Bitter Harvest: Systematically Fingerprinting Low- and Medium-interaction Honeypots at Internet Scale

Alexander Vetterl
University of Cambridge
alexander.vetterl@cl.cam.ac.uk

Richard Clayton
University of Cambridge
richard.clayton@cl.cam.ac.uk

Abstract

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 low- and medium interaction honeypots at Internet scale with just one packet 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. For SSH honeypots we also determined their patch level and find that they are poorly maintained – 27% of the honeypots have not been updated within the last 31 months and only 39% incorporate improvements from 7 months ago. We believe our findings to be a 'class break' in that trivial patches cannot address the issue.

1 Introduction

Early detection of new attack vectors and abusive behaviour is a cornerstone of contemporary approaches to improving Internet security. Honeypots, resources that appear to be legitimate systems, have long proven effective in capturing malware, helping to counter spam and providing early warning signals about upcoming threats. Secure Shell (SSH), Telnet, and HTTP were a focus of early honeypot research. Since SSH is the de facto standard to login to servers over an unsecured network, SSH honeypots continue to be extremely valuable. With the rise of the Mirai botnet, which recruits a variety of IoT devices using Telnet, it became evident that Telnet honeypots remain a key source of information [3].

For the last decade, honeypot research has received limited attention, with efforts mainly focused on the monitoring of human activity and the provision of a realistic environment for humans to interact with. Attackers have a strong motivation to detect honeypots at an early stage as they do not want to disclose their methods, exploits and tools [21]. These attackers have attempted to distinguish 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 [25]. 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.

Finding deviations between applications and their fingerprints has a long history and a number of theoretical and prototyped approaches, mainly based on binary analysis, have been proposed [6, 7, 37]. Specific to fingerprinting honeypots, Holz et al. [21] showed that the execution time of attackers' commands may be significantly longer due to the additional overhead of logging and sandboxing the execution of the honeypot itself. Similarly, Fu et al. [17] found that attackers could use the latency of the networking layer to fingerprint honeypots. In 2015 Cymmetria Research summarized the known ways to fingerprint a variety of honeypots and provided a list of recommendations for honeypot developers [5].

All of these are *individual* findings, focusing on identifying single honeypot implementations. However, 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, which can number in the thousands, and show that this allows us to locate a large variety of honeypots by Internet scale scanning.

We believe this to be a *class break* in that patches to the current generation of honeypots cannot address the issue. Until these honeypots are given a new architecture, anyone with moderate capabilities has a plethora of extremely simple and quick methods of identifying that a honeypot is running on a particular IPv4 address and can thereby treat it differently than otherwise.

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 later) vulnerable to a known fingerprinting technique that was first disclosed in 2014.

2 Background

Honeypots are classified by the type of system that they emulate, such as web or email servers, or general purpose remote access 'shells'. Honeypots are further classified as low interaction (in this context merely collecting credential guesses), medium interaction (providing a higher level of interaction) or high interaction (allowing attackers full control of a machine).

High-interaction honeypots have significant value, but many people are unable to accept the risk that such honeypots may be used for DDoS attacks, malware distribution or the sending of email spam. However, low- and medium-interaction honeypots have proven effective as they are easy to deploy and to maintain, while at the same time potential harm is minimised. They are especially useful in collecting quantitative data about large-scale attacks. 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 [20]. Honeypots that support Telnet 'out of the box' were not widely available at the time of that study.

Table 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

Table 1 provides an overview of the honeypots that we considered in our study, which we now discuss in detail.

SSH Honeypots: In this paper 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 [13].

Many SSH honeypots have been developed over the years and one of the first SSH honeypots was Kojoney [24] 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 [23] which was developed from 2009 to 2015 but the Kippo author now recommends people use a forked project called Cowrie [31]. 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 2018-01, Cowrie (partly) emulates 34 commands [8]. In 2015, Deutsche Telekom included Kippo in T-Pot, a multi-honeypot platform "based on well-established honeypots" [9]. 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" [10].

Telnet Honeypots: Since mid-2016, and the rise of the Mirai botnet, there has been an increasing interest in Telnet honeypots. In this paper we consider four recent systems: MTPot, Telnet-IoT-Honeypot (TIoT), Telnet-Password-Honeypot (TPwd) and Cowrie.

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 above. Glastopf is also included within the latest versions of T-Pot.

3 Systematically fingerprinting honeypots

We present a new generic technique to systematically identify low- and medium-interaction honeypots based on protocol deviations. 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.

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.

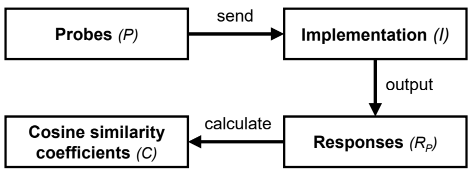


Figure 1: Steps to identify probes with distinctive responses, which can then be used for Internet-wide scans.

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 [40] and to measure domain similarity [28, 38]. 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 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. 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) 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.

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

[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

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 [26] and its intended update [2]. 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 4.4.1).

Table 5 in the Appendix gives the average similarity scores across all of the implementations. Overall, Kippo and Cowrie respond similarly to OpenSSH, with an average cosine similarity measure 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 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.

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 6 in the Appendix 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 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.

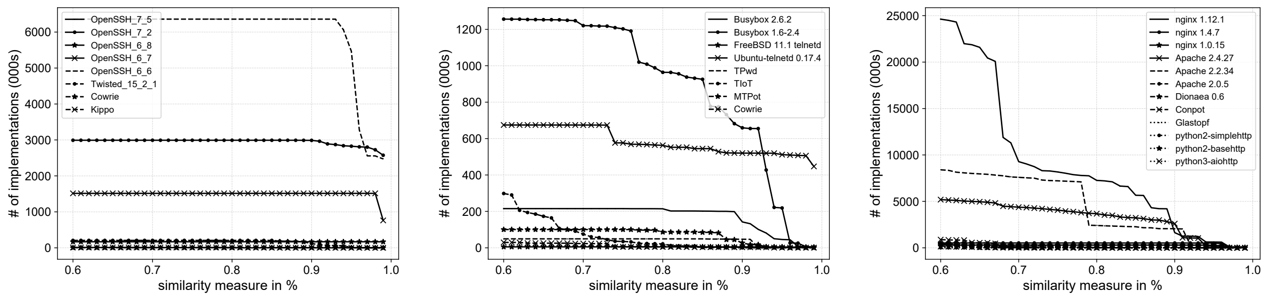
3.4 Protocol 3: HTTP/Web

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 [16] and RFC2518 [19] (including Webdav methods), the 123 non-alphanumeric ASCII characters

[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 [22]
[8]https://github.com/mushorg/glastopf/commit/bcbcebe
[9]https://github.com/mushorg/conpot/commit/74699fc
[10]https://github.com/DinoTools/dionaea/commit/02492e2b



(a) SSH implementations (b) Telnet implementations (c) HTTP implementations

Figure 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.

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 7 in the Appendix 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.

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 2 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 6.

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 [11]. 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.

4.1 Results

As shown in Figure 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 3.2) we have no easy way of defining a threshold at which cosine similarity score 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 4.2 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 4.4.2).

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 4.4.2). 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 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

Table 2: Results of the Internet-Wide Scan. *Scan 1 (SSH) was performed with the techniques outlined in Section 4.4

				SSH Honeypots		Telnet Honeypots				HTTP/Web Honeypots		
	Date	#ACKs	Sum	Kippo	Cowrie	TPwd	MTPot	TIoT	Cowrie	Dionaea	Glastopf	Conpot
Scan 1 (SSH)*	2017-09	18,196k	2844	906	1938							
Scan 2 (SSH)	2018-01	20,586k	2779	758	2021							
Scan 1 (Telnet)	2017-09	8,290k	1430			1	388	22	1019			
Scan 2 (Telnet)	2018-01	8,169k	1166			1	216	11	938			
Scan 1 (HTTP)	2017-10	58,775k	2616							139	2390	87
Scan 2 (HTTP)	2018-01	67,615k	3660							202	3371	87

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 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.

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 3a).

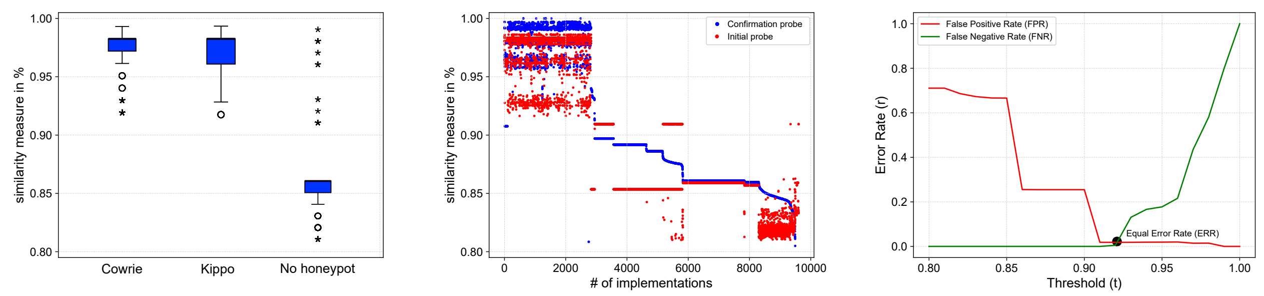
We further send the second-best probe to all of the 9 607 implementations that were initially been classified as SSH honeypots. As can be seen in Figure 3b, for all of the 2 779 fingerprinted honeypots the cosine similarity score for both, the initial probe and the confirmation probe (including the random 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.

4.3 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 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.



(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 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 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.

Table 3: 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%)

4.4.1 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.

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 [32]. 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.

4.4.2 Disconnection Messages

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

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.

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 ("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.

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".

4.5 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 3. 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. We find that the honeypot are hosted in 82 countries, with the majority being located at well-known cloud providers in the USA.

Table 4: 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

5 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. 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_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.

5.1 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. The value of the honeypot will drastically decrease, in particular its value in collecting data about large-scale attacks will be minimal. Thus, the main practical impact of our findings will be high switching costs for honeypot owners. The honeypot will need to be moved to a new IP address, perhaps a new hosting provider.

5.2 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 to be used in conjunction with Cowrie so there will be no difference in responses [36].

6 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 2018-06, 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 [11]. 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.

7 Related Work

Closest to this work, Bethencourt et al. [4] 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. Thus, over time, they could enumerate all sensors for particular systems. However, this approach requires that sensors make their data publicly available and our technique does not require this.

Brumley et al. [6] showed that by automatically building symbolic formulas from binaries they could find deviations in the protocol implementations of HTTP and NTP. Similarly, AUTOPROBE [39] generates fingerprints of malicious C&C servers through binary analysis. Focusing on protocol reverse engineering, Comparetti et al. [7] developed Prospex, a system to extract protocol specific information, but it may also be used to identify protocol deviations. 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 towards network hosts and analysing their response is a long established practice. Popular tools include Nmap [27] and more recently ZMap [14]. 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 [13]. Similarly. Feng et al. [15] performed an Internet-wide scan to characterize ICSs and their usage. 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, Albrecht et al. [1] present multiple vulnerabilities in SSH and perform an Internet-wide scan to obtain deployment statistics and estimate the impact of their new attacks. Similarly, Gasser et al. [18] showed that the rate of software updates for SSH is slow and that many SSH keys are reused on different hosts. While our results will not explain all of 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 [12, 34]. In 2004, Provos [33] developed Honeyd, a framework that simulates virtual computer systems and their TCP/IP stack to deceive fingerprinting tools. More recently, Mukkamala et al. [30] demonstrate that honeypots within virtual environments respond slower to ICMP echo requests than real systems. 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" [29]. In 2014, the SANS Technology Institute [34] reported that attackers issue the command `file /sbin/init` that returns dynamic content to fingerprint Kippo. In 2015, Cymmetria Research summarized such known cases for a variety of honeypots and outlined a list of recommendations [5]. However, unlike our automated and generic approach, these are individual findings inspecting single honeypot implementations.

Krawetz [25] developed a tool called *Honeypot Hunter* to fingerprint honeypots. This tool tests if emails sent through the compromised system were actually delivered as advanced functionalities are likely not emulated by low-interaction honeypots. Similarly, Zou and Cunningham [41] proposed fingerprinting honeypots based on their limited ability to participate in real attacks and execute malicious activities.

More recently, *Shodan* provides an online tool [35] 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.

8 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 address the issues 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 verisimilitude 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 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; we just need a new generation that is far less easy to identify.

Acknowledgments

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. We are grateful to Ross Anderson, Alastair R. Beresford, Alice Hutchings, Daniel R. Thomas, Sergio Pastrana and to the anonymous reviewers for helpful comments on this paper.

References

[1] Albrecht, M. R., Degabriele, J. P., Hansen, T. B., and Paterson, K. G. A Surfeit of SSH Cipher Suites. In *Proc. of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS '16)* (2016), pp. 1480–1491.

[2] Baushke, M. Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH), 2017. Available: `https://tools.ietf.org/id/draft-ietf-curdle-ssh-kex-sha2-09.html`.

[3] Bertino, E., and Islam, N. Botnets and Internet of Things Security. *Computer 50*, 2 (2017), 76–79.

[4] Bethencourt, J., Franklin, J., and Vernon, M. Mapping Internet Sensors with Probe Response Attacks. In *Proc. of the 14th USENIX Security Symposium (USENIX'05)* (2005), pp. 193–208.

[5] Blackhat. Breaking Honeypots for Fun and Profit, 2015. Available: `https://www.blackhat.com/us-15/briefings.html#breaking-honeypots-for-fun-and-profit`.

[6] Brumley, D., Caballero, J., Liang, Z., Newsome, J., and Song, D. Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation. In *Proc. of the 16th USENIX Security Symposium (USENIX '07)* (2007), pp. 213–228.

[7] Comparetti, P. M., Wondracek, G., Kruegel, C., and Kirda, E. Prospex: Protocol specification extraction. In *Proc. of the 30th IEEE Symposium on Security and Privacy (S&P '09)* (2009), pp. 110–125.

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

[9] Deutsche Telekom AG. T-Pot: A Multi-Honeypot Platform, 2015. Available: `http://dtag-dev-sec.github.io/mediator/feature/2015/03/17/concept.html`.

[10] Deutsche Telekom AG. T-Pot 16.03 – Enhanced Multi-Honeypot Platform, 2016. Available: `http://dtag-dev-sec.github.io/mediator/feature/2016/03/11/t-pot-16.03.html`.

[11] DNS-OARC. Don't Probe List, 2018. Available: `https://www.dns-oarc.net/oarc/services/dontprobe`.

[12] Dornseif, M., Holz, T., and Klein, C. NoSEBrEaK - Attacking Honeynets. In *Proc. of the 5th Annual IEEE SMC Information Assurance Workshop* (2004), pp. 123–129.

[13] Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., and Halderman, J. A. A Search Engine Backed by Internet-Wide Scanning. In *Proc. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)* (2015), pp. 542–553.

[14] Durumeric, Z., Wustrow, E., and Halderman, J. A. ZMap: Fast Internet-wide Scanning and Its Security Applications. In *Proc. of the 22nd USENIX Security Symposium (USENIX '13)* (2013), pp. 605–619.

[15] Feng, X., Li, Q., and Wang, H. Characterizing Industrial Control System Devices on the Internet. In *Proc. of the 24th IEEE International Conference on Network Protocols (ICNP '16)* (2016), pp. 1–10.

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

[17] Fu, X., Yu, W., Cheng, D., Tan, X., Streff, K., and Graham, S. On Recognizing Virtual Honeypots and Countermeasures. In *Proc. of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC '06)* (2006), pp. 211–218.

[18] Gasser, O., Holz, R., and Carle, G. A deeper understanding of SSH: Results from Internet-wide scans. In *Proc. of the Network Operations and Management Symposium (NOMS '14)* (2014), pp. 1–9.

[19] Goland, Y., Whitehead, E., Faizi, A., Carter, S., and Jensen, D. RFC 2518 – HTTP Extensions for Distributed Authoring – WEBDAV, 1999.

[20] Grudziecki, T., Jacewicz, P., Juszczyk, L., Kijewski, P., and Pawlinski, P. Proactive Detection of Network Security Incidents. Tech. rep., 2011. Available: `https://www.enisa.europa.eu/activities/cert/support/proactive-detection/proactive-detection-report`.

[21] Holz, T., and Raynal, F. Detecting Honeypots and Other Suspicious Environments. In *Proc. of the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop (SMC '05)* (2005), pp. 29–36.

[22] IETF. Telnet Options, 2011. Available: `https://www.ietf.org/assignments/telnet-options/`.

[23] Kippo. SSH Honeypot, 2015. Available: `https://github.com/desaster/kippo`.

[24] Kojoney. A Honeypot for the SSH Service, 2015. Available: `https://github.com/madirish/kojoney2`.

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

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

[27] Lyon, G. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, USA, 2009.

[28] Mahmood, A. N., Leckie, C., and Udaya, P. An Efficient Clustering Scheme to Exploit Hierarchical Data in Network Traffic Analysis. *IEEE Transactions on Knowledge and Data Engineering 20*, 6 (2008), pp. 752–767.

[29] Morris, A. Detecting Kippo SSH Honeypots, Bypassing Patches, and All That Jazz, 2014. Available: `https://www.redpacketsecurity.com/problems-i-have-found-with-kippo-honeypot/`.

[30] Mukkamala, S., Yendrapalli, K., Basnet, R., Shankarapani, M. K., and Sung, A. H. Network Based Detection of Virtual Environments and Low Interaction Honeypots Network Based Detection of Virtual Environments and Low Interaction Honeypots. In *Proc. of the Information Assurance and Security Workshop (IAW '07)* (2007), pp. 92–98.

[31] Oosterhof, M. Cowrie, 2016. Available: `https://github.com/micheloosterhof/cowrie`.

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

[33] Provos, N. Honeyd: A Virtual Honeypot Daemon. In *Proc. of the 12th USENIX Security Symposium (USENIX '03)* (2003), pp. 1–7.

[34] SANS Technology Institute. Kippo Users Beware: Another Fingerprinting Trick, 2014. Available: `https://isc.sans.edu/forums/diary/Kippo+Users+Beware+Another+fingerprinting+trick/18119/`.

[35] Shodan. Shodan – Honeyscore, 2017. Available: `https://honeyscore.shodan.io/`.

[36] Vetterl, A. OpenSSH Honeypot (sshd-honeypot), 2018. Available: `https://github.com/amv42/sshd-honeypot`.

[37] Wondracek, G., and Comparetti, P. Automatic Network Protocol Analysis. In *Proc. of the 16th Annual Network & Distributed System Security Symposium (NDSS '08)* (2008), pp. 1–18.

[38] XU, K., BUTLER, P., SAHA, S., AND YAO, D. D. DNS for massive-scale command and control. *IEEE Transactions on Dependable and Secure Computing 10*, 3 (2013), pp. 143–153.

[39] XU, Z., NAPPA, A., BAYKOV, R., YANG, G., CABALLERO, J., AND GU, G. AutoProbe: Towards Automatic Active Malicious Server Probing Using Dynamic Binary Analysis. In *Proc. of the 21st ACM SIGSAC Conference on Computer and Communications Security (CCS '14)* (2014), pp. 179–190.

[40] YEGNESWARAN, V., GIFFIN, J., BARFORD, P., AND JHA, S. An Architecture for Generating Semantics-aware Signatures. In *Proc. of the 14th USENIX Security Symposium (USENIX '05)* (2005), pp. 97–112.

[41] ZOU, C., AND CUNNINGHAM, R. Honeypot-Aware Advanced Botnet Construction and Maintenance. In *Proc. of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '06)* (2006), pp. 199–208.

Appendix

Table 5: 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 6: 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 7: 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	1.13e−3	0.09	0.20
python2-basehttpserver	D				X	0.64	0.01	0.17	0.08	0.08	0.11	0.02	0.08
python2-simplehttpserver	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	3.68e−7	3.34e−3	0.01
nginx 1.12.1	G							X	0.57	0.57	4.63e−4	0.04	0.17
nginx 1.4.7	H								X	0.57	4.68e−4	0.02	0.10
nginx 1.0.15	I									X	4.37e−4	0.02	0.11
Glastopf	J										X	2.53e−4	1.97e−4
Conpot	K											X	0.02
Dionaea 0.6.0	L												X