↑
UDP	1900/UPnP	239	128	111↑
UDP	53/various	239	174	65↑
TCP	3333/dec-notes	222	102	120↑
TCP	5555/freeciv	203	57	146↑
TCP	5431/UPnP	177	48	129↑
UDP	137/netbios	154	82	72↑
TCP	53/domain	139	73	66↑
TCP	443/https	107	105	2↑
UDP	5353/mdns	102	34	68↑
UDP	69/tftp	104	26	78↑
TCP	1900/UPnP	56	60	4↓
TCP	49152/UPnP	53	62	9↓

to crash or execute arbitrary code. The attack seen in the wild was unusual in that an initial connection pre-qualified the devices as likely to be vulnerable before a second phase of the attack was attempted. 360Netlab reported that it took them over a month to code a custom honeypot to appropriately respond to each of the connections in turn [151]. With honware, because all services were operational, we were able to observe the described attack within 24 hours of connecting the honeypot to the Internet.

To capture the attack, we emulated an ADSL router modem (D-Link DSL-2741B with firmware version 517b50, released on 2010-03-08). This router uses the MIPS architecture (big endian) and by default has listening applications on ports 21/tcp, 22/tcp, 23/tcp, 80/tcp, 1028/tcp, 67/udp, 69/udp, and 5431/tcp. Before we connected the emulation to the Internet, we used the proof of concept code [71] to test the exploit. The exploit uses the functions `SetConnectionType` and `GetConnectionTypeInfo` on port 5431, the UPnP SOAP service. The first function is used to set the format string and the latter one to read the output. As expected, we managed to cause the UPnP daemon to crash, read arbitrary memory and execute arbitrary code with root privileges, giving us full control of the device.

On 2019-23-01 a machine from India connected to our honeypot on port 5431 and sent `<NewConnectionType>.%08X.%08X.</NewConnectionType>` so as to exploit the vulnerability. We sent `<NewConnectionType> .7F8805AC.004332F0.</NewConnectionType>` to reveal the memory mapping. Subsequently a malware loader connected to our honeypot from the same IP address as observed by 360Netlab [151].

Unfortunately, the loader failed to download malware and instead sent a single character `X`. We are not sure why this happened: it may be that the attacker forgot to update a placeholder and include some malware or shell code, or that the attacker does not have shell code for the particular type of router that we used in our experiment.

Although we did not capture any malware, we have shown that we can, within a single day, create a honeypot for a particular device and observe an attack upon it. There is clear value in rapidly understanding complex attack vectors and shortening the time window in which attackers can abuse vulnerabilities without anyone being able to precisely identify their methods.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" s:encodingStyle="
    http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
<u:AddPortMapping xmlns:u="urn:schemas-upnp-org:service:WANIPConnection:1">
<NewRemoteHost></NewRemoteHost>
<NewExternalPort>47359</NewExternalPort>
<NewProtocol>TCP</NewProtocol>
<NewInternalPort>135</NewInternalPort>
<NewInternalClient>192.168.8.1</NewInternalClient>
<NewEnabled>1</NewEnabled>
<NewPortMappingDescription>galleta silenciosa</NewPortMappingDescription>
<NewLeaseDuration>0</NewLeaseDuration>
</u:AddPortMapping>
</s:Body>
</s:Envelope>
```

Figure 6.2: EternalSilence: Malicious port forwarding rule captured by honware

6.3.2.2 DNS hijack

We observed a previously unknown attack in which the default DNS server was changed within one of our honeypots that emulates an ipTIME N604R wireless router with firmware version 7.50 (released on 2011-01-31). This particular device is manufactured by EFM networks and is primarily used in Korea where it is distributed by ipTIME.

By default, the router has listening applications on port 80/tcp, 113/tcp, and 68/udp. In 2015, Kim [72] discovered a remote code exploitation vulnerability triggered by sending a crafted DHCP request. We were running two instances of this firmware on two different IP addresses located in Finland and Germany and were hoping to see this attack – instead of which we recorded a previously unknown attack.

The attacker used the timepro.cgi script and the WAN setup menu to overwrite the default DNS to a rogue IP address located in the Netherlands. This caused the device to change the iptables rule as follows: `/sbin/iptables -t nat -A PREROUTING -i br0 -d 192.168.0.1 -p udp --dport 53 -j DNAT --to-destination X.X.X.X` with x.x.x.x being a DNS server controlled by the attacker. They also configured a second DNS server to ensure that all DNS traffic went to their machines.

We experimentally resolved www.yahoo.com on the attacker's DNS server and found that it resolved to a machine in China on AS41718 (China Great Fire Wall Network Limited Company). The resolved IP address has an unusual (self-signed) certificate which makes it authoritative for a range of websites, many with a Korean connection, but also www.paypal.com, www.yahoo.com, www.google.com.tw and many more. According to Shodan, there are 39 other IP addresses with the same certificate spread across the world, including Hong Kong, Taiwan and United States/California.

A search of online support forums showed that the same DNS servers were associated with other attacks. Three users with TP-Link[2] and Anderson[3] devices had noticed that their DNS settings had been changed. We reported our findings to a vetted community of security professionals and law enforcement so they can take appropriate action.

[2]https://community.tp-link.com/en/home/forum/topic/158073?page=1&t=2019 and https://trzepak.pl/viewtopic.php?f=20&t=61263

[3]https://eforum.idg.se/topic/358185-firefox-660-64-bit-quantum-säkerhetsvarnar-för-youtube-och-s¨kmotorn-duckduckgo/

6.3.2.3 UPnPProxy

EternalSilence is a newly discovered family of UPnPProxy forwarding malware [121]. EternalSilence adds port forwarding rules to devices to expose TCP ports 139 and 445 behind routers. Every rule is added with an identical description of "galleta silenciosa" and can therefore be easily fingerprinted. The rules are persistently stored in the router's configuration so reboots will not clear them; victims have to explicitly delete them.

Using our framework, we set up an ADSL modem router (Eminent EM4544 with firmware version 8.38, released 2013-05-30). This type of router uses the MIPS architecture (big endian) and by default has listening applications on port 280/tcp, 113/tcp, 68/udp and 5431/tcp. We set up the honeypot on 2019-01-04 and four days later on 2019-01-08 a machine from Bolivia issued a M-SEARCH request (`M-SEARCH * HTTP/1.1`) followed by a GET request (`GET /etc/linuxigd/gatedesc.xml HTTP/1.0`) to retrieve the device details. The response includes various device details such as deviceType, manufacturer and modelnumber. Subsequently the attacker issues an AddPortMapping request as shown in Figure 6.2. This rule triggers the miniupnp daemon to redirect port 47359 to 192.168.8.1:135, an action which is appropriately logged as follows: `miniupnpd[202]: redirecting port 47359 to 192.168.8.1:135 protocol TCP for: galleta silenciosa`.

We successfully captured the alteration of the port forwarding rules, but along with Akamai [121] who originally identified this attack, we saw no further attempt to exploit the compromised devices. Nonetheless, honware is providing a mechanism to easily capture any such attack traffic by pretending to be a vulnerable device. If the malicious firewall rules are exploited in future, we will be able to observe this behaviour instantly and report it appropriately.

6.3.2.4 Mirai variants

The Mirai source code is constantly evolving and recently a new variant called Yowai/Hakai was found. This variant exploits a vulnerability in the *invokeFunction* of ThinkPhP [137] and allows the execution of arbitrary code on the underlying server. ThinkPhP is a PHP framework widely used by a variety of networked devices, particularly those manufactured in China [135]. Once devices are infected, they do further scanning to find other vulnerable devices. One advantage of ThinkPhP is that it does not compete with the original Mirai malware as it targets the web server on port 80, not Telnet on port 23. Off-the-shelf Mirai honeypots would not record such attacks as they look for Mirai traffic on the Telnet ports 23 and 2323 [104].

To capture attack traffic, we processed the firmware for the ADSL modem router TP-Link TD-W8960N, released on 2011-11-08, with honware and set up a honeypot. By default, this devices has listening ports on 21/tcp, 22/tcp, 23/tcp, 80/tcp, 67/udp, 69/udp, 1900/udp and 5431/tcp. Our honeypot ran from 2019-02-17 to 2019-03-01 (14 days) and captured 566 attacks that tried to exploit the vulnerability in the ThinkPhP framework. In total, 49 different URLs, i.e. malware instances, were captured with a median of 4 attacks for each unique URL. One malware sample was associated with 70 of the attacks. We checked with Virustotal and 16 (32.7%) of the 49 samples were entirely new. Of the rest, over two thirds were captured by honware before they were first recorded by Virustotal; and only 13 (26.5%) samples had been detected by someone else and uploaded to Virustotal before honware. Across all 49 samples honware detected the malware a median of 9.7 days

Table 6.3: ThinkPhP vulnerability: Top 15 Malware files observed within a 14-day period emulating a TP-Link TD-W8960N

# Seen	Filename	Country	First seen		Detection ratio
			Honware	Virustotal	Virustotal
52	Tsunami.x86	DE	2019-23-02	unknown	5/67
35	cayo4	DE	2019-28-02	2019-21-03	10/68
34	Tsunami.x86	RO	2019-19-02	unknown	5/67
8	X86_64	CA	2019-28-02	unknown	0/66
6	shiina	US	2019-28-02	unknown	7/67
5	Tsunami.x86	US	2019-27-02	unknown	0/66
5	Tsunami.x86	US	2019-24-02	unknown	2/67
5	lessie.x86	NL	2019-26-03	2019-23-02	2/66
4	Tsunami.x86	ZA	2019-26-03	2019-01-03	13/71
4	Tsunami.x86	US	2019-18-02	unknown	4/67
3	Tsunami.x86	DE	2019-23-02	unknown	0/66
3	Tsunami.x86	US	2019-21-02	unknown	2/66
2	cayo4	NL	2019-22-02	unknown	0/66
2	x86	US	2019-19-02	unknown	0/66
2	Tsunami.x86	US	2019-27-02	unknown	1/66

before Virustotal had a copy. It appears that we are able to make malware available to the defender community considerably faster than traditional honeypots.

6.3.3 Timing attacks to fingerprint honware

Honware is, by design, difficult to fingerprint at the network stack or application layer, but timing attacks are a potential concern – the emulation may significantly affect the speed of operation, so we evaluated this issue extremely carefully.

Emulation inevitably introduces overhead in terms of CPU usage, network latency and I/O operations. However, many CPE and IoT devices have limited resources; for example, the D-Link home router DIR 825 has a CPU clocked at 680 MHz and just 64MB of RAM. Furthermore, it is typically used in residential networks with limited (upload) bandwidth. In contrast, even the lowest tier virtual machines (VMs) offered by popular cloud providers where honeypots might be deployed have a *virtual* CPU core clocked at several GHz, offer 1GB of RAM and have a 1Gbit connection. Hence, any timing issue is as likely to result from running too fast as from running too slow.

To compare our honeypots to real devices, we sought out self-identifying devices with listening services on port 21, 23 and 443. We specifically chose 443 as encrypting traffic needs more resources and thus may serve as a good distinguisher between our honeypot and the real servers.

Unfortunately, most devices do not reveal their model and firmware version through their listening services without further interaction, but using Shodan we were able to find three suitable devices: The ASUS RT-AC52U Dual-Band AC750 wireless router (FTP server on port 21), the Zyxel VMG1312-B Wireless N VDSL2 Gateway (Telnet server on port 23) and the D-Link Wireless N Dual-Band Router DIR-825 (web server on port 443). To identify the servers, we assume that every device that returns the string `VMG1312-B10A Login:` when connected to on port 23 is the particular model in question, and Shodan reports receiving that string from 120 devices. Likewise, we expect the string `220 Welcome to ASUS RT-AC52U FTP service` to be emitted only by ASUS RT-AC52U devices (74

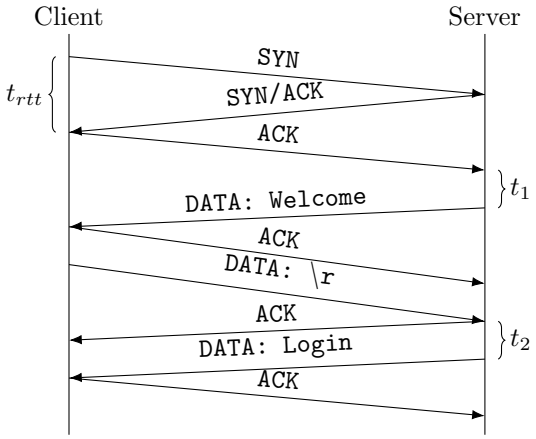


Figure 6.3: FTP Server: We initiate a connection and measure t_1 and t_2, the time the FTP daemon takes to responds to the ACK t_{ack} and the carriage return t_{cr}. Upon receiving the ACK, the remote server will send the welcome message `220 Welcome to ASUS RT-AC52U FTP service` $t_{welcome}$ and the carriage return will cause the FTP server to present the login prompt t_{login}. In both cases, the RTT (t_{rtt}) is used to adjust the timing information on the received and transmitted messages so that $t_1 = t_{welcome} - t_{rtt}$ and $t_2 = t_{cr_{ack}} - t_{rtt}/2$ or $t_2 = t_{login} - t_{cr} - t_{rtt}$

devices) and the string `HTTP/1.1 200 Ok Server: DIR-825 web server/v1.00` only to be sent by D-Link DIR-825 models (127 devices).

Having identified three suitable devices, we set up 30 honeypots to emulate them, ten for each, on two popular cloud providers with instances around the world including Singapore, Canada, USA, Germany, India, Netherlands and the United Kingdom. Then for each protocol (FTP, Telnet, HTTPS), we measure the time the application servers take to respond to our requests both for the honeypots and for the real devices identified via Shodan.

For each measurement, the initial round trip time (RTT), the time between the SYN and SYN-ACK packet, is calculated and is subsequently used to adjust the timing information on received and transmitted messages. As an example, figure 6.3 shows the interaction with the devices' FTP servers and the adjustments we made. Hence our measurements do not aim to identify network delays or Internet-induced latency, but solely to measure the time the application servers need to generate the appropriate response to an incoming probe.

When we connect to the FTP server on port 21, the remote server will send, upon completing the TCP handshake, the welcome message `220 Welcome to ASUS RT-AC52U FTP service` at time $t_{welcome}$. Similarly, after sending a further carriage return (t_{cr}), the FTP server will respond with `530 Please login with USER and PASS` (t_{login}). The time it takes the application server to respond is therefore $t_1 = t_{welcome} - t_{rtt}$ and $t_2 = t_{cr_{ack}} - t_{rtt}/2$ or $t_2 = t_{login} - t_{cr} - t_{rtt}$. The Telnet and HTTP protocol are essentially similar, but instead of sending a carriage return, we start and complete the TLS handshake and subsequently ask the web server to send the main web page with a standard GET request. For Telnet servers, we do not negotiate with the remote server or send any data other than the SYN and ACK packets as by default, Telnet servers will start the negotiation process and present a login prompt without further interaction.

As shown in Figure 6.4, the application servers' response times do vary between the

real and emulated devices. Each line in Figure 6.4 represents the empirical cumulative distribution function of one device in various locations as described above. For each of these devices, we made 300 measurements over a one-day period to measure the response time to our connections and resource requests. We find real systems that are faster than our emulated devices, and systems that are slower – but with significant differences between protocols.

For FTP, the welcome message is consistently sent faster on real devices than on emulated ones. The latter need about 5ms to populate the welcome messages while the real systems took about 1.8ms. Interestingly, the message in response to our carriage return is not significantly slower on the emulated devices. We cannot be sure why this is the case, but we speculate that the ASUS FTP server performs additional operations for each initial FTP connection such as filesystem checks or initialising memory. These operations are likely to be particularly fast on real devices as they are using flash memory whereas the VMs use SSDs or even slower HDDs.

For Telnet, the emulated devices respond very much faster than real ones. We further find that the response time for the latter is fairly variable with most Telnet servers responding in about 60-75ms (adjusted for RTTs) whereas emulated devices consistently took a few milliseconds. Similarly, the real devices need longer to present the login prompt.

For HTTPS we find the real devices and emulated devices respond in about the same time with most servers completing the TLS handshake in about 30ms. The time between the start of the TLS handshake and the end of the application data transfer (web page) is also comparable.

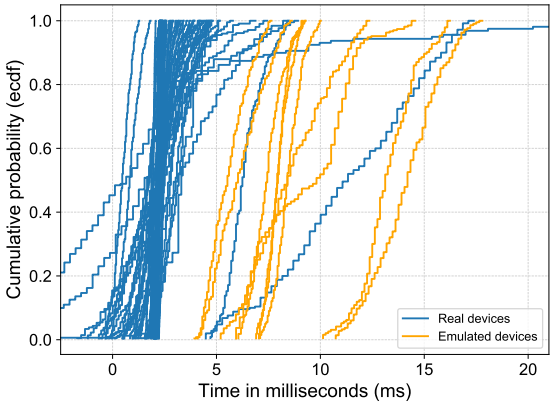
Overall, we find that emulation does not generally slow down application servers – which we attribute to even the low-cost cloud instances we used having a far better specification than most CPE and IoT devices. Where emulation is faster, it would be possible to artificially slow honware responses.

It is of course true (and entirely expected) that in some instances it would be possible to fingerprint honware. However, the differences can be made small enough that it would take repeated measurements and a reference group (i.e. access to real devices) before an attacker could reliably distinguish an emulation from a real device. Furthermore, the Internet inherently introduces jitter, network delays and artefacts which all serve to further increase the time and effort to mount such attacks – and in the case of HTTPS, where honware is running at almost the same speed as the real devices – fingerprinting is going to be extremely problematic.

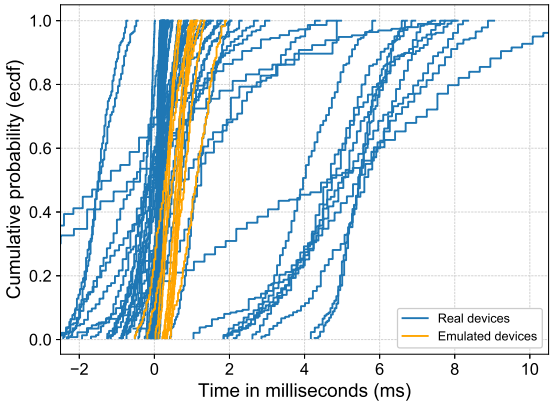
6.4 Ethical considerations

We followed our institution's ethics policy at all times with appropriate authorisation at every stage. We reported the DNS hijack attack (Section 6.3.2.2) to a vetted community of security professionals and law enforcement so they can take appropriate action.

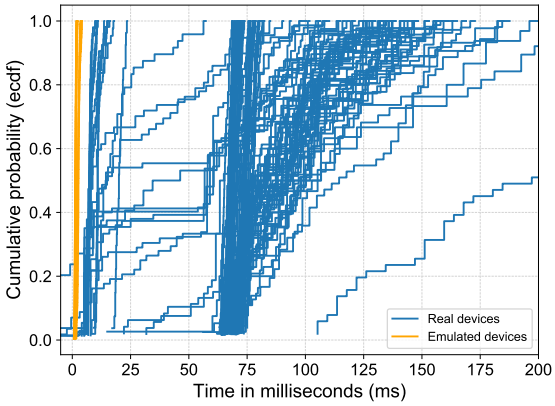
Extracting firmware images to analyse their security properties locally, is long established practice [161, 124, 33, 27]. It may be argued that honware is different because we connect the firmware together with our custom kernel to the Internet where it is not then analysed by us, but by unknown malicious entities. Our view is that our research is in the public interest since being able to create honeypots rapidly for a variety of Internet-connected devices enables not only security researchers, but also manufacturers, to detect novel attack vectors and provide updates to patch vulnerabilities. Furthermore,



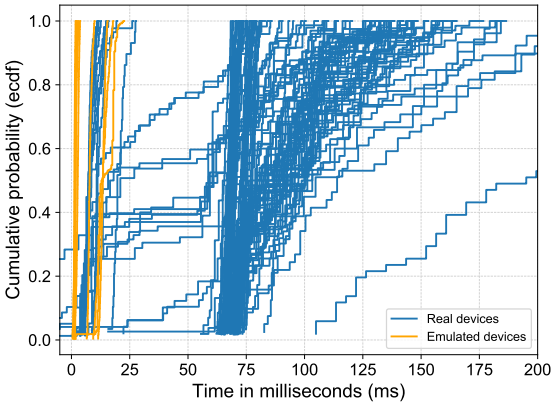
(a) ASUS RT-AC52U FTP server: Time to welcome message



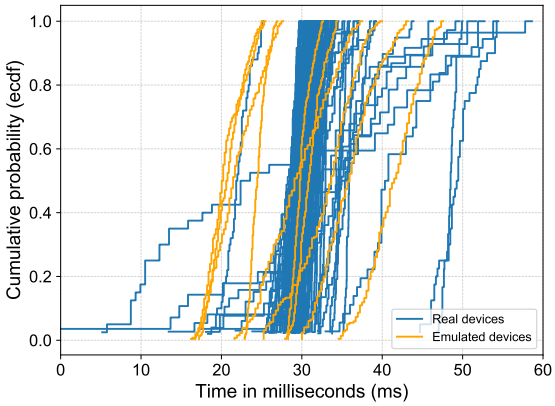
(b) ASUS RT-AC52U FTP server: Time between resource request (carriage return) and login message



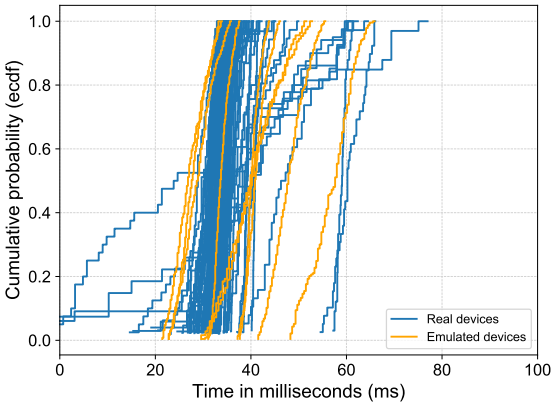
(c) Zyxel VMG1312-B10A Telnet server: Time to Telnet negotiation characters



(d) Zyxel VMG1312-B10A Telnet server: Time to Login message



(e) D-Link DIR-825 HTTPS server: Time to complete the TLS handshake



(f) D-Link DIR-825 HTTPS server: Time between ClientHello and resource received (web page)

Figure 6.4: We used three devices and protocols to measure the overhead of emulating honeypots with honware. Each line represents the empirical cumulative distribution function (ecdf) for one device. For each device, 300 measurements were made over a one-day period to measure the time the application servers need to respond to our requests. To do so, the timing information is adjusted to factor in Round Trip Time(s) (RTT).

we only use firmware images that are publicly accessible on the companies' websites and do not require registration or license keys.

We avoided doing our own Internet-wide scans to identify suitable devices for our timing measurements, but used Shodan instead. Our timing probe's contribution to the remote servers' overall traffic is negligible as we only initiated 300 connections for each device for a one-day period. Devices that have open ports on 21, 23 and 443 will typically receive orders of magnitude more traffic, in particular from malware (e.g., Mirai on port 23) or from search engines that index the web continuously.

When running high-interaction honeypots there is always the potential for damage to third parties, i.e. someone might use our emulated devices to conduct further attacks or perform malicious activities. All our honeypots ran in virtual machines and we closely monitored our honeypots and stopped abuse once it became substantial in volume or of no further interest. We blocked all ports on the host machine which would typically not be used by the device to reduce the attack surface. We further closely monitored our honeypots including traffic sent and received, disk and cpu usage, and kernel log messages. For example, using our devices as a proxy to send spam emails is expected, but after the modus operandi, i.e. the attack vector, the attack itself and the consequences are well understood, we blocked all outgoing traffic and reset the device. For one device we had to block outgoing traffic as it was roped into a DDoS attack using the SSDP protocol.

Our approach is completely in line with traditional good practices when running high-interaction honeypots. It should be noted that taking actions to block outgoing traffic is a trade-off between understanding the attack scenario better and hiding from attackers who want to identify honeypots. An extended discussion about these trade-offs can be found in Nawrocki et al. [100].

6.5 Discussion

Honware is intended to identify attacks that cannot easily be captured with the traditional approach of low-/medium-/high- interaction honeypots. It is not designed as a tool for understanding large-scale, repetitive attacks, i.e. if a device is capable of being compromised by Mirai we are only interested in the attack once, not in seeing that the same attack occurs again and again every few minutes. After the attack vector is known, we need to prevent further compromises, for example by blocking certain IPs or by recognising attack traffic (for example, Mirai's initial scanning sets the Initial Sequence Number to match the destination IPv4 address). Once an attack is well-understood, medium-interaction honeypots should be set up to collect more quantitative data about large-scale attacks, reducing maintenance and minimising potential harm.

Honware's real value is in its potential to rebalance the economics of attackers and defenders. It has become feasible to scan all of IPv4 address space for vulnerable devices with modest investment. Once an exploit is found for one technology, device, or specific implementation, attackers can easily find devices with that vulnerability embedded – and instantly benefit from that exploit. Using honware to identify the exact attack vector and obtain copies of malware means that countermeasures can be deployed faster and with far more precision.

We accept that honware can be fingerprinted by attackers who are prepared to perform a significant amount of measurement work to identify small timing discrepancies or have local access. In particular with local access, adversaries may be able to fingerprint the

honeypot based on differences in our pre-built kernel, the modified filesystem or application behaviour as a result of the changes made to the signal handling. However, honware is not susceptible to remote fingerprinting attacks based on protocol deviations [142] or self-revealing properties [96]. Our approach of exposing the *real* services to the Internet and our use of the standard configuration files that are shipped by the manufacturers means that our honeypots will be indistinguishable from real devices. Traditionally honeypots were engaged in 'Red Queen's Race' in which new fingerprinting attacks were countered by updating the emulation code. Honware avoids this entirely, which is particularly important as it has been show in Chapter 5 that honeypot operators rarely update their honeypots or pay attention to how they are configured.

We envision honware being used at Internet scale, for example, by manufacturers setting up honeypots for every one of their products and firmware versions. They will learn whether known vulnerabilities are being actually exploited and they will learn of previously unknown issues in an extremely timely manner. Currently the number of potentially vulnerable devices listed on search engines such as Shodan is often used to classify vulnerabilities as low-, medium- or high-impact. The framework will allow a much better assessment of the risk of having (unpatched) devices connected to the Internet and allows for a more thorough approach in determining the impact of vulnerabilities.

At present the honware framework focuses on CPE and IoT devices, but aims to support a wide variety of these devices. Thus we made certain design decisions which may not be optimal for a specific brand or device type. However, adjustments can be made at any point, in particular a cooperating device manufacturer could assist in specifying missing nvram values or by suggesting other configuration tweaks.

Currently honware is limited to Linux-based devices for ARM and MIPS architectures. As other architectures become more prevalent, it is straightforward to compile the Linux kernel with minimum effort. However, the emulation is still limited by the capabilities of QEMU and its support for architectures and considerably more work would be needed to support devices which are not based on Linux, but use proprietary operating systems.

6.6 Related work

IoTPOT was one of the first generic high-interaction honeypots tailored to impersonate IoT devices [92]. It supports eight architectures including ARM, MIPS and x86 and aims to return appropriate strings to connections on port 23. When the command is unknown, it tries to run the command in a sandboxed environment based on OpenWRT and infers the appropriate return string(s). Similarly, *Conpot* emulates industrial control systems based on the protocols Modbox and SNMP. It supports the emulation of large infrastructures so that adversaries may believe they are interacting with a complex industrial system network.

As IoTPOT and Conpot [114] do not use actual firmware to emulate devices and therefore return static information, Litchfield et al. developed HoneyPhy [84]. This framework tries to provide an appropriate simulation by taking a data feed from attached physical devices. For example, if an attacker turns on the heating via a compromised web-interface, the honeypot has to genuinely reflect these changes so that adversaries do not become aware that they are interacting with a honeypot. However, IoTPOT, Conpot and HoneyPhy only emulate specific application/ network layers and are not based on actual firmware. Thus their behaviour is bound to differ from actual IoT devices.

The approach of IoTCandyJar [86] is more sophisticated. IoTCandyJar utilises publicly available IoT devices on the Internet to collect responses for HTTP and then uses a Markov decision process (MDP) to respond to attackers' probes. They show that, after a learning period scanning the Internet, they are able to send meaningful responses and capture attack traffic. However, their technique only works for non-encrypted traffic and can only capture responses before any login, i.e. router admin interfaces and similar cannot be represented. They also rely on finding a significant number of publicly available devices, which must also identify themselves, to provide meaningful responses.

In 2017 Guarnizo et al. presented a "scalable high-interaction" honeypot platform based on physical devices [60]. They exposed six physical security cameras, one networked video recorder and one networked printer through a distributed architecture on a range of IPv4 addresses. In the two months of their study they found that attacks on IoT devices are geographically spread ranging from 600 000 incoming TCP connections for popular regions to only 50 000 in less popular areas.

In 2016, vendors and ISPs were caught off-guard by the *TR-069 NewNTPServer* exploit which can be used to execute arbitrary commands on vulnerable routers [140]. TR-069 is an application layer protocol for remote management of end-user devices and custom honeypots that monitor TR-069 protocol are now available [48]. Similarly, new designs for programmable logic controller honeypots focusing on industrial control systems have been presented [82].

Recent advances have also been made by scanning the Internet IPv4 address space for vulnerable industrial control systems and identifying honeypots. Feng et al. use a heuristic algorithm to determine the probability that the detected ICS device is a honeypot [51]. More recently, it has been shown that industrial control systems are increasingly deployed around the world and that 60,000 thousand of these systems are publicly accessible [93].

Demonstrating the risk of IoT devices, Ronen et al. showed that Philips Hue smart lamps can be used to spread malware [117]. In their example, the malware spreads from one lamp to its neighbours and infects lamps located in the near vicinity. They estimate that for a city the size of Paris, only 15 000 randomly located light bulbs are sufficient to get every light bulb infected.

Closest to the present work Chen et al. presented Firmadyne which dynamically analyses Linux-based firmware images to find vulnerabilities [26]. They rely on a precompiled Linux kernel and use *QEMU* [13] as a full system emulator. However, their approach is not scalable as it requires constant manual effort and experts to classify failures during the firmware extraction and emulation phases. It is intended to allow developers to find vulnerabilities rather than to be deployed on the Internet to be probed by attackers. Our honeypot framework is significantly better in configuring the networking aspects of firmware images, in making the devices network reachable, and most importantly, in running the listening applications such as web servers (see Section 6.3.1).

Gustafson et al. presented Pretender, a proof-of-concept system which records interactions between the original hardware and the firmware so that it can then create models of hardware, including devices, peripherals and CPU [61]. While the approach seems promising, their system has only be tested with three different CPUs on development hardware. Furthermore, their system requires access to physical hardware as they assume that the memory layout of the target device is known.

Another project to focus on firmware images, *Firmalice* is a binary analysis framework that aims to find authentication bypass vulnerabilities [124]. It supports inspecting the

codebase to find hardcoded credentials, hidden authentication interfaces and unintended bugs which allow adversaries to skip authentication and perform privileged operations.

6.7 Conclusion

Honware is the first system that allows system designers, developers and security researchers to efficiently and effectively deploy high verisimilitude, high interaction, honeypots for networked devices. Instead of having to buy and set up physical devices as honeypots, the framework facilitates the virtualisation of CPE and IoT devices merely by downloading standard firmware images from manufacturers' websites.

We demonstrate that honware can emulate a large variety of devices of many different brands within a virtual environment, independent of the underlying hardware. Our framework outperforms existing emulation strategies which are limited in their scalability, and honware is significantly better than previous projects in providing network functionality and in emulating the firmware applications – a crucial aspect as vulnerabilities are frequently exploited by attackers in 'front-end' functionality such as web interfaces or UPnP daemons.

An increasing number of exploits use multiple protocols in different phases of the attack and are targeted at very specific software implementations and devices. Generic honeypots are ineffective in capturing these attacks as they do not return the appropriate traffic to allow later parts of the attack to commence and so be recorded. Honware uses the original firmware applications and their configurations which means that every phase of an attack can be monitored and fully understood. Furthermore, using the original applications makes honware instances more fingerprint resistant and prevents fingerprinting attacks based on protocol deviations or those that identify configurations specific to honeypots. We further showed that the performance of honware is comparable to that of real devices and that it is not susceptible to trivial fingerprinting based on timing attacks.

Our honeypot framework has huge potential in detecting vulnerabilities in CPE and IoT devices that might otherwise be exploited for considerable periods of time without anyone noticing. We presented four real world case studies showing the practical value of our approach and in particular that within one day we were able to characterise a sophisticated attack which had taken experts a month to identify using traditional techniques. Additionally, while hoping to see an attack that had been reported to be occurring, we identified a previously unknown DNS changing attacker associated with a complex infrastructure.

Attackers are constantly scanning the Internet to find vulnerable devices. We believe honware is a major step forward in rebalancing the economics of attackers and defenders by cutting the attackers' ability to exploit vulnerabilities, particularly 'zero day' vulnerabilities, for considerable periods while defenders are unable to capture the details of the attack and thereby start the process of mitigation.

Chapter 7

Conclusion

The core aim of this work was creating effective honeypots to detect and profile the growing threat of autonomous and Internet-scale attacks against the Internet of Things, an endeavour previously limited by a fundamentally flawed generation of honeypots and associated misconceptions of the threat landscape.

In Chapter 3, we challenged previous research findings and showed that warning banners in an attacked computer system have no deterrent effects. In fact, we find that displaying Lorem Ipsum text has the same effect as warning banners and a standard welcome message. This is because the vast majority of system trespassing is performed autonomously and so honeypots capture the behaviour of bots rather than humans. This study not only informs honeypot developers, but also highlights that honeypot data used without careful assessment of the threat landscape results in incorrect conclusions and policy advice. Future research may want to survey the threat actors on underground forums to get a better understanding of their motives and psychological traits – we attempted to do this with honeypots but failed to convince the very few individuals interacting with our honeypots to take part in our survey.

We then investigated state-of-the-art low- and medium-interaction honeypots and presented a 'class break', showing that we can fingerprint a large number of low- and medium-interaction honeypots at Internet scale solely based on their protocol implementation (Chapter 4). Their use of off-the-shelf libraries to implement protocols meant that we were able to identify thousands of distinguishing probes. Because the RFCs that define the protocols do not mandate every protocol detail, there are numerous differences on-the-wire between entirely standards-compliant implementations. Using only a single probe sent to every host on the Internet, we found more than 7 600 honeypots across nine different honeypot implementations for the network protocols SSH, Telnet, and HTTP. In addition to demonstrating the power of our discrimination, the technique also shows that even if honeypot developers fix single protocol deviations, there are thousands more deviations that could be used instead, suggesting that efforts to mimic a protocol implementation will always fail. It is also worrying that our technique can be easily implemented by adversaries in malware to avoid interacting with and being detected by honeypots. Thus we need a new generation of honeypots in which more emphasis is put on the accuracy of the lower levels of the networking stack.

In Chapter 5, we characterised the honeypot landscape in that we use our technique to find honeypot deployments at Internet-scale. Our goal was to count the number of deployed honeypots, to better understand how and where honeypots are set up, and to find out if honeypot operators are actively looking after them. To do so, we repeatedly

conducted Internet-wide scans in a one-year period and authenticated to all detected honeypots. We found that a large number of honeypots are out of date and that many honeypot operators are relying on standardised deployment scripts or public configuration files, allowing their honeypots to be trivially fingerprinted.

We have been the first to knowingly authenticate to honeypots and this caused concern: the paper reporting our results was rejected from a major conference because the Program Committee thought our interactions with the honeypots were illegal and hence the research was unethical. To this end, we give a detailed account and an extended legal analysis why we did not infringe computer-misuse laws and why our research is ethical. Our access was not 'unauthorised' because we did not impersonate a legitimate user of the system; we knew that after sending a standard value, the system will present an impersonation of a shell prompt under the pretence that we have logged in. Thus the controller of the honeypot has intentionally made available a vulnerable system and invites access of the kind in question, which we knew at the time we accessed the systems. We hope that this will enable more research in this area and will help other researchers to publish related work.

In Chapter 6, we designed and implemented honware, a honeypot framework for rapid implementation and deployment of high-interaction honeypots for CPE and IoT devices. Honware automatically processes a standard firmware image and utilises a special pre-built Linux kernel to emulate the device's behaviour within a virtual environment. Our proposed system is significantly better than existing systems at extracting firmware images, and making sure applications within the image are network accessible. We outline four case studies which demonstrate that honware is effective in detecting both known and previously unknown attacks. We show that using the actual device firmware eliminates the implementation vulnerabilities we exploited in Chapters 4 and 5, and that attempts to fingerprint our system based on simple timing analysis cannot be done at scale. By simplifying the process of deploying realistic honeypots at Internet scale, honware supports the detection of malware types that often go unnoticed by users and manufactures. We hope that honware, which is available by request, will be used at Internet scale by manufacturers setting up honeypots for all of their products and firmware versions or by researchers looking for new types of malware.

Apart from the design challenges solved in this dissertation, future research may want to investigate the use of micro virtual machines to speed up the process of booting and isolating the firmware from the underlying host operating system. At the moment, virtualisation projects such as Firecracker need a Linux kernel with version 4.14 or newer, and Firecracker is not available for ARM and MIPS architectures. As newer devices steadily use newer Linux kernels and MIPS is less prevalent, the use of such lightweight virtualisation technologies may become increasingly important. We also can envision a honeypot framework in which a listening service such as a web server is run in one environment and another service (e.g. SSH) is run in another instance. This would allow us to quickly reset the device and minimises maintenance efforts. Of course, a consistent state across services must be maintained to avoid fingerprinting attacks.

The slow but steady up-take of IPv6 may also make it harder for attackers to find vulnerable devices at Internet scale because scanning the complete IPv6 address space seems infeasible. However, particular attention should be paid on how the available IPv6 space is utilised. If device default configurations or ISP deployment strategies prefer predictable IPv6 address assignments, for example by choosing the first or last IPv6 address of the assigned IPv6 subnet, then the search space remains relatively small and

IPv6 would not yield any value over IPv4. But even with perfect deployment strategies and sensible default configurations, passive IPv6 address leakages as side-effects of using services such as DNS and NTP remain of concern. Attackers could use them to learn about IPv6 addresses in use and potentially vulnerable devices.

It also remains a significant challenge to detect if attackers are actively using fingerprinting techniques. It may be that more sophisticated attacks such as ours remain unused until the traditional method of fingerprinting, i.e. functionality-based distinction such as examining commands, are no longer working because the defense community is gradually improving existing honeypot solutions. However, what if fingerprinting honeypots becomes the norm? Would it be possible to modify real systems to look like honeypots so that attackers would avoid them? What are the (unintended) side-effects, if any?

We may also have to look further and find a way to incentivise manufacturers to monitor and safeguard their products better. Do manufactures have a responsibility to detect attacks targeting their devices or is that an unreasonable expectation? If so, for how long? Would it be enough if they provide the necessary tools and firmware images to allow others to run honeypots for their products?

Honeypots are an important cornerstone of today's Internet security, but for too long have been pretending to be *a* thing instead of the *right* thing. We hope that this thesis will make a useful and practical contribution to combat the abuse of CPE and IoT devices with a new generation of honeypots, and will serve as a foundation towards better threat intelligence in a world of ubiquitous networked 'things'.

Bibliography

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick McDaniel, and Matthew Smith. SoK: Lessons learned from Android security research for appified software platforms. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P '16)*, pages 433–451, San Jose, CA, 2016. IEEE.

[2] Eric Alata, Vincent Nicomette, Mohamed Kaaniche, Marc Dacier, and Matthieu Herrb. Lessons learned from the deployment of a high-interaction honeypot. In *Proceedings of the 6th European Dependable Computing Conference (EDCC '06)*, pages 39–46, Coimbra, PT, 2006. IEEE.

[3] Martin R. Albrecht, Jean Paul Degabriele, Torben Brandt Hansen, and Kenneth G. Paterson. A surfeit of SSH cipher suites. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, pages 1480–1491, Vienna, AUT, 2016. ACM.

[4] Omar Alrawi, Chaz Lever, Manos Antonakakis, and Fabian Monrose. SoK: Security evaluation of home-based IoT deployments. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (S&P '19)*, pages 208–226, Los Alamitos, CA, 2019. IEEE.

[5] John B. Althouse, Jeff Atkinson, and Josh Atkins. JA3 – A method for profiling SSL/TLS clients, 2019. URL `https://github.com/salesforce/ja3`.

[6] Anomali Inc. Modern honey network, 2014. URL `https://github.com/threatstream/mhn`.

[7] 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. In *Proceedings of the 26th USENIX Security Symposium (USENIX '17)*, pages 1093–1110, Vancouver, BC, 2017. USENIX Association.

[8] Emmanuel Baccelli, Oliver Hahm, Mesut Gunes, Matthias Wahlisch, and Thomas C. Schmidt. RIOT OS: Towards an OS for the Internet of Things. In *Proceedings of the 32th IEEE International Conference on computer communications Workshops (INFOCOM '13)*, pages 79–80, Turin, ITA, 2013. IEEE.

[9] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of the 9th International Workshop on Recent Advances in Intrusion Detection (RAID '06)*, pages 165–184, Hamburg, DE, 2006. Springer.

[10] Daniel J. Barrett and Richard E. Silverman. *SSH – The Secure Shell: The Definitive Guide*. O'Reilly Associates, Sebastopol, CA, 2001.

[11] Timothy Barron and Nick Nikiforakis. Picky attackers: Quantifying the role of system properties on intruder behavior. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC '17)*, pages 387–398, Orlando, FL, 2017. ACM.

[12] M. Baushke. Key exchange (KEX) method updates and recommendations for Secure Shell (SSH), 2017. URL `https://tools.ietf.org/id/draft-ietf-curdle-ssh-kex-sha2-09.html`.

[13] Fabrice Bellard. QEMU – A fast and portable dynamic translator. In *Proceedings of the 11th USENIX Annual Technical Conference (USENIX ATC '05)*, pages 41–46, Anaheim, CA, 2005. USENIX Association.

[14] M. Belshe, R. Peon, and R. Thomson. RFC 7540 – Hypertext Transfer Protocol Version 2 (HTTP/2), 2015.

[15] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945 – Hypertext Transfer Protocol – HTTP/1.0, 1996.

[16] Elisa Bertino and Nayeem Islam. Botnets and Internet of Things security. *Computer*, 50(2):76–79, 2017.

[17] John Bethencourt, Jason Franklin, and Mary Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium (USENIX '05)*, pages 193–208, Boston, MA, 2005. USENIX Association.

[18] Binwalk.org. Binwalk, 2019. URL `https://github.com/ReFirmLabs/binwalk/`.

[19] Blackhat. Breaking honeypots for fun and profit, 2015. URL `https://www.blackhat.com/us-15/briefings.html#breaking-honeypots-for-fun-and-profit`.

[20] Adam M. Bossler. Need for debate on the implications of honeypot data for restrictive deterrence policies in cyberspace. *Criminology & Public Policy*, 16(3):681–688, 2017.

[21] Janet M. Box-Steffensmeier and Bradford S. Jones. *Event history modeling: A guide for social scientists*. Cambridge University Press, 2004.

[22] David Brumley, Juan Caballero, Zhenkai Liang, James Newsome, and Dawn Song. Towards automatic discovery of deviations in binary implementations with applications to error detection and fingerprint generation. In *Proceedings of the 16th USENIX Security Symposium (USENIX '07)*, pages 213–228, Bosten, MA, 2007. USENIX Association.

[23] Robert E. Calem. New York's Panix service is crippled by hacker attack, 1996. URL `https://archive.nytimes.com/www.nytimes.com/library/cyber/week/0914panix.html`.

[24] Canonical. Taking charge of the IoTs security vulnerabilities, 2016. URL `https://pages.ubuntu.com/rs/066-EOV-335/images/IoTSecurityWhitepaper-FinalReport.pdf`.

[25] Carnivore. Dionaea – Low interaction honeypot, 2014. URL `http://dionaea.carnivore.it`.

[26] Daming Dominic Chen, Manuel Egele, Maverick Woo, and David Brumley. Towards fully automated dynamic analysis for embedded firmware. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS '16)*, pages 21–37, San Diego, CA, 2016. Internet Society.

[27] Jiongyi Chen, Wenrui Diao, Qingchuan Zhao, Chaoshun Zuo, Zhiqiang Lin, XiaoFeng Wang, Wing Cheong Lau, Menghan Sun, Ronghai Yang, and Kehuan Zhang. IoTFuzzer: Discovering memory corruptions in IoT through app-based fuzzing. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS '18)*, pages 1–15, San Diego, CA, 2018. Internet Society.

[28] Bill Cheswick. An evening with Berferd in which a cracker is lured, endured, and studied. In *Proceedings of the 1992 Winter USENIX Conference*, pages 163–174, San Francisco, CA, 1992. USENIX Association.

[29] Fred Cohen. The use of deception techniques: Honeypots and decoys, 2004. URL `http://all.net/courses.all.net/Deception/deception/Deception_Techniques_.pdf`.

[30] Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex: Protocol specification extraction. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P '09)*, pages 110–125, Oakland, CA, 2009. IEEE.

[31] Andrei Costin and Jonas Zaddach. IoT malware: Comprehensive survey, analysis framework and case studies. *BlackHat USA*, 2018.

[32] Andrei Costin, Jonas Zaddach, Aurélien Francillon, and Davide Balzarotti. A large-scale analysis of the security of embedded firmwares. In *Proceedings of the 23rd USENIX Security Symposium (USENIX '14)*, pages 95–110, San Diego, CA, 2014. USENIX Association.

[33] Andrei Costin, Apostolis Zarras, and Aurlien Francillon. Automated dynamic firmware analysis at scale: A case study on embedded web interfaces. In *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIACCS '16)*, pages 437–448, Xi'an, China, 2016. ACM.

[34] Council of Europe. Convention on cybercrime. Treaty No. 185, 2001.

[35] Cowrie. Project Philosophy, 2015. URL `https://github.com/micheloosterhof/cowrie/pull/48`.

[36] Ang Cui and Salvatore J. Stolfo. A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan. In *Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10)*, pages 97–106, New York, NY, 2010. ACM.

[37] RN. Dahbul, C. Lim, and J. Purnama. Enhancing honeypot deception capability through network service fingerprinting. *Journal of Physics: Conference Series*, 801 (1), 2017.

[38] Fan Dang, Zhenhua Li, Yunhao Liu, Ennan Zhai, Qi Alfred Chen, Tianyin Xu, Yan Chen, and Jingyu Yang. Understanding fileless attacks on Linux-based IoT devices with HoneyCloud. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19)*, pages 482–493, Seoul, KP, 2019. ACM.

[39] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. DDoS-capable IoT malwares: Comparative analysis and Mirai investigation. *Security and Communication Networks*, 2018.

[40] Deutsche Telekom AG. T-Pot: A multi-honeypot platform, 2015. URL `http://dtag-dev-sec.github.io/mediator/feature/2015/03/17/concept.html`.

[41] Deutsche Telekom AG. T-Pot 16.03 – Enhanced multi-honeypot platform, 2016. URL `http://dtag-dev-sec.github.io/mediator/feature/2016/03/11/t-pot-16.03.html`.

[42] /dev/ttyS0. Emulating NVRAM in QEMU, 2012. URL `http://www.devttys0.com/2012/03/emulating-nvram-in-qemu/`.

[43] DNS-OARC. Don't probe list, 2019. URL `https://www.dns-oarc.net/oarc/services/dontprobe`.

[44] Maximillian Dornseif, Thorsten Holz, and C.N. Klein. NoSEBrEaK – Attacking honeynets. In *Proceedings of the 5th Annual IEEE SMC Information Assurance Workshop*, pages 123–129, West Point, NY, 2004. IEEE.

[45] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium (USENIX '13)*, pages 605–619, Washington, D.C, 2013. USENIX Association.

[46] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by Internet-Wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*, pages 542–553. ACM, 2015.

[47] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. Exploring how privacy and security factor into IoT device purchase behavior. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*, pages 534:1–534:12, Glasgow, UK, 2019. ACM.

[48] Ömer Erdem. HoneyThing – A honeypot for Internet of TR-069 things, 2016. URL `https://github.com/omererdem/honeything`.

[49] Shayan Eskandari, Andreas Leoutsarakos, Troy Mursch, and Jeremy Clark. A first look at browser-based cryptojacking. In *Proceedings of the 2018 IEEE European*

Symposium on Security and Privacy Workshops (Euro S&PW '18), pages 58–66, London, UK, 2018. IEE.

[50] Wenjun Fan, Zhihui Du, David Fernández, and Víctor A Villagrá. Enabling an anatomic view to investigate honeypot systems: A survey. *IEEE Systems Journal*, 12(4):3906–3919, 2017.

[51] Xuan Feng, Qiang Li, and Haining Wang. Characterizing industrial control system devices on the Internet. In *Proceedings of the 24th IEEE International Conference on Network Protocols (ICNP '16)*, pages 1–10, Singapore, SG, 2016. IEEE.

[52] R. Fielding and J. Reschke. RFC 7231 – Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, 2014.

[53] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1, 1999.

[54] UPnP Forum. UPnP device architecture 1.1, 2008. URL `http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf`.

[55] Tal Garfinkel, Keith Adams, Andrew Warfield, and Jason Franklin. Compatibility is not transparency: VMM detection myths and realities. In *Proceedings of the 11th Workshop on HotTopics in Operating Systems (HotOS '07)*, San Diego, CA, 2007. ACM.

[56] Oliver Gasser, Ralph Holz, and Georg Carle. A deeper understanding of SSH: Results from Internet-wide scans. In *Proceedings of the 14th Network Operations and Management Symposium (NOMS '14)*, pages 1–9, Krakow, PL, 2014. IEEE.

[57] J.P. Gibbs. *Crime, Punishment, and Deterrence*. Elsevier, 1975.

[58] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. RFC 2518 – HTTP Extensions for distributed authoring – WEBDAV, 1999.

[59] Thomas Grudziecki, Pawel Jacewicz, Lukasz Juszczyk, Piotr Kijewski, and Pawel Pawlinski. Proactive detection of network security incidents. Technical report, European Network and Security Information Agency (ENISA), 2011. URL `https://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-report`.

[60] Juan David Guarnizo, Amit Tambe, Suman Sankar Bhunia, Martin Ochoa, Nils Ole Tippenhauer, Asaf Shabtai, and Yuval Elovici. SIPHON: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security (CPSS '17)*, pages 57–68, Abu Dhabi, UAE, 2017. ACM.

[61] Eric Gustafson, Marius Muench, Chad Spensky, Nilo Redini, Aravind Machiry, Yanick Fratantonio, Davide Balzarotti, Aurélien Francillon, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. Toward the analysis of embedded firmware through automated re-hosting. In *Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID '19)*, pages 135–150, Chaoyang District, Beijing, 2019. USENIX Association.

[62] Muhammad A. Hakim, Hidayet Aksu, A. Selcuk Uluagac, and Kemal Akkaya. U-PoT: A honeypot framework for UPnP-based IoT devices. In *Proceedings of the 37th IEEE International Performance Computing and Communications Conference (IPCCC '18)*, pages 1–8, Orlando, FL, 2018. IEEE.

[63] Nikolai Hampton and Patryk Szewczyk. A survey and method for analysing SOHO router firmware currency. In *Proceedings of the 13th Australian Information Security Management Conference*, pages 11–27. SRI Security Research Institute, 2015.

[64] Xiao Han, Nizar Kheir, and Davide Balzarotti. Deception techniques in computer security: A research perspective. *ACM Computing Survey*, 51(4):80:1–80:36, 2018.

[65] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Proceedings of the 27th Network and Distributed System Security Symposium (NDSS '19)*, San Diego, CA, 2019. Internet Society.

[66] Thomas J. Holt. On the value of honeypots to produce policy recommendations. *Criminology & Public Policy*, 16(3):739–747, 2017.

[67] Thorsten Holz and Frederic Raynal. Detecting honeypots and other suspicious environments. In *Proceedings of the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop (SMC '05)*, pages 29–36, West Point, NY, 2005. IEEE.

[68] IETF. Telnet options, 2011. URL `https://www.ietf.org/assignments/telnet-options/`.

[69] Loc Jaquemet. SSLSNOOP, 2015. URL `https://github.com/trolldbois/sslsnoop`.

[70] Harriet Mary Jones. The restrictive deterrent effect of warning messages on the behavior of computer system trespassers, 2014. URL `https://hdl.handle.net/1903/15544`.

[71] Leon Juranic. From zero to ZeroDay journey: Router hacking (WRT54GL Linksys case), 2013. URL `https://defensecode.com/whitepapers/From_Zero_To_ZeroDay_Network_Devices_Exploitation.txt`.

[72] Pierre Kim. 127 ipTIME router models vulnerable to an unauthenticated RCE by sending a crafted DHCP request, 2015. URL `https://pierrekim.github.io/blog/2015-07-06-127-iptime-router-models-unauthenticated-RCE-with-DHCP.html`.

[73] Kippo. SSH honeypot, 2015. URL `https://github.com/desaster/kippo`.

[74] J. Klensin and M. Padlipsky. RFC 5198 – Unicode format for network interchange, 2008.

[75] Kojoney. A honeypot for the SSH service, 2015. URL `https://github.com/madirish/kojoney2`.

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

[77] Lukas Krämer, Johannes Krupp, Daisuke Makita, Tomomi Nishizoe, Takashi Koide, Katsunari Yoshioka, and Christian Rossow. Amppot: Monitoring and defending against amplification DDoS attacks. In *Proceedings of the 18th International Symposium on Recent Advances in Intrusion Detection (RAID '15)*, pages 615–636, Kyoto, JP, 2015. Springer.

[78] N. Krawetz. Anti-honeypot Technology. *IEEE Security & Privacy Magazine*, 2(1): 76–79, 2004.

[79] Christian Kreibich and Jon Crowcroft. Honeycomb: Creating intrusion detection signatures using honeypots. *ACM SIGCOMM Computer Communication Review*, 34(1):51–56, 2004.

[80] Deepak Kumar, Kelly Shen, Benton Case, Deepali Garg, Galina Alperovich, Dmitry Kuznetsov, Rajarshi Gupta, and Zakir Durumeric. All things considered: An analysis of IoT devices on home networks. In *Proceedings of 28th USENIX Security Symposium (USENIX '19)*, pages 1169–1185, Santa Clara, CA, 2019. USENIX Association.

[81] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic. Distributed denial of service attacks. In *Proceedings of the 2000 IEEE International Conference on Systems, Man & Cybernetics (SMC '00)*, volume 3, pages 2275–2280, Nashville, TN, 2000. IEEE.

[82] Stephan Lau, Johannes Klick, Stephan Arndt, and Volker Roth. POSTER: Towards highly interactive honeypots for industrial control systems. In *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, pages 1823–1825, Vienna, AUT, 2016. ACM.

[83] S. Lehtinen and C. Lonvick. RFC 4250 – The Secure Shell (SSH) Protocol Assigned Numbers, 2006.

[84] Samuel Litchfield, David Formby, Jonathan Rogers, Sakis Meliopoulos, and Raheem Beyah. Rethinking the honeypot for cyber-physical systems. *IEEE Internet Computing*, 20(5):9–17, 2016.

[85] Lei Liu, Robert A. Wolfe, and Xuelin Huang. Shared frailty models for recurrent events and a terminal event. *Biometrics*, 60(3):747–756, 2004.

[86] Tongbo Luo, Zhaoyan Xu, Xing Jin, Yanhui Jia, and Xin Ouyang. IoTCandyjar: Towards an intelligent-interaction honeypot for IoT devices. In *Blackhat USA*, pages 1–11, Las Vegas, NV, 2017. Blackhat.

[87] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning.* Insecure, USA, 2009.

[88] Abdun Naser Mahmood, C. Leckie, and P. Udaya. An efficient clustering scheme to exploit hierarchical data in network traffic analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):752–767, 2008.

[89] David Maimon and Eric R. Louderback. Cyber-dependent crimes: An interdisciplinary review. *Annual Review of Criminology*, 2(1):191–216, 2019.

[90] David Maimon, Mariel Alper, Bertrand Sobesto, and Michel Cukier. Restrictive deterrent effects of a warning banner in an attacked computer system. *Criminology*, 52(1):33–59, 2014.

[91] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood. Home networking with universal plug and play. *IEEE Communications Magazine*, 39(12):104–109, 2001.

[92] Yin Minn, Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. IoTPOT: Analysing the rise of IoT compromises. In *Proceedings of the 9th USENIX Workshop on Offensive Technologies (WOOT '15)*, pages 1–9, Washington, D.C, 2015. USENIX Association.

[93] Ariana Mirian, Zane Ma, David Adrian, Matthew Tischer, Thasphon Chuenchujit, Tim Yardley, Robin Berthier, Joshua Mason, Zakir Durumeric, J. Alex Halderman, and Michael Bailey. An Internet-wide view of ICS devices. In *Proceedings of the 14th IEEE Conference on Privacy, Security, and Trust (PST '16)*, pages 1–8, Belfast, UK, 2016. IEEE.

[94] Mohammed Mohssen and Rehman Habib-ur. *Honeypots and Routers: Collecting Internet Attacks*. CRC Press, Boca Raton, 2015.

[95] Philipp Morgner, Christoph Mai, Nicole Koschate-Fischer, Felix Freiling, and Zinaida Benenson. Security update labels: Establishing economic incentives for security patching of IoT consumer products. In *Proceedings of the 41th IEEE Symposium on Security and Privacy (S&P '20)*, San Francisco, CA, 2020. IEEE.

[96] Shun Morishita, Takuya Hoizumi, Wataru Ueno, Rui Tanabe, Carlos Hernandez Ganan, Michel van Eeten, Katsunari Yoshioka, and Tsutomu Matsumoto. Detect me if you... oh wait. An Internet-wide view of self-revealing honeypots. In *Proceedings of 16th IFIP/IEEE International Symposium on Integrated Network Management (IM '19)*, Washington, D.C, 2019. IEEE.

[97] Andrew Morris. Detecting Kippo SSH honeypots, bypassing patches, and all that jazz, 2014. URL `https://www.redpacketsecurity.com/problems-i-have-found-with-kippo-honeypot/`.

[98] S. Mukkamala, K. Yendrapalli, R. Basnet, M. K. Shankarapani, and A. H. Sung. Network based detection of virtual environments and low interaction honeypots. In *Proceedings of the Information Assurance and Security Workshop (IAW '07)*, pages 92–98, West Point, NY, 2007. IEEE.

[99] Asuka Nakajima, Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Maverick Woo. A pilot study on consumer IoT device vulnerability disclosure and patch release in Japan and the United States. In *Proceedings of the 14th ACM Asia Conference on Computer and Communications Security (ASIACCS '19)*, pages 485–492, Auckland, NZ, 2019. ACM.

[100] Marcin Nawrocki, Matthias Wählisch, Thomas C. Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis, 2016. URL https://arxiv.org/abs/1608.06249.

[101] Jose Nazario. PhoneyC: A virtual client honeypot. In *Proceedings of the 2nd USENIX conference on large-scale exploits and emergent threats: Botnets, spyware, worms, and more*, pages 1–8, Berkeley, CA, 2009. USENIX Association.

[102] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. Set-up and deployment of a high-interaction honeypot: Experiment and lessons learned. *Journal in Computer Virology and Hacking Techniques*, 7(2):143–157, 2011.

[103] Jeremiah Onaolapo, Enrico Mariconti, and Gianluca Stringhini. What happens after you are pwnd: Understanding the use of leaked webmail credentials in the wild. In *Proceedings of the 16th Internet Measurement Conference (IMC '16)*, pages 65–79, Santa Monica, CA, 2016. ACM.

[104] Michael Oosterhof. Cowrie, 2016. URL https://github.com/micheloosterhof/cowrie.

[105] OpenSSH. Implications of using clear padding instead of random padding, 2017. Private communication.

[106] OpenWrt. Linux distribution for embedded devices, 2017. URL https://openwrt.org/.

[107] J. Postel and J. Reynolds. RFC 854 – Telnet protocol specification, 1983.

[108] J. Postel and J. Reynolds. RFC 855 – Telnet option specification, 1983.

[109] The Honeynet Project. Know your enemy: Sebek – A kernel based data capture tool, 2003. URL http://old.honeynet.org/papers/sebek.pdf.

[110] Niels Provos and Thorsten Holz. *Virtual honeypots: From botnet tracking to intrusion detection*. Pearson Education, 2007.

[111] Florian Quinkert, Eduard Leonhardt, and Thorsten Holz. Dorkpot: A honeypot-based analysis of google dorks. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb '19)*, San Diego, CA, 2019.

[112] Daniel Ramsbrock, Robin Berthier, and Michel Cukier. Profiling attacker behavior following SSH compromises. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*, pages 119–124, Washington, D.C, 2007. IEEE.

[113] IBM Research. The inside story on botnets: How threat actors exploit networks of infected computers to commit cybercrime, 2016. URL https://www.ibm.com/downloads/cas/V3YJVYZX.

[114] Lukas Rist. Conpot, 2019. URL https://github.com/mushorg/conpot/.

[115] Lukas Rist. Glastopf – Web application honeypot, 2019. URL https://github.com/mushorg/glastopf.

[116] Lukas Rist. SNARE – Super next generation advanced reactive honeypot, 2019. URL https://github.com/mushorg/snare.

[117] Eyal Ronen, Adi Shamir, Achi-Or Weingarten, and Colin O'Flynn. IoT goes nuclear: Creating a ZigBee chain reaction. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P '17)*, pages 195–212, San Jose, CA, 2017. IEEE.

[118] Christian Rossow. Amplification hell: Revisiting network protocols for DDoS abuse. In *Proceedings of the 22nd Network and Distributed System Security Symposium (NDSS '14)*, San Diego, CA, 2014. Internet Society.

[119] SANS Technology Institute. Honey pots and Honey nets – Security through deception, 2001. URL https://www.sans.org/reading-room/whitepapers/attacking/honey-pots-honey-nets-security-deception-41.

[120] SANS Technology Institute. Kippo users beware: Another fingerprinting trick, 2014. URL https://isc.sans.edu/forums/diary/Kippo+Users+Beware+Another+fingerprinting+trick/18119/.

[121] Chad Seaman. UPNPROXY: EternalSilence, 2018. URL https://blogs.akamai.com/sitr/2018/11/upnproxy-eternalsilence.html.

[122] Christian Seifert, Ian Welch, and Peter Komisarczuk. HoneyC – The low-Interaction client honeypot. In *Proceedings of the 2007 NZCSRCS*, pages 1–9, 2007.

[123] Shodan. Shodan – Honeyscore, 2017. URL https://honeyscore.shodan.io/.

[124] Yan Shoshitaishvili, Ruoyu Wang, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Firmalice – Automatic detection of authentication bypass vulnerabilities in binary firmware. In *Proceedings of 22nd Network and Distributed System Security Symposium (NDSS '15)*, pages 1–15, San Diego, CA, 2015. Internet Society.

[125] Stylianos Sidiroglou, Angelos D. Keromytis, and Kostas G. Anagnostakis. Systems and methods for detecting and inhibiting attacks using honeypots, 2011.

[126] Lance Spitzner. The value of honeypots, part one: Definitions and values of honeypots, 2001. URL http://www.symantec.com/connect/articles/value-honeypots-part-one-definitions-and-values-honeypots.

[127] Lance Spitzner. *Honeypots: Tracking Hackers*. Pearson Education, Boston, Massachusetts, USA, 2002.

[128] Lance Spitzner. The Honeynet Project: Trapping the hackers. *IEEE Security & Privacy Magazine*, 1(2):15–23, March 2003.

[129] Klaus Steding-Jessen, Nandamudi L Vijaykumar, and Antonio Montes. Using low-interaction honeypots to study the abuse of open proxies to send spam. *INFOCOMP Journal of Computer Science*, 7(1):44–52, 2008.

[130] Kevin F. Steinmetz. Ruminations on warning banners, deterrence, and system intrusion research. *Criminology & Public Policy*, 16(3):727–737, 2017.

[131] Mark Stockman, Robert Heile, and Anthony Rein. An open-source honeynet system to study system banner message effects on hackers. In *Proceedings of the 4th Annual ACM Conference on Research in Information Technology (RIIT '15)*, pages 19–22, New York, NY, 2015. ACM.

[132] Cliff Stoll. *The cuckoo's egg: Tracking a spy through the maze of computer espionage*. Simon and Schuster, 2005.

[133] Emil Tan. Thug – A client honeypot, 2014. URL `https://www.div0.sg/single-post/thug-client-honeypot`.

[134] Alexander Testa, David Maimon, Bertrand Sobesto, and Michel Cukier. Illegal roaming and file manipulation on target computers. *Criminology & Public Policy*, 16(3):689–726, 2017.

[135] ThinkPHP. ThinkPHP5 Framework, 2019. URL `https://github.com/top-think/framework/`.

[136] Daniel R. Thomas, Richard Clayton, and Alastair R. Beresford. 1000 days of UDP amplification DDoS attacks. In *Proceedings of the 12th APWG Symposium on Electronic Crime Research (eCrime '17)*, pages 79–84, Phoenix, AZ, 2017. IEEE.

[137] TrendMicro. ThinkPHP vulnerability abused by botnets Hakai and Yowai, 2019. URL `https://blog.trendmicro.com/trendlabs-security-intelligence/thinkphp-vulnerability-abused-by-botnets-hakai-and-yowai/`.

[138] Shreya Udhani, Alexander Withers, and Masooda Bashir. Human vs bots: Detecting human attacks in a honeypot environment. In *Proceedings of the 7th International Symposium on Digital Forensics and Security (ISDFS '19)*, pages 1–6, Barcelos, PT, 2019. IEEE.

[139] Joni Uitto, Sampsa Rauti, Samuel Laurén, and Ville Leppänen. A survey on anti-honeypot and anti-introspection methods. In *Proceedings of the 5th World Conference on Information Systems and Technologies (WorldCIST '17)*, pages 125–134, Porto Santo Island, PT, 2017. Springer.

[140] Johannes B. Ullrich. TR-069 NewNTPServer exploits: What we know so far, 2016. URL `https://isc.sans.edu/forums/diary/TR069+NewNTPServer+Exploits+What+we+know+so+far/21763/`.

[141] Ovidiu Vermesan, Peter Friess, et al. *Internet of things-from research and innovation to market deployment*, volume 29. River publishers Aalborg, 2014.

[142] Alexander Vetterl. OpenSSH honeypot (sshd-honeypot), 2018. URL `https://github.com/amv42/sshd-honeypot`.

[143] Alexander Vetterl and Richard Clayton. Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at Internet scale. In *12th USENIX Workshop on Offensive Technologies (WOOT '18)*, Baltimore, MD, 2018. USENIX Association.

[144] Alexander Vetterl and Richard Clayton. Honware: A virtual honeypot framework for capturing CPE and IoT zero days. In *Proceedings of the 14th APWG Symposium on Electronic Crime Research (eCrime '19)*, Pittsburgh, PA, 2019. IEEE.

[145] Alexander Vetterl, Richard Clayton, and Ian Walden. Counting outdated honeypots: Legal and useful. In *Proceedings of the 4th International Workshop on Traffic Measurements for Cybersecurity (WTMC '19)*, pages 224–229, San Francisco, CA, 2019. IEEE.

[146] Eric Vittinghoff and Charles E. McCulloch. Relaxing the rule of ten events per variable in logistic and cox regression. *American Journal of Epidemiology*, 165(6): 710–718, 2007.

[147] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 294–311. Springer, 2003.

[148] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C Snoeren, Geoffrey M Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *ACM SIGOPS Operating Systems Review*, 39(5): 148–162, 2005.

[149] Ian Walden. *Computer crimes and digital investigations, 2nd edition.* Oxford University Press, 2017.

[150] Ian Walden and Anne Flanagan. Honeypots: A sticky legal landscape. *Rutgers Computer & Technology Law Journal*, 29(2):317–370, 2003.

[151] Hui Wang and Root Kiter. BCMPUPnP_Hunter: A 100k botnet turns home routers to email spammers, 2018. URL `https://blog.netlab.360.com/bcmpupnp_hunter-a-100k-botnet-turns-home-routers-to-email-spammers-en/`.

[152] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security*, 4(1):30–51, February 2010.

[153] Theodore Wilson, David Maimon, Bertrand Sobesto, and Michel Cukier. The effect of a surveillance banner in an attacked computer system: Additional evidence for the relevance of restrictive deterrence in cyberspace. *Journal of Research in Crime and Delinquency*, 52(6):829–855, 2015.

[154] Kui Xu, Patrick Butler, Sudip Saha, and Danfeng Daphne Yao. DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing*, 10(3):143–153, 2013.

[155] Zhaoyan Xu, Antonio Nappa, Robert Baykov, Guangliang Yang, Juan Caballero, and Guofei Gu. AutoProbe: Towards automatic active malicious server probing using dynamic binary analysis. In *Proceedings of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, pages 179–190. ACM, 2014.

[156] Xu Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks with FTCS and DCC (DSN '08)*, pages 177–186, Anchorage, AK, 2008. IEEE.

[157] Vinod Yegneswaran, Jonathon Giffin, Paul Barford, and Somesh Jha. An architecture for generating semantics-aware signatures. In *Proceedings of the 14th USENIX Security Symposium (USENIX '05)*, pages 97–112, Baltimore, MD, 2005. USENIX Association.

[158] T. Ylonen and C. Lonvick. RFC 4253 – The Secure Shell (SSH) Transport Layer Protocol, 2006.

[159] Akira Yokoyama, Kou Ishii, Rui Tanabe, Yinmin Papa, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, Daisuke Inoue, Michael Brengel, Michael Backes, and Christian Rossow. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID '16)*, pages 165–187, Paris, FR, 2016. Springer.

[160] Jonas Zaddach and Andrei Costin. Embedded devices security and firmware reverse engineering. *BlackHat USA*, pages 1–9, 2013.

[161] Jonas Zaddach, Luca Bruno, Aurelien Francillon, and Davide Balzarotti. AVATAR: A framework to support dynamic security analysis of embedded systems' firmwares. In *Proceedings of the 21st Network and Distributed System Security Symposium (NDSS '14)*, pages 1–16, San Diego, CA, 2014. Internet Society.

[162] Cliff Zou and Ryan Cunningham. Honeypot-aware advanced botnet construction and maintenance. In *Proceedings of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '06)*, pages 199–208, Philadelphia, PA, 2006. IEEE.