Counting Outdated Honeypots: Legal and Useful

Alexander Vetterl*, Richard Clayton† and Ian Walden‡
* Computer Laboratory, University of Cambridge, UK. Email: alexander.vetterl@cl.cam.ac.uk
† Computer Laboratory, University of Cambridge, UK. Email: richard.clayton@cl.cam.ac.uk
‡ Centre for Commercial Law Studies, Queen Mary University of London, UK. Email: i.n.walden@qmul.ac.uk

***Abstract*—Honeypots are intended to be covert and so little is known about how many are deployed or who is using them. We used protocol deviations at the SSH transport layer to fingerprint Kippo and Cowrie, the two most popular medium interaction SSH honeypots. Several Internet-wide scans over a one year period revealed the presence of thousands of these honeypots. Sending specific commands revealed their patch status and showed 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. However, our paper reporting these results was rejected from a major conference on the basis that our interactions with the honeypots were illegal and hence the research was unethical. We later published a much redacted account of our research which described the fingerprinting but omitted the results we had gained from the issuing of commands to check the patch status. In the present work we provide the missing results, but start with an extended ethical justification for our research and a detailed legal analysis to show why we did not infringe cybersecurity laws.**

***Index Terms*—honeypots, ethics, ethical issues, measurement, intrusion detection, unauthorised access**

I. Introduction

Many security researchers find it valuable to deploy apparently insecure systems on the Internet in the hope of learning about new attacks by monitoring interactions with these ‘honeypot’ machines. The intent is that attackers cannot distinguish the honeypots from ‘real’ targets and so they will reveal novel techniques, allowing the defenders to deploy appropriate countermeasures.

We have been particularly interested in ‘medium interaction honeypots’, which are often used to collect quantitative data about large-scale, scan-based attacks. Typically medium interaction honeypots emulate key services such as SIP, SMTP, FTP, Telnet, and SSH with a subset of commands commonly executed by intruders, but in a more sophisticated way than ‘low interaction honeypots’. Their attraction over ‘high interaction honeypots’, that expose full operating system functionality, is that they are less likely to be compromised and so the maintenance cost and the risk involved in running them is minimised.

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. In earlier work [11], we showed how this architecture is fatally flawed because the Python libraries being used have a large number of minor differences in their implementation of the Internet protocols when compared with a ‘real’ system – particularly when it comes to the handling of errors. 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 then able to count how many medium interaction honeypots are currently deployed for the SSH, Telnet, and HTTP protocols.

What we have not previously reported is that 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 and 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 paper 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 paper we start by exploring in detail the legal situation in 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.

II. 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) [2]. So the usual caveats that ‘your jurisdiction may vary’ tend to be less relevant in this area.

A. Statutory Text

Our conduct described in this paper can be broadly divided into two kinds: scanning for, and interaction with, honeypots. Scanning for identification purposes is obviously a subset of the latter, but should be distinguished because it involves interaction with a broad range of systems, not only the target honeypots. In both cases, our conduct might appear to be 'unauthorised' or 'illegal' access; a form of computer integrity offence [12].

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 [6], 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 [2]. 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.

B. 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, firstly, 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. Secondly, 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 that we discuss in the second part of this paper, 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, perhaps after a number of iterations, the honeypot we are interacting with will present us with an impersonation of a shell prompt in a pretence that we have 'logged in'.

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 (one author of this paper) and Flanagan considered the entrapment issue in a comparative law approach [13]. 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.

III. 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 rapidly identify medium interaction honeypots at Internet scale should serve as a wake-up call for the people who deploy these honeypots – they would 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. Full details of who we informed and when can be found in our earlier paper [11].

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 [5]. 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 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, generally not to make it more secure but to remove 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 of Cambridge 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.

In retrospect this message was unnecessarily cryptic and caused some confusion to honeypot operators, 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.

IV. 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.

Our earlier paper [11] 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.

One of the first SSH honeypots was Kojoney 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[1] which was developed from 2009 to 2015 but the Kippo author now recommends people use a forked project called Cowrie.[2] 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) emulated 34 commands. In 2015, Deutsche Telekom included Kippo in T-Pot, a multi-honeypot platform "based on well-established honeypots" [3]. 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" [4].

A. 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 II-B 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.

There has been a long history between finding ways to detect honeypots and camouflaging their presence. Attackers have attempted to distinguish honeypots by executing commands within 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. More recently, honeypot fingerprinting has moved from issuing commands to finding protocol deviations [7], [9], [11].

These findings have raised the awareness of honeypot operators that their systems might be identified and so they not only update the shell, but also the protocol layer. Thus, without authenticating to the honeypots, solely based on the protocol interactions, we are able to get a estimate of the honeypots' patch level (Table I).

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 III, 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. These commands were: `uname -a`, to get more information about the host system, `ls -d`, added on 2016-09-05, and `tftp`,[3] added on 2016-11-02. 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 V.

B. 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 [5]. 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 our earlier paper [11]. This told us unambiguously when we had identified

[1] https://github.com/desaster/kippo

[2] https://github.com/cowrie/cowrie

[3] This command is issued without any arguments and solely to check if the command is implemented.

TABLE I
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	

an instance of Kippo or Cowrie. 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.

V. Results

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

A. Authentication Configuration

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 (65.5%). For 136 (11.2%) of the honeypots, all 500 passwords were successfully attempted, but failed. Table II 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`.

B. 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 [1], 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 V-A). 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%).

TABLE II
Authentication Attempts for Cowrie Honeypots

	6 passwords		500 passwords		500 passwords		500 passwords		500 passwords	
Outcome	Scan 1: 2017-03		Scan 1: 2017-03		Scan 2: 2017-06		Scan 3: 2017-09		Scan 4: 2018-01	
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%)

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.[4] 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.

VI. 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 [10]. 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 [8]. 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.[5]

VII. Conclusion

In earlier work [11] 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 paper 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.

Acknowledgments

This work was supported by the EPSRC [grant number EP/M020320/1]. We are grateful to Alastair R. Beresford, Alice Hutchings, Daniel R. Thomas, and to the anonymous reviewers for helpful comments on this paper.

References

[1] Anomali Inc., "Modern Honey Network," 2014. [Online]. Available: https://github.com/threatstream/mhn

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

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

[4] ——, "T-Pot 16.03: A enhanced multihoneypot platform," 2016, available: http://dtag-dev-sec.github.io/mediator/feature/2016/03/11/t-pot-16.03.html.

[5] DNS-OARC, "Don't probe list," 2017. [Online]. Available: https://www.dns-oarc.net/oarc/services/dontprobe

[6] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. Hernandez Ganan, M. van Eeten, K. Yoshioka, and T. Matsumoto, "Detect me if you... oh wait. An internet-wide view of self-revealing honeypots," in *IFIP/IEEE International Symposium on Integrated Network Management*. Washington DC: IEEE, 2019.

[7] A. Morris, "Kippo SSH honeypot detector," 2014. [Online]. Available: https://www.rapid7.com/db/modules/auxiliary/scanner/ssh/detect_kippo

[8] D. Ramsbrock, R. Berthier, and M. Cukier, "Profiling attacker behavior following SSH compromises," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '07)*, 2007, pp. 119–124.

[9] D. Sysman, I. Sher, and G. Evron, "Breaking honeypots for fun and profit," in *Blackhat*, Las Vegas, NV, 2015.

[10] A. Vetterl, "OpenSSH Honeypot (sshd-honeypot)," 2018, available: https://github.com/amv42/sshd-honeypot.

[11] A. Vetterl and R. 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.

[12] I. Walden, *Computer Crimes and Digital Investigations, 2nd Edition*. Oxford University Press, 2017.

[13] I. Walden and A. Flanagan, "Honeypots: A sticky legal landscape," *Rutgers Computer & Technology Law Journal*, vol. 29(2), pp. 317–370, 2003.

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

[5] 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.