NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

DECEPTION USING AN SSH HONEYPOT

by

Ryan J. McCaughey

September 2017

Thesis Advisor: Neil Rowe
Second Reader: Alan Shaffer

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE DECEPTION USING AN SSH HONEYPOT	**5. FUNDING NUMBERS**
6. AUTHOR(S) Ryan J. McCaughey	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

The number of devices vulnerable to unauthorized cyber access has been increasing at an alarming rate. A honeypot can deceive attackers trying to gain unauthorized access to a system; studying their interactions with vulnerable networks helps better understand their tactics. We connected an SSH honeypot responding to secure-shell commands to the Naval Postgraduate School network, bypassing the firewall. During four phases of testing, we altered the login credential database and observed the effects on attackers using the honeypot. We used different deception techniques during each phase to encourage more interaction with the honeypot. Results showed that different attackers performed different activities on the honeypot. These activities differed in total login attempts, file downloads, and commands used to interact with the honeypot. Attackers also performed TCP/IP requests from our honeypot to direct traffic to other locations. The results from this experiment confirm that testing newer and updated tools, such as honeypots, can be extremely beneficial to the security community by helping to prevent attackers from quickly identifying a network environment.

14. SUBJECT TERMS honeypot, deception, SSH	**15. NUMBER OF PAGES** 83
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

DECEPTION USING AN SSH HONEYPOT

Ryan J. McCaughey
Second Lieutenant, United States Army
B.S., University of North Georgia, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
September 2017

Approved by: Neil Rowe
Thesis Advisor

Alan Shaffer
Second Reader

Dan Boger
Chair, Department of Information Sciences

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The number of devices vulnerable to unauthorized cyber access has been increasing at an alarming rate. A honeypot can deceive attackers trying to gain unauthorized access to a system; studying their interactions with vulnerable networks helps better understand their tactics. We connected an SSH honeypot responding to secure-shell commands to the Naval Postgraduate School network, bypassing the firewall. During four phases of testing, we altered the login credential database and observed the effects on attackers using the honeypot. We used different deception techniques during each phase to encourage more interaction with the honeypot. Results showed that different attackers performed different activities on the honeypot. These activities differed in total login attempts, file downloads, and commands used to interact with the honeypot. Attackers also performed TCP/IP requests from our honeypot to direct traffic to other locations. The results from this experiment confirm that testing newer and updated tools, such as honeypots, can be extremely beneficial to the security community by helping to prevent attackers from quickly identifying a network environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

DMZ	demilitarized zone
Gbps	gigabits per second
HTTP	hypertext transfer protocol
IoT	Internet of things
IP	Internet protocol
Mbps	megabits per second
MSF	Metasploit framework
NMAP	network mapper
OS	operating system
SFTP	secure file transfer protocol
SMDB	server message block
SSH	secure shell
TTP	tactics, techniques, and procedures
VM	virtual machine

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I offer a special thank you to my advisor, Dr. Rowe, for the guidance he has given me throughout the process. You were always quick to answer any questions I had. Thank you for keeping me focused and working with me when you were out of the office.

I also would like to thank my second reader, Dr. Shaffer. Thank you for your quick responses, quality feedback, and patience throughout the editing process.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

With the recent growth of devices connected to the Internet, the security community has been faced with solving many new problems. Most users own more than one device, and the predicted number of Internet of Things (IoT) devices will be 26 billion by 2020 [1]. Today, 60 percent of the IoT devices connected to the Internet have security issues with their user interfaces [1]. Furthermore, Hewlett Packard Enterprise reported "80 percent failed to require passwords of sufficient complexity and length." If this trend continues over the next three years, Hewlett Packard Enterprise reported approximately 21 million IoT devices will be connected to the Internet that are not secured. This not only applies to the device itself but also the device's cloud and mobile infrastructure.

This increase in unsecured devices connected to the Internet presents problems to users and organizations. Malicious users take advantage of this negligence to gain unauthorized access to devices, using commonly used protocols such as SSH (Secure Shell). This could lead to an increase in botnets and the amount of malicious traffic across networks.

One example of an IoT botnet is the Mirai botnet, which spreads by infecting millions of IoT devices. It uses these IoT bots to conduct distributed-denial-of-service (DDoS) attacks against targets. Each individual compromised node waits for commands and then floods targeted networks or servers with traffic.

Akamai [2] reported that the first quarter of 2017 saw a 17 percent decrease in total DDoS attacks from the last quarter of 2016. Although the number of attacks has decreased, Akamai observed half of attacks still remained between 250 Mbps and 1.25 Gbps in size. These attacks might not necessarily harm bigger organizations, but they could cripple smaller organizations that are less protected.

Akamai released another report to help inform consumers who are starting to use IoT devices in their daily lives. It recommends that users change the default credentials on these devices and entirely disable SSH services. However, if the user is unable to

disable SSH on the devices or uses that service regularly, Akamai suggests the user create firewall rules to prevent SSH access to the device from outside the internal network. Akamai also recommended establishing firewall rules to deny the device communication with unknown IP addresses to prevent tunneling [3]. In addition, many organizations continue to use unsecured servers, and also have SSH enabled on their systems without knowledge that this service is running.

Computer-security professionals use several techniques to gain an advantage over malicious users that attempt to penetrate systems. Setting up a honeypot on the network has been an important method since the early 2000s. Honeypots are decoy computer systems intended for no other purpose than to collect data on attackers. They gather information about attackers' sessions secretly, while allowing them to operate normally [4].

Honeypots allow researchers to analyze a variety of malicious user behaviors on a system. To connect to a computer remotely, users often enable SSH, which allows them to communicate on a secure connection [5]. An SSH honeypot specifically logs the username and password combinations from users attempting to connect into the system. Once they have gained access, their commands and input provided are logged.

If a honeypot has been on the Internet for an extended period, malicious users can fingerprint the machine and determine it is a honeypot by using a variety of techniques. Source code is often available to the general public, so attackers can analyze this code and write programs to determine whether the system is a honeypot. Honeypots also exhibit different timing than non-honeypot machines, and this can be noted. As a result, malicious users and security professionals are in a continuous battle to be one step ahead of the other [6]. Security professionals develop new tools and provide updates to existing tools to reduce fingerprinting on honeypots. It is important to test updated honeypots to see the effectiveness of these newer tools.

This thesis explored the tool Cowrie, an SSH honeypot that can be used to analyze specific deterrence methods to better understand attacker motives inside a system. It can, for example, study access to directories and the ease of access to the

system. We ran a Cowrie honeypot exposed to the Internet for two-and-a-half months and recorded 2,071,823 events.

Chapter II explores honeypots and their risk of being fingerprinted. The Cowrie tool used in this study and the problem this thesis is attempting to solve are introduced in Chapter III. The methodology of the experiment conducted and the setup of the honeypot are discussed in Chapter IV. A summary of the data collected is in Chapter V. Lastly, thesis conclusions, problems, and potential future work are described in Chapter VI.

THIS PAGE INTENTIONALLY LEFT BLANK

II. PREVIOUS WORK

To gather intelligence on adversaries in the cyber domain, security professionals run decoy systems known as honeypots to deceive unauthorized users into gaining access to these systems. By allowing unauthorized users to access honeypots, security professionals can study their tactics to uncover new methods that attackers may be using to exploit operational systems. Honeypots can be placed in multiple locations on a network, including outside a DMZ, inside a DMZ, and inside an internal network, as displayed in Figure 1 [4]. To deceive attackers and allow them to interact with the system, a full-interaction honeypot must appear to be a normal host. As a result, full-interaction honeypots can provide beneficial information but also present risks, as they provide attackers with full access to their systems.

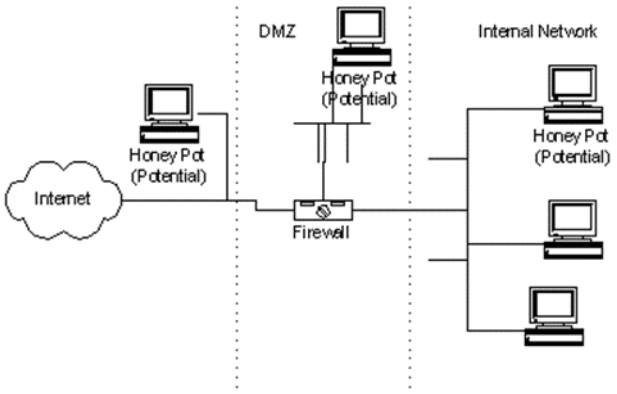

Figure 1. Honeypot deployment locations. Source: [4].

A. HONEYPOT CLASSIFICATION

Honeypots can be distinguished by three dimensions. One dimension is whether they are used for research or production purposes [7]. The second dimension is the environment a honeypot runs on. The third dimension relates to the interaction level that the honeypot provides the user.

1. Usage

Research honeypots gather large amounts of data on attackers. For example, the military can use information gathered internally to learn new malware and tactics being used. With a well-concealed honeypot, attackers would not know their methods have been revealed and would continue to operate as normal, while the military can establish safe countermeasures for the observed tactics. Also, organizations can use research honeypot information to patch systems where attackers have exploited a vulnerability, release reports on current methodologies, and expose the IP addresses of attackers.

On the other hand, production honeypots provide commercial organizations with added protection specifically for their internal networks [7]. A company may place honeypots inside the DMZ, or outside connecting to the Internet, to monitor who is trying to access their internal network systems, and what information they are seeking.

2. System

Honeypots can be classified by whether they are implemented in a virtual or physical environment. Honeypots running in a virtual environment run on virtual machine monitor (VMM) software such as VMware or VirtualBox. This software allows for downloading an ISO file and running an operating system virtually. An advantage of running software in a virtual environment is the use of snapshots. Users can take a snapshot of a VM's OS execution state, and revert back to it at any time. This is particularly useful if an attacker has made major modifications to an operating system that are difficult to undo. It is also useful if an attacker is able to break out of the VM, since then they can determine what environment they are running in, and proceed with caution [8].

Physical machines running honeypots can be very beneficial, as they lead attackers to believe they are on a real system. In addition, virtual honeypots tend to have limited memory and processing power, and have a higher probability of being detected than physical honeypots. Instead of snapshots, physical machines can undergo regular backups to trace what the attacker did on the system. However, physical machine

honeypots are typically more expensive to create and maintain than virtual ones, and less flexible for rapid reconfiguration.

3. Interaction

Honeypots can also be distinguished based on the interaction level they provide the user. Low-interaction honeypots open service ports, and then wait for a user or program to interact with the system, but do not do much beyond that [9]. By opening specific ports, their goal is to attract attackers into exploiting vulnerabilities in the services running on that specific honeypot, and record the initial steps in that exploitation. Low-interaction honeypots have many uses, such as web crawlers (Thug), logging web attacks (Glastopf), network emulation (HoneyD), and imitating SCADA/ICS systems (Conpot) [10].

Medium-interaction honeypots allow an attacker to interact with the system directly, and to exploit the target. Their services are still emulated, but appear to be more realistic. These honeypots provide information to researchers that allow them study the tactics of the attackers and gain a better understanding of their methodology. Medium-interaction honeypots are more expansive than low-interaction honeypots, allowing the attacker to perform more interactive activities such as uploading files and manipulating the file system. An example of a medium-interaction honeypot is Kippo, which emulates SSH and logs brute force attacks and attacker shell interaction [11]. Oosterhof [12] has added multiple features to Kippo to create his own version, Cowrie.

High-interaction honeypots use real systems and services to deceive and coerce attackers. Data collected from high-interaction honeypots looks more like data collected from real-world attacks [9]. Because of this higher level of attacker interaction, these honeypots require more care and closer monitoring to prevent damage to the host network running them.

B. HONEYPOT DATA

The primary goal of implementing a successful honeypot is to gather information. To accomplish this, data control, data capture, and data analysis need to be satisfied [13].

Data control is harder to implement, since attackers expect the system to perform as if it were an actual host. When the honeypot overly restricts the actions and commands that attackers can perform, they can become skeptical. Too loose control, however, allows attackers too much access to communicate with other systems and attack other targets through methods such as SSH tunneling. Determining the right amount of privilege and freedom the attacker has is something one must consider when implementing a honeypot on a network [13].

Data capture is the second data requirement that needs to be addressed. Communications on certain services can be encrypted, for example, with SSH [13]. With attackers using encrypted protocols, retrieving session information becomes more difficult. In addition, the data captured needs to be logged for future analysis. If these logs are stored on the honeypot itself, the attacker could delete them from the system. Storing captured data logs outside the honeypot is important and aids data integrity.

Lastly, once the captured data is stored in a separate location, it needs to be analyzed [13]. Analyzing honeypot data to find new trends, threats, and methodologies can help to better understand the motives and techniques of the attackers.

C. HONEYPOT RISKS

Although deploying a honeypot on a host network can help to secure that network, there are some risks associated with their implementation. As mentioned earlier, allowing malicious users to infiltrate a honeypot system would provide them with elevated access to that system as if it were their own. Attack methods such as SSH tunneling or allowing the honeypot to operate as part of a botnet are enabled, and may be attributed to the honeypot [13],[14]. In particular, honeypots bring risks to the internal network as they promote “an aggressive atmosphere” [14].

1. Detection and Fingerprinting

A honeypot is most effective when it completely deceives users into believing they have hacked a real system; therefore, attackers often use fingerprinting to determine if they are in a honeypot. Different versions of software often handle inputs in different

ways under different operation systems. These differences can help an attacker determine what software and what system are being used on a honeypot [14]. This can be done through testing various inputs or analyzing the source code and writing scripts to determine the software. By pinpointing responses not typical of any known operating system, users are able to successfully fingerprint the new device and pinpoint a honeypot.

Once a honeypot is identified, attackers would do one of two things. First, they may not waste their time further on the honeypot as they know it is not a real system. Second, they may give the honeypot invalid information in order to degrade the data the honeypot is collecting [13]; analyzing such corrupted data would not lead to any significant findings. In either case the honeypot is no longer useful against that attacker, and perhaps others with whom the attacker communicates.

Many honeypot implementations are open-source, thus providing the honeypot's source code to the public is necessary to allow legitimate users to test the honeypot and collect data. Over time, attackers can analyze the source code in depth and find bugs or vulnerabilities. This static analysis is unavoidable. Attackers can also perform dynamic analysis on the honeypot, and based on its behavior, can make educated guesses to determine the environment.

Fingerprinting the honeypot gets significantly harder as the degree of interaction increases, since low-interaction honeypots generally have a higher chance of being detected, while high-interaction honeypots do not. Before gaining access to the system, an attacker would perform a vulnerability scan on the target. The low-interaction honeypot Dionaea, which attempts to capture malware, is vulnerable to fingerprinting from NMAP scans alone [15,][16]. FTP, MySQL, and SMDB services are also detected by NMAP scans. An NMAP scan on Dionaea would reveal the SMBD workgroup name as "WORKGROUP" and the Homeuser name as HOMEUSER-XXXXXX (x = integers) [16]. In addition, another low-interaction honeypot, HoneyD, can give invalid HTTP replies [17]. Another honeypot, Glastopf gives the user invalid error replies when they try to access a folder not located in the vulnerability group [16]. These honeypots are all examples of ones that are susceptible to fingerprinting by attackers.

D. KIPPO

Kippo, as mentioned earlier, is an SSH honeypot that logs brute-force logins and shell interactions. Yahyaoui [11] ran two separate Kippo honeypots over several months to test their ability to detect attackers. One of these honeypots ran in a virtual environment for four months, using VMware as a virtual platform. The other ran in a physical environment for three months. Analyzing the results from the two honeypots, it appears that the login database was altered to make it difficult for users to have successful logins. Users had limited shell interaction with these two honeypots. He found that the quantity of attacks was similar for both environments, but the shell interaction varied. He also found more unique IP addresses attempting to gain access to the virtual honeypot over the physical honeypot. These experiments by Yahyaoui focused on username and passwords used to gain access to the system.

In other experiments, researchers discovered inputs that would alert attackers that the target was a honeypot [17]. Morris [17] showed that a ping to an unknown IP such as 999.999.999.999 on a Kippo honeypot would display that the machine is up and running, as seen in Figure 2.

```
$ ssh root@1.1.1.1
Password:
root@devops008:~# ping 999.999.999.999
PING 999.999.999.999 (999.999.999.999) 56(84) bytes of data.
64 bytes from 999.999.999.999 (999.999.999.999): icmp_seq=1 ttl=50 time=45.4 ms
64 bytes from 999.999.999.999 (999.999.999.999): icmp_seq=2 ttl=50 time=40.3 ms
...
```

Figure 2. Kippo ping of an invalid IP address. Source: [14].

Morris also pointed out that when sending a message with carriage returns, Kippo gives the error “bad packet length 168430090”, whereas OpenSSH gives the error “Protocol mismatch” [17]. Using this information, he wrote a script to detect whether a host is running Kippo; the script has since been added to the Metasploit Framework (MSF) database as an auxiliary called “detect_kippo” [18]. The MSF database contains scripts used for penetration testing, which can also be used by malicious users.

Another possible fingerprinting method involves the username and password combination of “root/123456,” as this is the default Kippo username and password [19]. If an attacker is able to connect using this combination, they potentially would be suspicious of being inside a Kippo honeypot and could disconnect immediately. Attackers would ignore this system as they feel it is not worth their time.

E. CONCLUSION

This chapter has explained the basics of honeypots and identified their characteristics, benefits, and risks. It is important to understand that attackers may be able to fingerprint honeypots, which can reduce or corrupt results. Nonetheless, honeypots are an essential tool for security analysis, and new and improved ones are appearing frequently.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TOOL SET

The previous chapter discussed honeypots, including how they are deployed and the risks of setting up a honeypot inside a network. This chapter describes the particular honeypot we tested.

A. PROBLEM

With attackers having the ability to detect the environment in which a full-interaction honeypot is situated, deploying it successfully can be difficult. Manipulation of the username and password database on the honeypot can change the number of users able to access the system. Many of the newer devices attached to the Internet use login credentials with default username and passwords. Although this is a bad security practice, it could be a good tactic for a honeypot in encouraging attackers to access it. However, allowing multiple usernames to successfully login to the honeypot increases the risk it would be detected. For example, if a user performs a dictionary attack that provides two different passwords for the same username, neither password should be expected to work; otherwise, the user would become suspicious of the system. At the least, the user would run further tests to determine the environment. If they remain in the system, they would likely proceed very cautiously.

This thesis explores the effectiveness of the Cowrie honeypot, which is based on the Kippo honeypot described earlier, and tests its fingerprinting methods. Prior to the experiment, we were curious to see whether we would gather different results when we altered the username and password database. Specifically, the interactions the users had with the honeypot. For example, we wanted to determine if there was a difference in interactions from those who ran dictionary attacks from those who only used one username to login on the honeypot. Furthermore, we wanted to observe if attackers showed a degree of persistence. If users had their session terminated, they could move on to other targets and scan other networks. The users could also attempt to regain access. Finally, this thesis analyzed fingerprinting methods used.

Prior to the experiment, it was assumed that a honeypot is more effective when its users do not know they are inside a honeypot, and that the longer users remain in the honeypot, the more likely they are to recognize it. We assume activity would change once the username and password database is changed. When attackers are able to gain access to the honeypot using a dictionary attack that uses default username and password credentials, we suspect their activity to differ from that of an attacker who has to use a more creative dictionary or manually login. An attacker who uses a simple dictionary attack may try to use our host as a bot or an attack pivot point. Attackers who used only one username would have conducted some form of reconnaissance. These users could be more interested in information location on the machine.

We also assume attackers would lower their guard when they cannot determine the honeypot's login credentials. As a result, if they can gain access after multiple attempts, they would more likely believe it is a real, hardened system.

B. COWRIE HONEYPOT

Cowrie is software created by Michel Oosterhof for building and running SSH honeypots. It has a file system that resembles a Debian 5.0 installation, and can be used to add and remove files [20]. It supports SFTP, which allows for secure transfer of files using SSH. Files uploaded to the honeypot are placed inside a directory and accessible for future investigation. Cowrie also logs direct-tcpip requests [20], which logs and then redirects SSH forward requests to other addresses [21]. All logs are formatted in json format and are broken down by date. Cowrie accepts additional administrator commands, including sleep, sync, uname –r, dir, help, and jobs. Support for these extra commands reduces the chance of the honeypot being fingerprinted and allows the user to interact more fully with the honeypot. If a user issues a certain command to the system, but receives an error message that it is an "unknown command," they would likely second-guess the machine's validity.

The SSH protocols in Cowrie have also been updated. Oosterhof has enabled the "diffie-hellman-group-exchange-sha1 algorithm" in the honeypot [20]. However, he also notes that attackers can analyze the ciphers provided by the SSH server and be able to

fingerprint the honeypot [20]. SSH servers provide users different ciphers to encrypt data. These ciphers alter the data into a non-readable form and can be decrypted with a key. The ciphers supported by Cowrie attempt to mirror OpenSSH (an SSH protocol suite). These ciphers are aes128-ctr, aes192-ctr, aes256-ctr, aes128-cbc, 3des-cbc, blowfish-cbc, cast28-cbc, aes192-cbc, and aes256-cbc [20]. As a result, a user could determine the supported ciphers and potently fingerprint the system. They could use the command "ssh –Q cipher" to display the ciphers supported. Oosterhof also took previous fingerprinting methods used against Kippo into consideration, adding functionality, and altering error responses to mimic a real system. The commands used to install Cowrie appear in Appendix A.

We used the software Tableau to analyze the logs gathered from the Cowrie Honeypot [22]. Tableau displays information about events in the Cowrie logs. Tableau creates tables and graphs, and can graph data points based on geographic coordinates or country names. No geographical information is provided by Cowrie, so we used MaxMind database to convert the IP addresses into countries [23]. Python code was used to read IP addresses from a text file, convert the IP addresses to countries from the database, and print the countries to a new text file; see Appendix B for this code.

As seen in Table 1, Cowrie logs 16 different types of events. Each event has its own specific set of information, for example, username and password fields are logged for the login failure and success events, whereas the message itself is logged in the command input event, which captures the user's interaction with the honeypot.

Table 1. Cowrie events and parameters within each field

Events	Fields
Cowire.client.size	Timestamp, session, src_port, message, system, isError, src_ip, des_port, des_ip, sensor
Cowrie.client.var	Name, timestamp, message, system, value, isError, src_ip, session, sensor
Cowrie.client.version	macCS, timestamp, session, kexAlgs, message, system, isError, src_ip, des_ip, version, compCS, sensor, encCS
Cowrie.command.failed	Timestamp, message, isError, src_ip, session, input, sensor
Cowrie.command.input	Timestamp, message, isError, src_ip, session, input, sensor
Cowrie.command.success	Timestamp, message, isError, src_ip, session, input, sensor
Cowrie.direct-tcpip.data	Timestamp, sensor, system, isError, src_ip, session, des_port, des_ip, data
Cowrie.direct-tcpip.request	Timestamp, session, src_port, message, system, isError, src_ip, des_port, des_ip, sensor
Cowrie.log.closed	Timestamp, message, ttylog, system, isError, src_ip, sensor
Cowrie.log.open	Timestamp, message, ttylog, system, isError, src_ip, sensor
Cowrie.login.failed	Username, timestamp, message, system, isError, src_ip, session, password, sensor
Cowrie.login.success	Username, timestamp, message, system, isError, src_ip, session, password, sensor
Cowrie.session.closed	Timestamp, message, system, isError, src_ip, duration, session, sensor
Cowrie.sessoin.connect	Timestamp, session, message, src_port, system, isError, src_ip, des_port, des_ip, sensor
Cowrie.session.file_upload	Shasum, timestamp, message, system, filename, src_ip, outfile, session, sensor, IsError
Cowrie.session.file_download	Src_ip, session, shasum, url, timestamp, outfile, sensor, message

C. SETUP

We connected the Cowrie honeypot to the Internet using an AT&T line that NPS provided. This line was outside NPS's firewall and allowed users to scan our IP range. We used two static IP addresses, one for the host machine, and one for the honeypot virtual machine. The virtual machine connected to the host machine by bridged networking. We configured the host machine and virtual machine subnet masks, gateway addresses, and name-servers, with the host machine's IP set to .54, and the virtual machine to .53 (full IP addresses not given since testing is ongoing). Figure 3 shows the network configuration used for the honeypot tests.

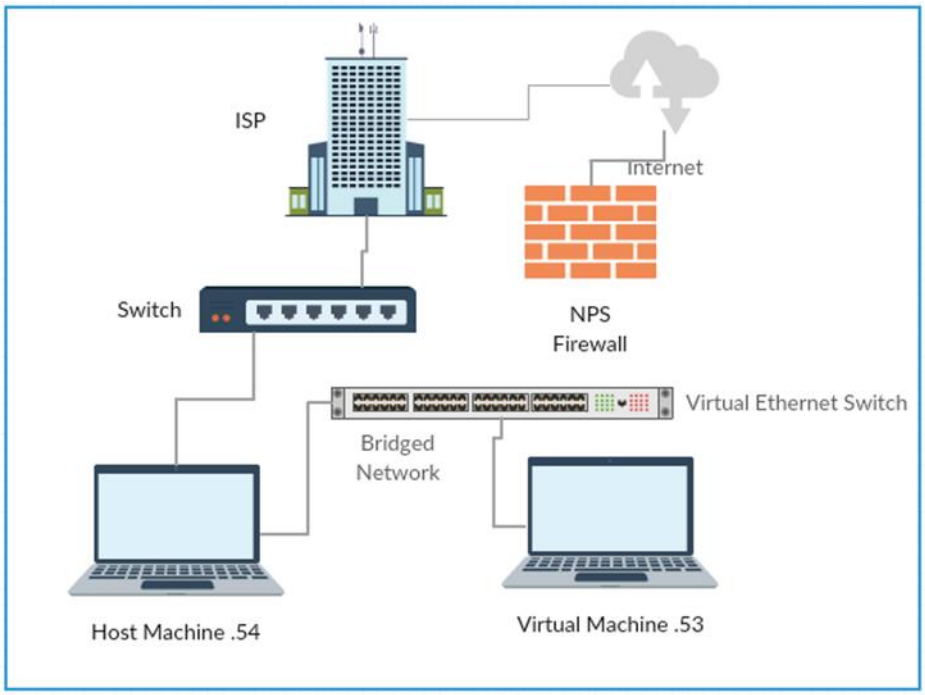

Figure 3. Diagram of the network setup

D. MACHINES

Information for the host machine appears in Table 2 and information regarding the virtual machine appears in Table 3. We ran the virtual machine using OracleVM Virtualbox 5.1.20.

Table 2. Hardware information for host machine

Operating System	14.04 LTS
Memory	1..9 GB
Processor	Intel Core 2 Duo CPU P8600 @ 2.40GHz x 2
Graphics	Mobile Intel GM45 Express Chipset x86/MMX/SSE2
OS type	32-bit
Disk	244.0 GB

Table 3. Hardware information for virtual machine

Operating System	GNOME Version 3.14.1
Memory	1008.2 MB
Processor	Processor Intel Core 2 Duo CPU P8600 @ 2.40GHz
Graphics	Gallium 0.4 on llvmpipe (LLVM 3.5, 128 bits)
Base System	Debian GNU/Linux 8 (Jessie) 32-bit
Disk	7.9 GB

There are differences between Kippo and Cowrie. Cowrie attempts to prevent attackers from being able to fingerprint the system, and also allows the user to have more interaction with the system than Kippo does. This chapter described the setup used to run Cowrie in a virtualized environment, and the next chapter discusses our experiment methodology.

IV. METHODOLOGY

This chapter explains the methodology of the experiment. It also explains how we tested whether this honeypot was vulnerable to specific fingerprinting methods.

A. DESIGN OF THE HONEYPOT EXPERIMENT

Our Cowrie honeypot ran for two and a half months. We started to collect data on May 1, 2017, and stopped the experiment on June 29, 2017. We resumed collection activity on August 3 and stopped it on August 15. The honeypot did not run continuously during this duration. Initially, the honeypot ran for three days, from May 1 to May 3, to confirm it was working correctly and that users could interact with it. We resumed on May 8, and during the next two weeks we left the honeypot running on the weekdays and shut down during the weekends to simulate business operations. On May 23 and May 31, we took it down for approximately two hours to change some configuration files. From May 31 to June 29, we left the honeypot running uninterrupted, and used this time to observe the attacker's actions on the honeypot. From August 3 to August 15, the honeypot was again left running uninterrupted.

During the first month, the honeypot was brought offline during weekends and at times to reconfigure files, which closed any user sessions. This would happen Friday afternoon, the closing time for most organizations. One goal was to see whether attackers would revisit the honeypot when it was bought back online the following Monday to indicate that the attackers had a degree of persistence. During the second month of the experiment we compared the attacker's behavior when they could remain in the system for an extended period of time. With the honeypot running for a month uninterrupted, attackers would not have any sessions closed. When we turned off the honeypot on June 29 to start analyzing the data, all active sessions were closed and no attackers were present in the system. After we determined commonly used login credentials, we ran the honeypot again in August.

B. CONFIGURATION

During the two-and-a-half months the honeypot ran, we changed the database configuration three times, breaking it up into four different phases. In this section, we describe the experiment phases corresponding to the database configurations, and the times corresponding to the dates the honeypot was running for each phase.

1. Phase 1

The first phase of the honeypot ran from May 1 to May 23. During this time, we left the username and password database as the default values from installation. Table 4 shows these values.

Table 4. Login credentials during Phase 1

Username	Rule
root	!root
root	!123456
root	*
richard	*
richard	fout

The username "root" would be denied with passwords that are "root" and "123456." Any other password provided with the username "root" could gain access to the honeypot. The second username that could gain access to the honeypot is "richard." These rules allow the password "fout" to gain access with the username "richard." Additionally, any password provided with this username would work.

This phase was implemented with the honeypot running during the weekdays only. We left the database with its default values to see whether any users could fingerprint our honeypot based on these credentials. We expected that almost all users could access our honeypot with this configuration. We would still record failed login

attempts, as other usernames would be denied. If attackers used the username "root", they would be likely to gain access since the system would accept most any password (other than the two exceptions in the rules). On May 23, we changed the honeypot's login database to see whether we would gather different results.

2. Phase 2

The second phase of the honeypot ran from May 23 to May 31. We ran this phase for eight days straight without shutting it down over the weekend. Table 5 lists the rules we implemented for this phase.

Table 5. Login credentials during Phase 2

Username	Rule
richard	!root

During this phase, the only username an attacker could use to successfully access our honeypot was "richard." All passwords were accepted except for "root." When we changed this, all users who were on the system during Phase 1 would have their sessions closed. If they tried to log back on at a later time using the old credentials, they would be unable to. This would indicate to the user that we had hardened the system.

We ran this phase for only a week for two reasons. First, with only one allowable username, we expected a significant decrease in the number of users able to gain access. We expected a concomitant increase in login attempts as users would try more login credential combinations. The goal of this shortened phase was to collect username and password combinations. Second, we were interested in seeing how many users would be able to login with the username "richard." We would be able to see if these users performed a dictionary attack against our system and then happened to use the word "richard," or if the user was able to login on the first attempt to indicate a possible fingerprinting method. If they closed their session and tried to log back on with "root," as

in Phase 1, they would fail. After this, we were interested to see what actions they would take on the honeypot.

3. Phase 3

The third phase of the honeypot ran from May 31 to June 29, and during this time, we ran the honeypot uninterrupted. Table 6 lists the credentials required for this phase.

Table 6. Login credentials during Phase 3

Username	Rule
root	!root
root	!123456
root	!12345
root	!password
root	!pass
root	*
richard	*

During this phase, we reverted to the original credentials, with some changes. We denied the passwords "123456," "12345," "password," and "pass" for the username "root." We kept this configuration for the entire phase, as we did not want to close any active sessions. We were satisfied with some of the results during Phase 1, so we chose similar rules for this phase. The four additional rules did not necessarily deny entry to users but attempted to better conceal that the system was a honeypot.

4. Phase 4

The fourth phase of the honeypot ran from August 3 to August 15. Table 7 shows the credentials required during this phase. In the first three phases, we accepted multiple

passwords for a single username, representing a one-to-many relationship, but during this phase the relationship was one-to-one.

Table 7. Login credentials during Phase 4

Username	Rule
user1	admin
admin	welc0me
support	support
user	password
root	welc0me

We created the login credential database based on commonly used usernames and passwords observed from the first three phases. We accepted the password "welc0me" with the usernames "admin" and "root". This was the most used password in the first three phases.

C. POSSIBLE FINGERPRINTING METHODS

During Phases 1, 3, and 4, attackers analyzing our login credential database could fingerprint our honeypot. During Phase 2, our honeypot was not as vulnerable so fingerprinting was not possible. We assumed attackers had the resources to determine the environment based on usernames and passwords alone.

We also tested the usefulness of an error message displayed to a user who provided an invalid IP address. Chapter II shows how Kippo displays an error message when a user attempts to ping an invalid host. Figure 4 shows the error message produced by Cowrie when a user attempts to ping an invalid IP address.

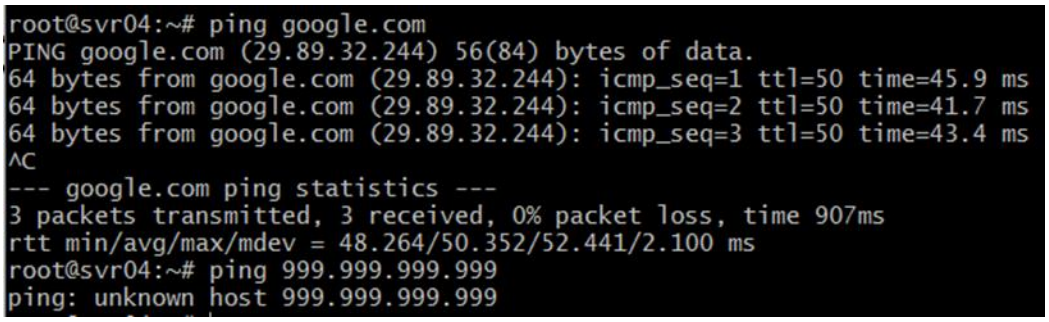

Figure 4. Ping on google.com and 999.999.999.999

Any IP address out of range that is pinged within Cowrie is displayed as an unknown host. This makes it harder for users to fingerprint our honeypot, but a vulnerability remains. When we ping google.com we see the IP address is 29.89.32.244, which is not Google's actual IP address but a fake IP address. If users compared this fake address to the actual address, then they could determine the response is invalid. Figure 5 displays the results from a successful google.com ping. The valid ping response has an IPv6 address and the invalid ping response has an IPv4 address. This is something to consider if we notice any ping commands on our honeypot.

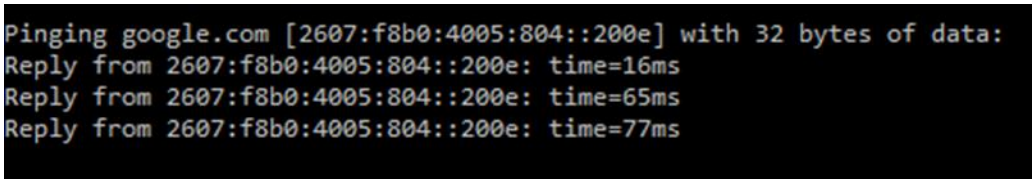

Figure 5. Ping on google.com

We also tested the Kippo detection auxiliary script on MSF against the Cowrie honeypot to ensure users would not be able to fingerprint our honeypot using this method. Figure 6 displays the results from the script.

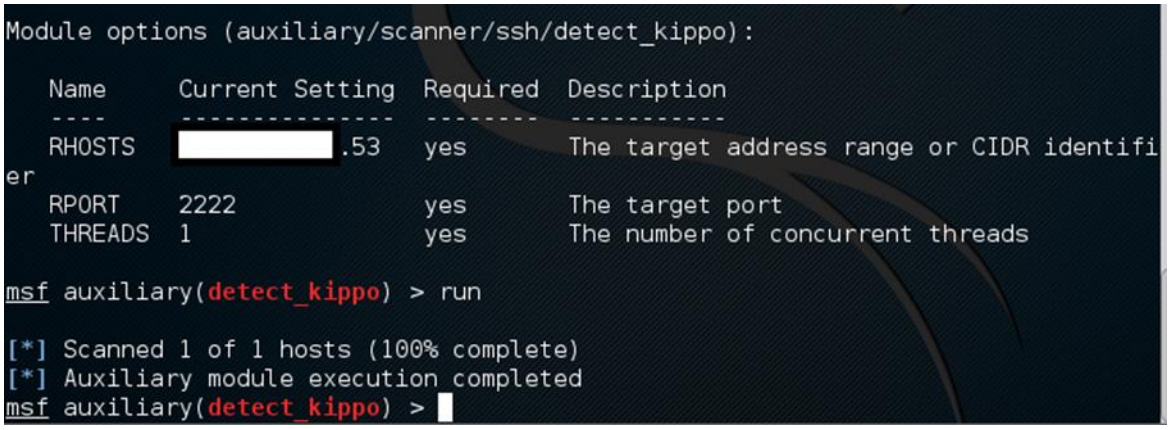

Figure 6. MSF script against Cowrie

The results shown in Figure 6 indicate that the MSF script could not identify our system as having Kippo functionality. Although this scan is intended for Kippo, Cowrie is based on Kippo and we wanted to test if it could behave similarly in this test.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DATA AND RESULTS

This chapter summarizes the data we collected from the honeypot over two-and-a-half months of operations. During this time we recorded 2,071,823 events, including two file uploads and 940 file downloads, and logged 10,314 successful commands and 325 failed commands for users interacting with the honeypot. During our testing, IP addresses from 89 different counties attempted to login to our honeypot.

A. INTERACTIONS DURING THE FIRST THREE PHASES

Once an attacker gained access to the system, we saw activities such as removing traces of their presence on the system logs, identifying system properties, performing file downloads through *wget* requests, and TCP/IP requests.

1. Logs

We observed 96 instances of users removing logs located in the honeypot *var* directory, a clearly suspicious activity. These logs included .bash/history, lastlog, and mailog. Other log deletion attempts appear in Figure 7, where the right column represents the number of occurrences for each command. After users deleted the logs from the system, they created a new file with an identical name. Even without this logged information, we could still determine changes made to the *var* directory by analyzing the creation timestamps of these files.

rm -rf /root/.bash_history	96
rm -rf /var/log/lastlog	96
rm -rf /var/log/maillog	96
rm -rf /var/log/messages	96
rm -rf /var/log/secure	96
rm -rf /var/log/wtmp	96
rm -rf /var/log/xferlog	96
rm -rf /var/run/utmp	96
touch /root/.bash_history	96
touch /var/log/lastlog	96
touch /var/log/maillog	96
touch /var/log/secure	96
touch /var/log/wtmp	96
touch /var/log/xferlog	96
touch /var/run/utmp	96

Figure 7. Commands used to remove and create logs

2. Unset History

Our honeypot also recorded 96 instances of users attempting to hide their activity on the system by using the *unset* command shown in Table 8. This command was entered on one line but the shell ran each partial command individually. By unsetting these values, the users' activities would not be recorded.

Table 8. Command used hide activity on the honeypot

Command	unset HISTORY HISTFILE HISTSAVE HISTZONE HISOTRY HISTLOG WATCH

3. Information on the System

Users also performed commands to gather system information (Table 9). The command *free –m* displays information about the memory of the system, such as how much memory is available, in use, and free. Command *ps -x* displays the processes running on the system and the respective process IDs. The *id* command provides the user with information about which group the user is in. The *uname* command returns information such as the kernel version, processor, and operating system. Finally, the *w* command provides information about who is logged onto the system and their login date.

These commands are able to help the attacker identify the environment they are on and the resources they have available.

Table 9. Commands to gain information about the system

Command	Number of occurrences we observed
free –m	203
ps –x / ps x	198
id / id –u	12
ls / ls –al / –d / -la	8
uname / uname –a / -m / -r	221
w	2

4. File Downloads

Users also performed *wget* commands to download files. These files that were downloaded were secretly copied to a directory of ours on the honeypot for future analysis. The majority of files downloaded were .sh files, shell scripts that are executable in a terminal. Table 10 provides examples of files downloaded and shows how they were saved on our system.

Table 10. URL downloads and the output filenames

URL	Outfile
http://27.118.21.217/sc.sh	dl/cb1d9c280fbdddf521946c9a6c026c1fa5 52e08e7a30ffcd2728744aedeaa6ec
http://77.247.178.189/bins.sh	dl/f88388a7250ab66c77d54834c0bd6422b 7b761935b0a0c8aca88d2f2248be58d
http://104.148.19.115/ys808e	dl/02ab39d5ef83ffd09e3774a67b783bfa34 5505d3cb86694c5b0f0c94980e5ae8
http://173.212.225.13/qbot/gtop.sh	dl/4a70a24516a51493cec2011c1e3e99f8a 365d4af8f66a49a34a518e83ca95363

We also recorded different IP addresses that downloaded the same file. Figure 8 shows four different IP addresses that downloaded the file xcv.sh from IP address 51.15.77.44 a total of eight times (right column shows number of attempts for each attacking IP).

http://51.15.77.44/xcv.sh	dl/b2e7dabc4736f1d1234239092b80fd07ac1cc66788b1 bc938fad08c3587da3d0	51.15.68.152	1
		88.249.199.76	1
		117.139.114.218	4
		195.22.127.35	2

Figure 8. URL download with different IP's and same file

The number of file downloads during the three phases was significantly different. During Phase 1, 10 file downloads took place over the course of three days. During Phase 2, no file downloads occurred. The beginning of Phase 3 had limited file downloads, but they started to increase as the honeypot ran. We observed a maximum of 123 file downloads on June 15. After June 15, the number of files downloaded per day never reached over 50 again. Figure 9 displays the total number of file downloads, by date, during the honeypot operations. The month is shown at the top of the horizontal axis and the date is at the bottom of the horizontal axis.

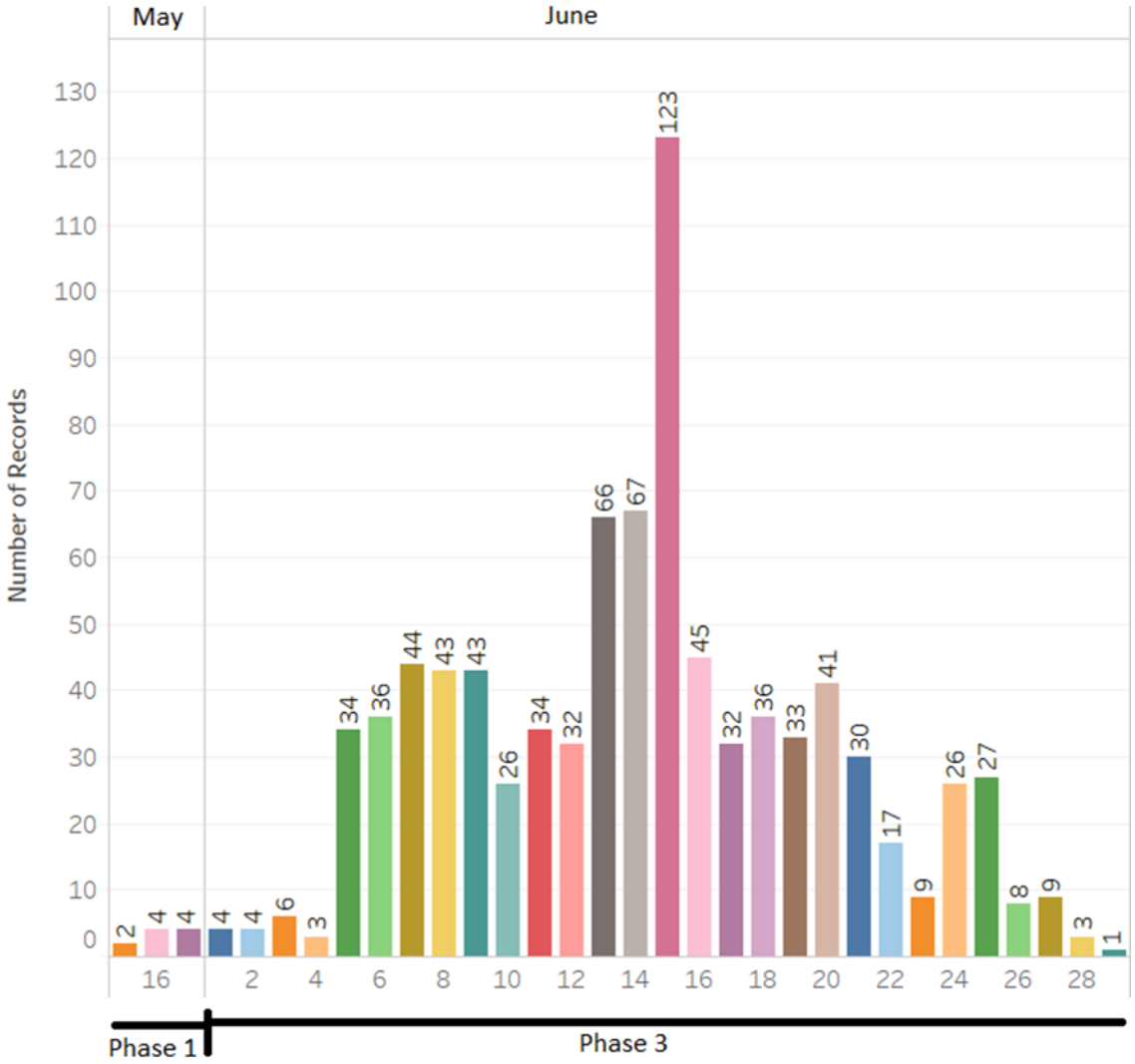

Figure 9. File downloads by date

Sixty-eight IP addresses performed a *wget* request to download files during Phases 1 through 3. Figure 10 displays the top 10 IP addresses that downloaded files, with the corresponding source country of each. The Netherlands accounted for the highest three instances of file downloads.

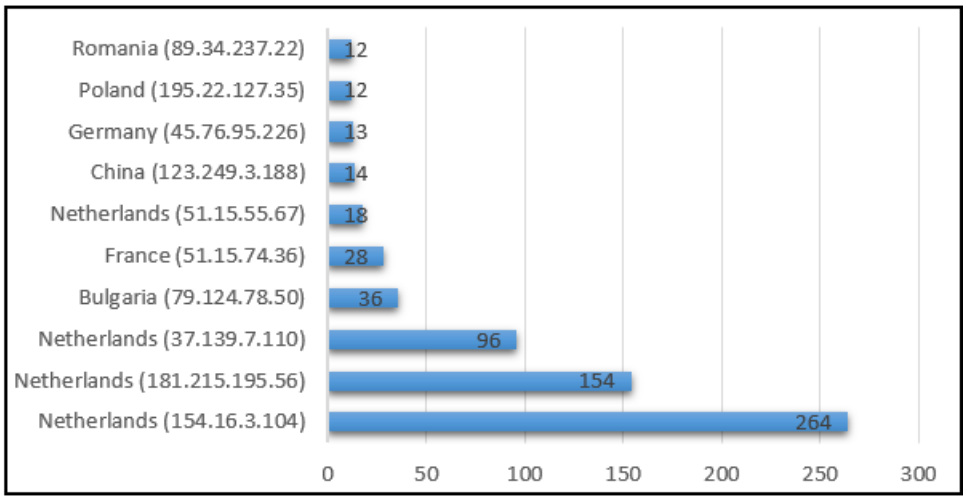

Figure 10. Top 10 IP addresses that downloaded files

The activity on the honeypot during these three phases consisted of removing traces, downloading files and collecting system information. The command used most often to gather information was *uname*. We noticed that file downloads varied significantly during each phase, and were the greatest during Phase 3. The top three IP addresses that downloaded files accounted for 514 of the 892 total downloads.

5. Commands

Figure 11 shows the top 10 IP addresses that provided input to the honeypot. This includes both successful and failed commands. Approximately 16 percent of inputs provided to the honeypot came from the IP address 195.22.127.83, which is five times the rate of the next highest IP address.

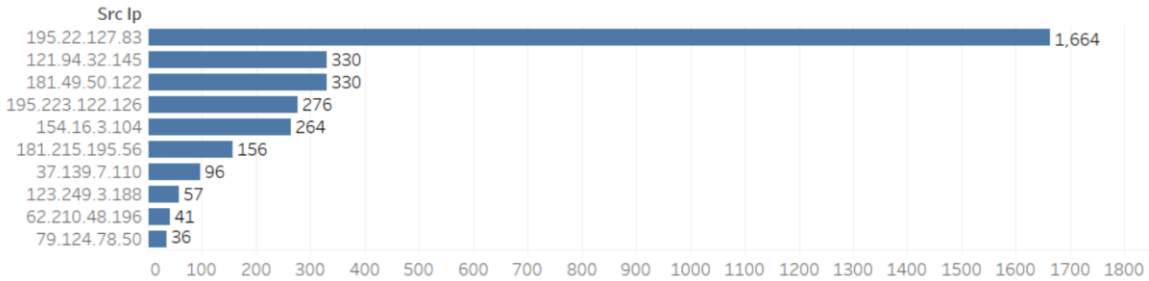

Figure 11. Top 10 IP addresses that provided inputs

We also combined all the IP addresses and gathered information based on country. Figure 12 shows that Poland was responsible for the majority of the inputs with 1697 records, and The Netherlands was the next highest with 580 records. By comparing the two graphs, we determined that 33 other inputs were provided by other IP addresses originating from Poland.

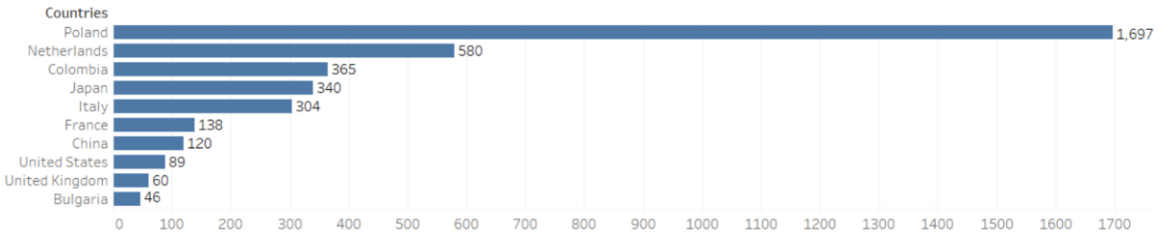

Figure 12. Top 10 Countries that provided inputs

Figure 13 shows the top 10 countries with failed input requests. The IP address 195.22.127.83 once again was an outlier, accounting for 83 percent of all failed inputs. Roughly 15 percent of its total inputs failed. The IP addresses 195.22.127.83 and 123.249.3.188 were the only two that appear in both failed inputs and overall inputs provided. The other eight addresses listed in Figure 11 do not appear in Figure 13.

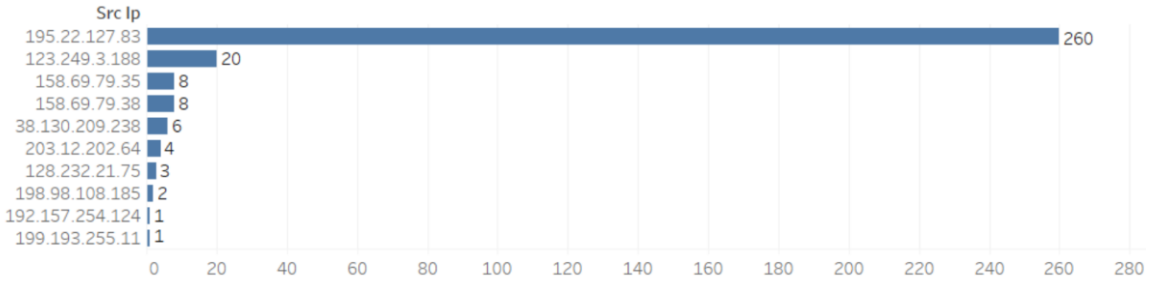

Figure 13. Top 10 IP addresses whose inputs failed

B. LOGIN ATTEMPTS DURING THE FIRST THREE PHASES

We recorded 222,641 login attempts during the first three phases of testing, of which 12,294 were successful and 210,247 were unsuccessful.

1. Usernames

During the first three phases, 4,539 different usernames were used to attempt login to the honeypot. Figure 14 shows the top 10 usernames. The username “root” was used a total of 179,437 times. During Phases 1 and 3, only five passwords were denied with username “root”, thus the majority of users who attempted login with “root” gained access to the honeypot.

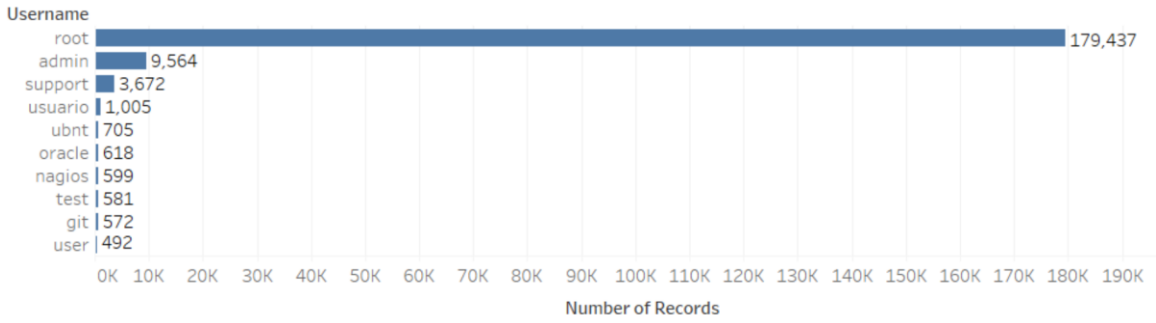

Figure 14. Top 10 usernames

2. Passwords

During the first three phases, we recorded 38,770 passwords used for login. During Phases 1 and 3, more passwords were used with a username other than “root”, and more were used with username “richard” during Phase 2. Figure 15 displays the top 10 passwords used. We expected to see common passwords such as “admin”, “12345”, and “123456”, but we were surprised to see the top password was “welc0me”.

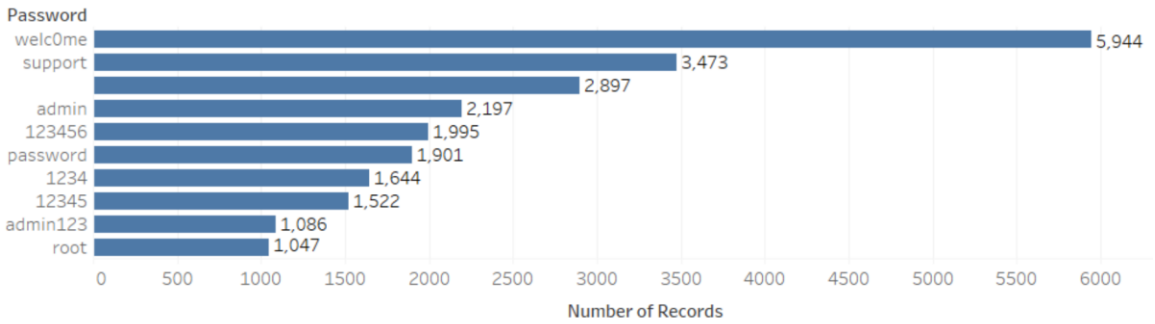

Figure 15. Top 10 passwords

3. Login Attempts by Date

During Phase 1, we recorded 1,145 successful login attempts. Only one successful login attempt took place during Phase 2, when on May 28 "richard/richard" was used for the login credentials. During Phase 3, we observed 11,148 successful login attempts. The date with the greatest number of successful login attempts was June 6. We added additional rules to deny access to the honeypot during Phase 3, but still observed an increase in the number of daily users gaining access to the honeypot. We speculate that this was due to the longer duration the honeypot was connected to the Internet, which allowed more users to scan its IP range. These daily login results are shown in Figure 16.

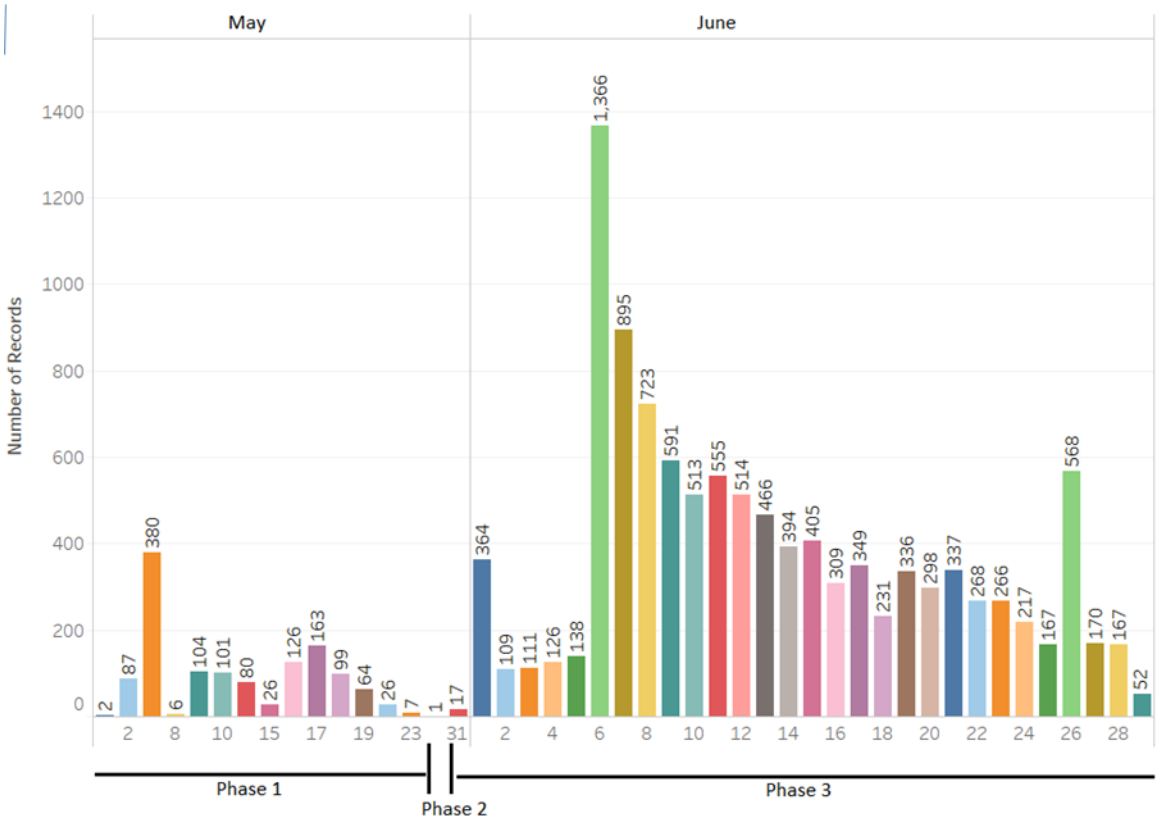

Figure 16. Successful login attempts by date

The number of failed login attempts during Phases 1 and 3 was less than 1000 for 35 of the 44 total days. During these phases, however, we recorded seven days with failed login attempts that exceeded 2000. During Phase 2, our honeypot recorded more failed login attempts, with May 28 being the highest with 69,552 failed attempts. The failed login attempts by date are displayed in Figure 17. The increase in failed login attempts during Phase 2 appeared to be due to the use of dictionary attacks. We observed numerous login attempts using the username “root” and a password that included common words, numbers, a combination of both, and random strings.

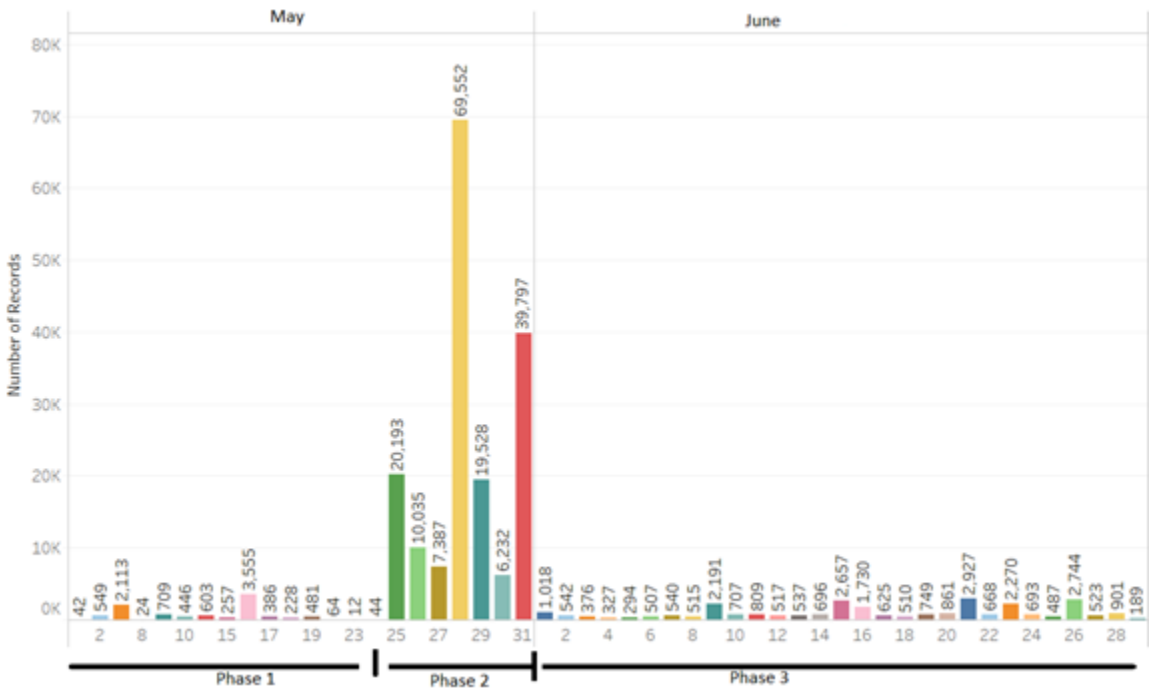

Figure 17. Failed login attempts by date

4. Countries and IP Addresses

Figure 18 shows the top 10 countries and the number of login attempts that each was responsible for. An overwhelming number of login requests were recorded from IP addresses located in China, with a total of 166,270 login attempts. No other country had more than 10,000 login attempts.

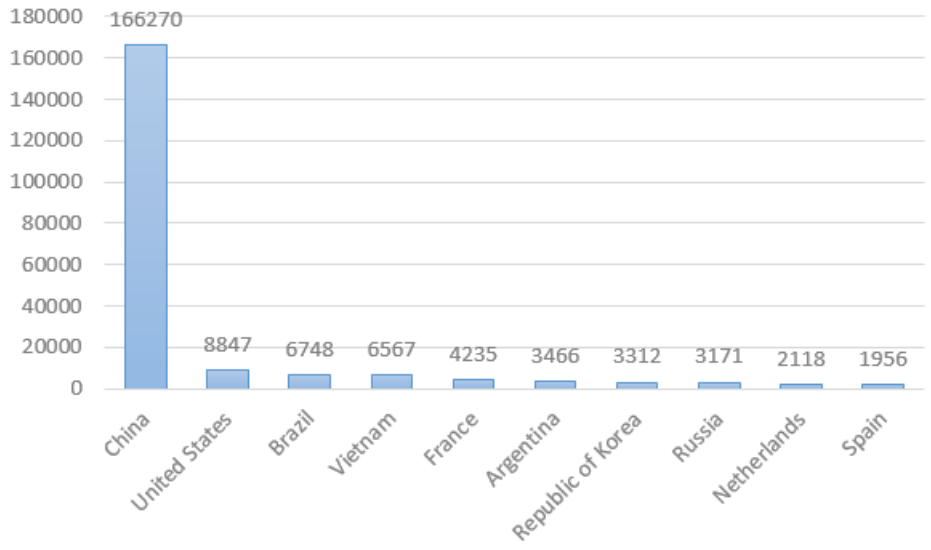

Figure 18. All login attempts by country

We analyzed the number of login attempts from countries with unique IP address. Although the vast majority of login attempts originated from China, most of them came from only eight IP addresses. During the experiments, we recorded multiple login attempts from a single unique IP address in a country, e.g., the address 61.177.172.55 attempted to login 19,657 times. We recorded these unique addresses only once, instead of the total number of login attempts. This data is displayed in Figure 19.

China still had the most unique IP addresses in attempting to log into our honeypot, but they were not outliers as they were with overall numbers. Eight of the top 10 countries from Figure 18 appear in Figure 19 with the exception of the Netherlands and Spain. All unique IP addresses that attempted to login are shown in Figure 20. From this geographical map, we were able to observe the distribution of login attempts from across the globe.

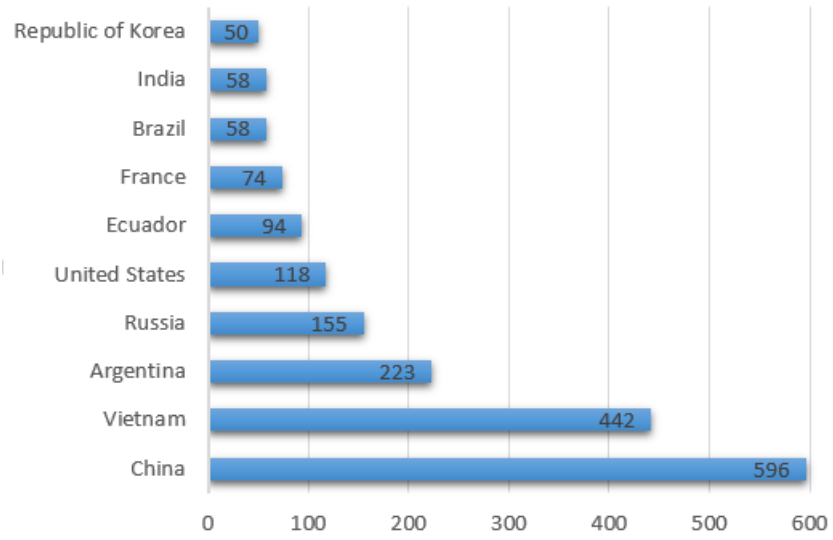

Figure 19. Top 10 countries with unique IP addresses

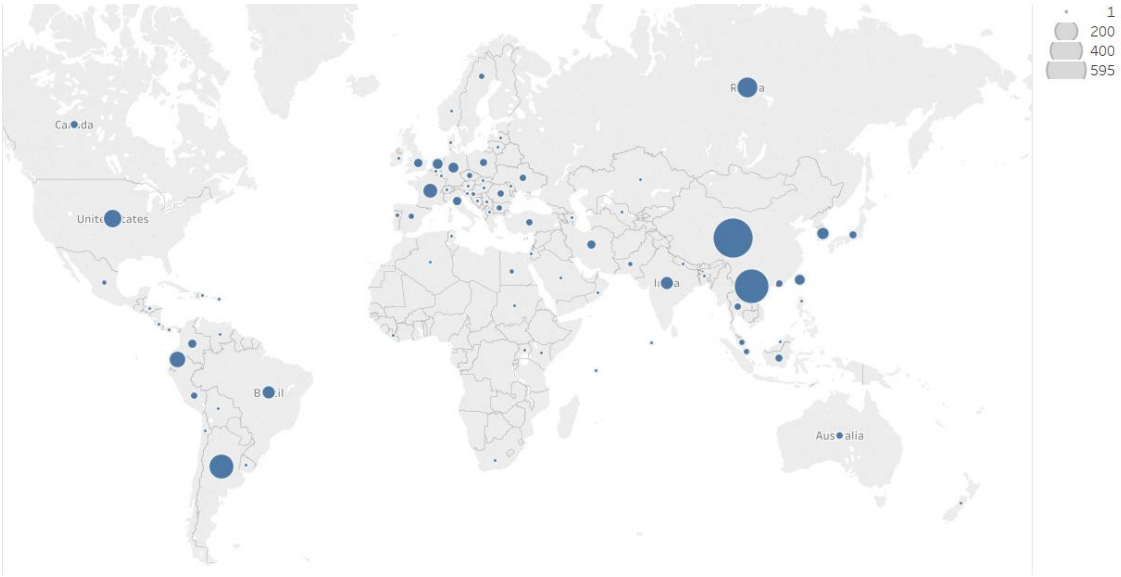

Figure 20. Geographical map of all unique IP addresses

Since the number of unique IP addresses attempting to login differed significantly from the overall number of login attempts, we looked at the failed login attempts associated with each IP address. Figure 21 shows the top 10 IP addresses that failed to login to the system. The top three all belong to the same 61.177.172.xxx subnetwork. All three of these IP addresses recorded 19,657 attempts, totaling roughly 60,000 failed login

attempts from this subnetwork, which was located in Lianyungang, China, and has the domain name chinatelecom.com.cn. As a result, China was an outlier in the total number of login attempts from each country.

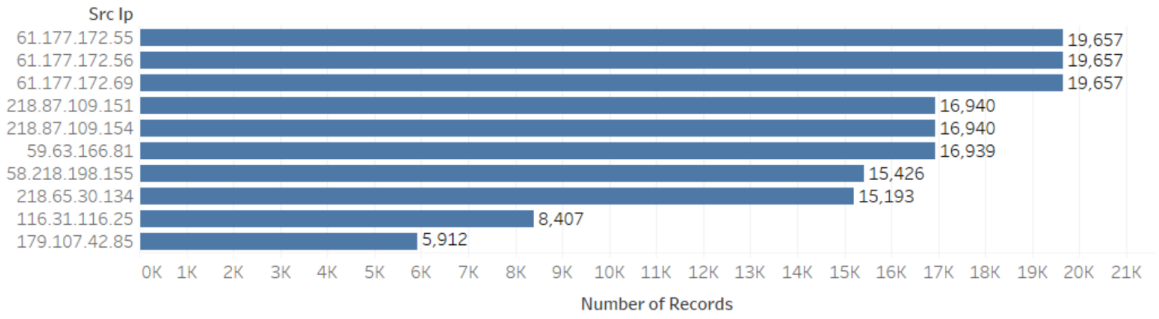

Figure 21. Top 10 IP addresses with failed login attempts

5. Multiple Successful Login Attempts

During the first three phases, we put our honeypot offline at certain times, which would close active sessions. In addition, the SSH server had a timeout request if there was inactivity for three minutes, and users also could exit the system. If one of these events occurred, the user would be required to log back on again. Figure 22 displays the top 10 IP addresses that had successful re-login attempts; these revisited the honeypot over 200 times.

The longest any user stayed on the system was 671,739 seconds, from June 6 to June 14. This user's IP address was 144.76.139.164. This user provided no commands but had a total of 16,412 direct-tcp/ip requests. The highest occurrence among all durations that users remained on the system was two seconds, which occurred 19,311 times. The top 10 most frequent times that users remained on the system, to the nearest second, were 0, 1, 2, 3, 4, 5, 6, 7, 36, and 37 seconds.

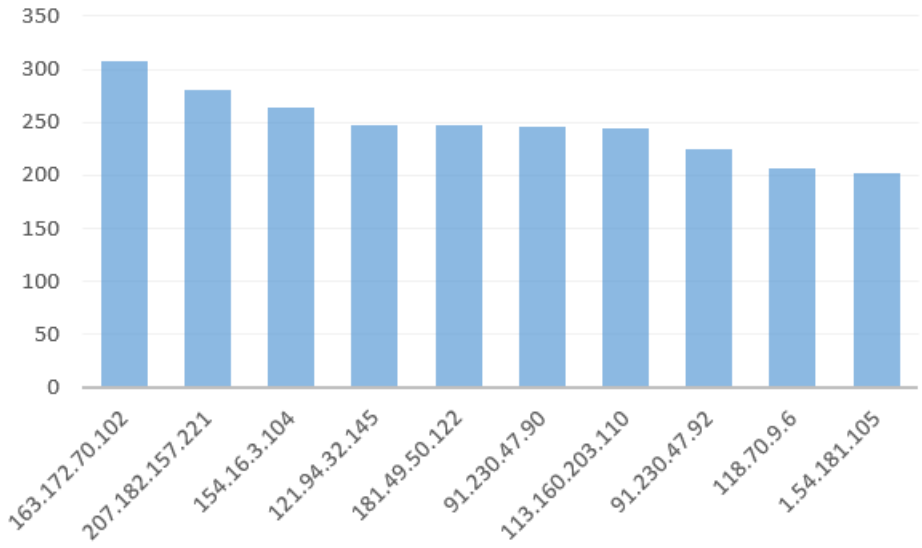

Figure 22. Top 10 IP addresses with successful login attempts

6. Username "richard"

During the first three phases, attackers providing the username "richard" gained access to the honeypot. We were curious to see whether these users used a dictionary attack against the honeypot, or were able to determine the username based on the default login database. Figure 23 shows the nine instances of the username "richard" being used. We looked at each IP address and its previous login attempts, and determined that all of the users gained access by some sort of dictionary attack. These IP addresses used different combinations of usernames and passwords before ultimately being successful.

In these cases, we were interested to see whether users were able to fingerprint our honeypot, but our analysis showed they were not. The address 108.170.41.186 had two successful logins using the username "richard". There was roughly a one-hour time difference between the two connections. Our logs showed that this address lost its connection and reran a dictionary attack to regain access to our system since it apparently did not record its previous successful attempt.

Timestamp	Message	Src Ip
2017-05-03T12:36:2..	login attempt [richard/richard] succeeded	207.182.157.221
2017-05-16T11:14:4..	login attempt [richard/richard] succeeded	108.170.41.186
2017-05-16T12:11:0..	login attempt [richard/richard] succeeded	108.170.41.186
2017-05-28T21:19:2..	login attempt [richard/richard] succeeded	169.56.81.30
2017-06-09T15:30:2..	login attempt [richard/richard] succeeded	149.56.118.82
2017-06-15T21:38:5..	login attempt [richard/richard] succeeded	82.135.136.227
2017-06-16T08:39:1..	login attempt [richard/richard] succeeded	103.195.90.122
2017-06-21T05:55:3..	login attempt [richard/richard] succeeded	195.235.172.89
2017-06-26T14:16:0..	login attempt [richard/richard123] succeeded	212.156.72.102

Figure 23. Richard login attempts

C. TCP/IP REQUESTS DURING THE FIRST THREE PHASES

Figure 24 shows the number of TCP/IP requests sent by users on the honeypot during the first three phases. The number of requests per date was similar during the first and third phases, when more users had access to the system. At the highest point, we recorded over 35,000 TCP/IP requests on June 12.

We did not observe any TCP/IP requests during phase 2. We speculate that these requests attempted to communicate back home with a bot over IRC (Internet Relay Chat). To test this theory, we could alter the host file on our honeypot system and define some of the addresses to be directed to our localhost, and then analyze how the bot was communicating.

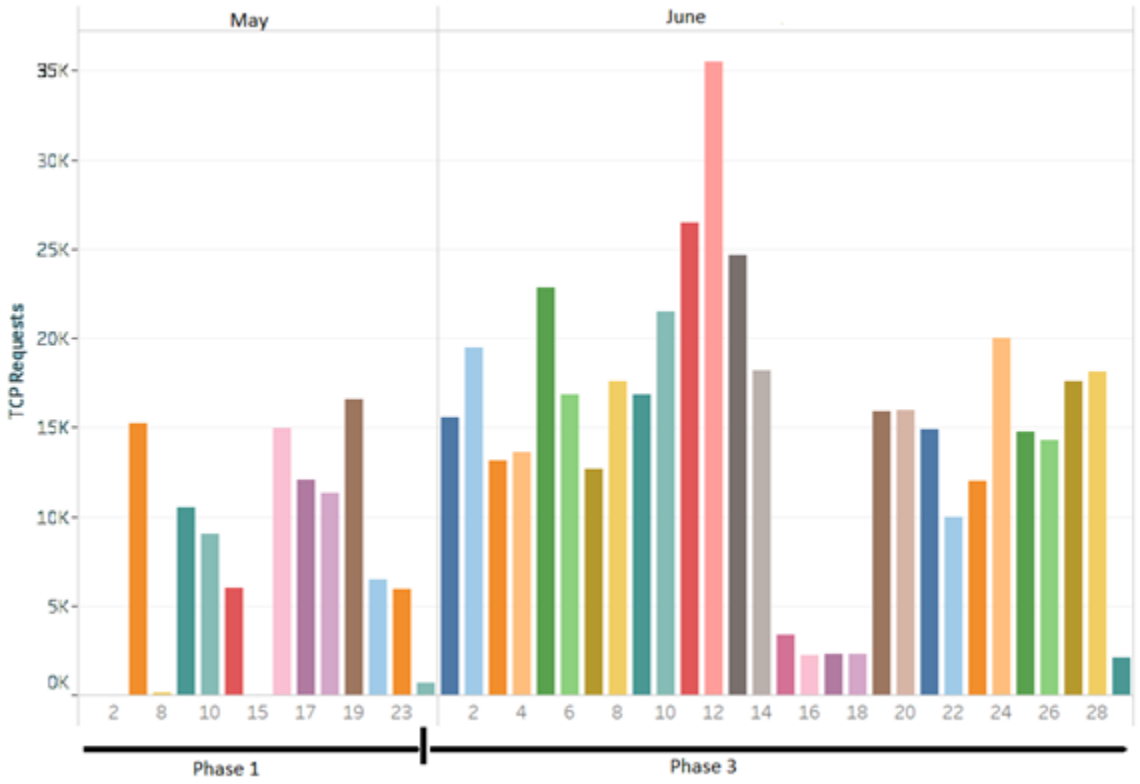

Figure 24. Number of TCP/IP requests by date

The top 10 IP addresses that users on our honeypot attempted to communicate with are shown in Figure 25. The address 162.115.18.200 had the greatest number of connection requests with 235,291. Among these top ten, we observed the outgoing port to be 5556 in all instances. This is a TCP port known to be used for malware communications. Port 443 was the incoming port for communications nine of 10 times. The attackers liked this trusted and reliable port because it is not usually blocked by firewalls, and would not appear suspicious to a user on the machine. The other port that was used significantly to communicate with the infected machine was 8888, a port used for TCP communications.

cowrie.direct-tcpip.request	direct-tcp connection request to 162.115.18.200:443 from localhost:5556	235,291
	direct-tcp connection request to 72.247.85.169:443 from localhost:5556	48,745
	direct-tcp connection request to 169.47.25.79:443 from localhost:5556	26,071
	direct-tcp connection request to 134.119.192.227:8888 from localhost:5556	18,871
	direct-tcp connection request to 137.188.82.200:443 from localhost:5556	15,353
	direct-tcp connection request to 149.126.72.187:443 from localhost:5556	11,795
	direct-tcp connection request to 192.230.96.187:443 from localhost:5556	11,511
	direct-tcp connection request to 104.80.105.89:443 from localhost:5556	9,000
	direct-tcp connection request to 162.115.210.200:443 from localhost:5556	7,720
	direct-tcp connection request to 104.80.100.232:443 from localhost:5556	6,702

Figure 25. Top 10 TCP/IP requests

D. PHASE 4 VS PHASES 1–3

During Phase 4, we observed 520,603 events that confirmed our results from the first three phases.

1. File Downloads and Commands

During Phase 4, IP addresses originating from The Netherlands performed the most file downloads, as was the case during Phases 1-3. Figure 26 shows the number of downloads by country. We did not observe any addresses during Phase 4 that were in the top 10 during Phases 1-3. The address 109.236.91.85 from the Netherlands accounted for 26 file downloads.

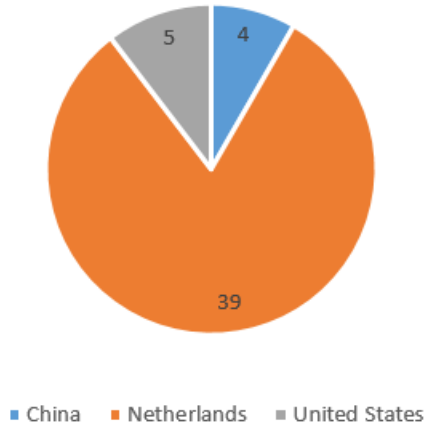

Figure 26. File downloads by country from Phase 4

During Phase 4, attackers had similar interactions with the honeypot as they did during Phases 1 through 3, despite the gap in time. Attackers provided inputs to gain information about the system, removed logs, and traversed directories. Figure 27 shows the top 10 IP addresses that interacted with the honeypot by providing inputs during this phase. We observed that an address from Poland, 195.22.127.83, provided the honeypot with the most commands in both Phase 4, and Phases 1 through 3.

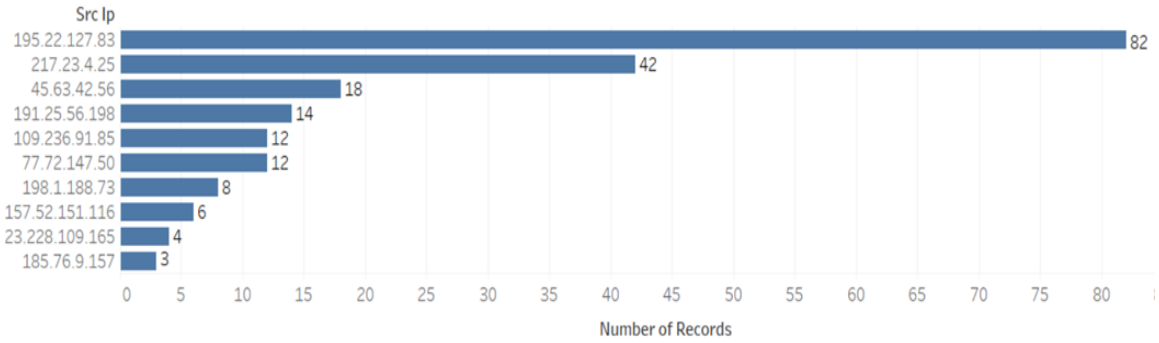

Figure 27. Top 10 addresses that provide inputs from Phase 4

2. Login Attempts

We recorded 916 successful login attempts and 253,021 unsuccessful login attempts during this phase. The top three usernames during this phase were “roots” “admin,” and “support,” among 1,944 usernames attempted. Figure 28 displays the top 10 usernames used during Phase 4. We observed 53,672 passwords used. Figure 29 displays the top 10 passwords used.

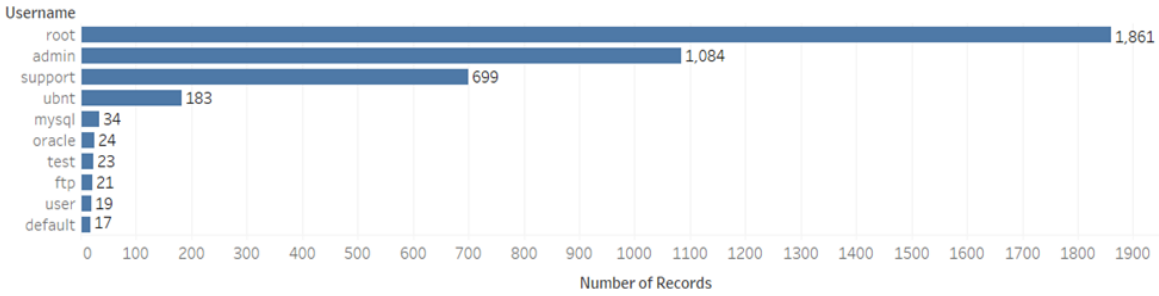

Figure 28. Top 10 usernames from Phase 4

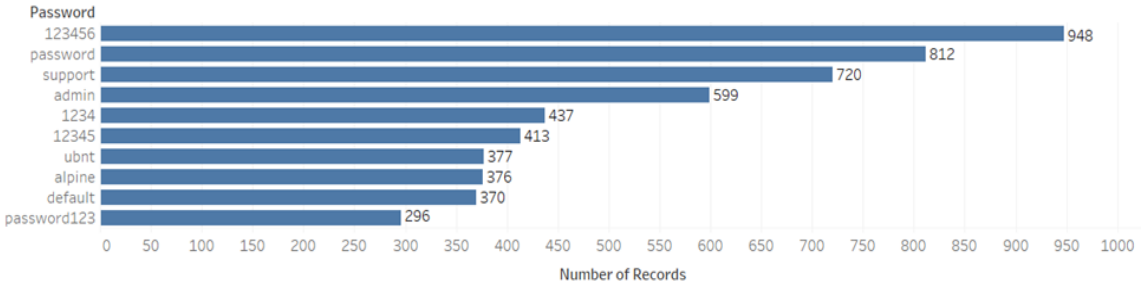

Figure 29. Top 10 passwords from Phase 4

Based on what we observed during Phases 1 through 3, we created a rule during this phase to allow password “welc0me” to be used with usernames “root” and “admin”, however, we did not observe this password in the top 10. This is likely because all instances of “welc0me” were used with the username “root”. We assume this login credential has become a common combination in dictionary attacks. All successful logins attempts are displayed in Figure 30.

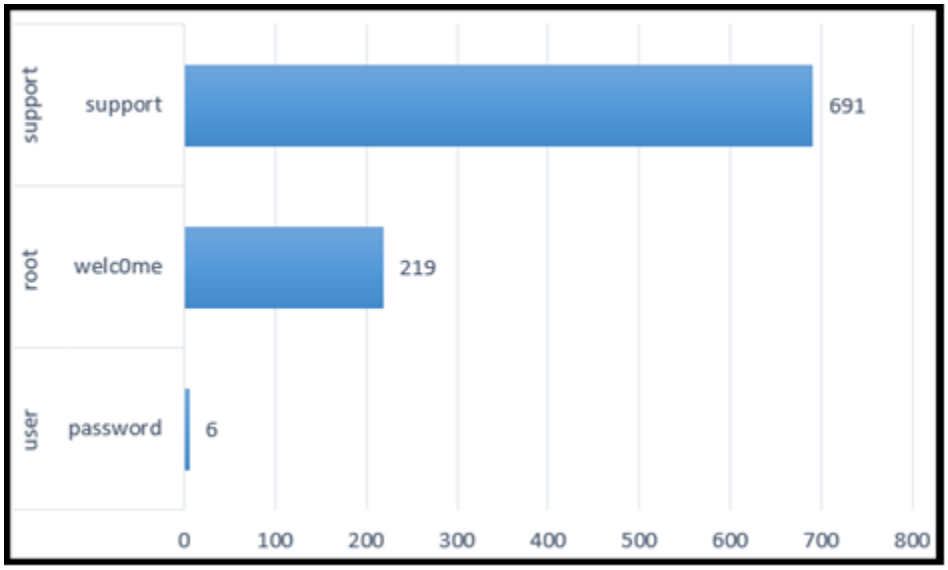

Figure 30. Successful login attempts with username and password

The number of unique addresses attempting to login was similar to those recorded during Phases 1 through 3. Figure 31 displays the number of login attempts from countries with unique IP addresses. Germany and Iran appeared in this phase but did not appear in the top 10 in the earlier phases.

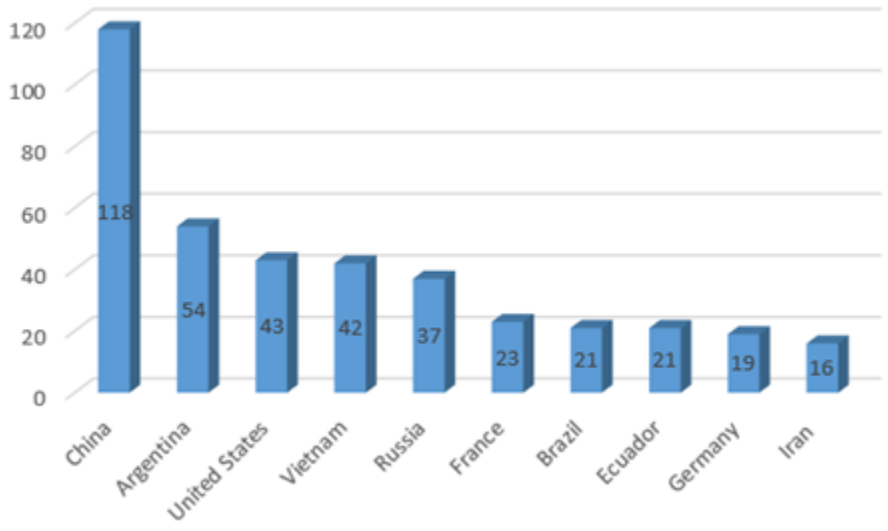

Figure 31. Top 10 countries with unique IP addresses

We also recorded the number of login attempts by date, shown in Figure 32. The most occurrences fell on weekends. On August 6, 12, and 13 (weekend days), we recorded a total of 130,893 attempts, which accounted for roughly 52 percent of all login attempts.

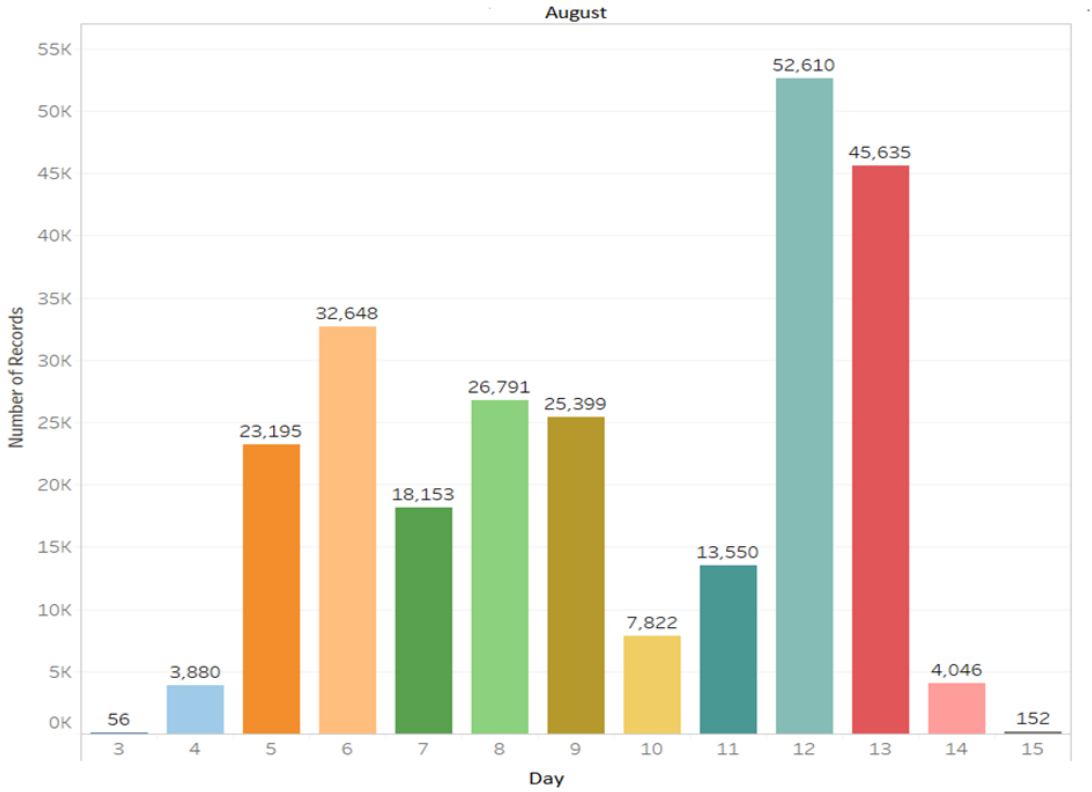

Figure 32. Number of login attempts by date

3. TCP/IP Requests

We analyzed TCP/IP requests and the addresses they were attempting to connect to. During both Phase 4 and Phases 1 through 3, address 162.115.18.200 accounted for the most connection requests. We also observed 137.188.82.200 in both groups. The top 10 connection requests during the first three phases are displayed in Figure 33, and the top 10 connection requests during Phase 4 are displayed in Figure 34. During Phase 4, we noticed that a subnetwork located in the United States had four IP addresses in the top 10 for outbound connection requests.

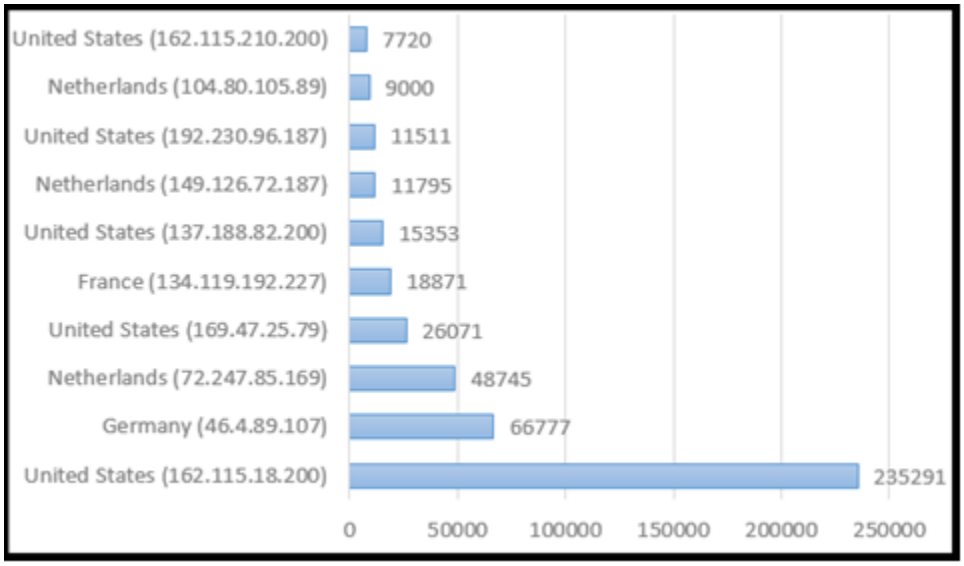

Figure 33. Top 10 connection requests from Phases 1-3

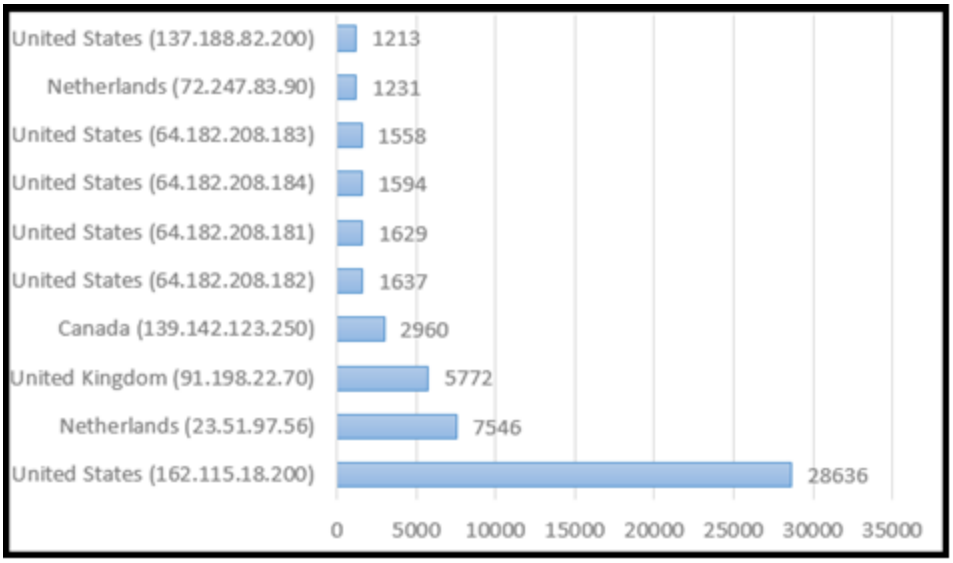

Figure 34. Top 10 connection requests from Phase 4

E. FINGERPRINTING

During the 4 phases, we did not record any ping commands, which allows users to fingerprint our honeypot before logging in. We were also curious to see whether users could gain access using the username "richard" with a valid password, then logout and reattempt login with the username "richard" and a different password. We did not observe this during the test, and no users were able to fingerprint our honeypot using this method.

On June 4, we did observe one user conducting a Cowrie fingerprinting experiment. This user's interaction with honeypot is displayed in Figure 35. Before the user interacted with the honeypot and closed the session, he wrote a message asking us to ignore this session. The three commands used afterward were *uname –a*, *ls –d*, and *ulimit –help*. Two of these commands gave the user a response, and one command was not supported by the honeypot.

Timestamp	Message
2017-06-04T16:23:49..	login attempt [root/!@] succeeded
2017-06-04T16:23:50..	CMD: Cowrie fingerprinting experiment, please ignore this session/connection
2017-06-04T16:23:50..	Command not found: Cowrie fingerprinting experiment , please ignore this session/connection
2017-06-04T16:23:50..	CMD: uname -a
2017-06-04T16:23:50..	Command found: uname -a
2017-06-04T16:23:51..	CMD: ls -d
2017-06-04T16:23:51..	Command found: ls -d
2017-06-04T16:23:51..	CMD: ulimit -help
2017-06-04T16:23:51..	Command not found: ulimit -help
2017-06-04T16:23:51..	CMD: Cowrie fingerprinting experiment, please ignore this session/connection
2017-06-04T16:23:51..	Command not found: Cowrie fingerprinting experiment , please ignore this session/connection

Figure 35. Cowrie fingerprinting experiment

We were not confident that this user was able to fingerprint our honeypot using these commands. The user more than likely got the results from the commands and compared them with other servers connect to the Internet.

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSIONS

This thesis tested a honeypot tool that took earlier fingerprinting methods into consideration. This was Cowrie, an SSH honeypot. We ran four phases that had different parameters for the login credential database. We also varied the times which the honeypot was running.

The honeypot ran on an ISP (Internet Service Provider) network for two and a half months and logged over two million events. We focused on login attempts, interactions attackers had with the honeypot, file downloads, and TCP/IP requests sent by attackers. We concluded that different countries performed different activities with honeypot. China had the most login attempts, whereas the Netherlands preformed the most file downloads. The file downloads were from wget requests and were mainly .sh scripts. Poland interacted with the honeypot the most, as they provided the most commands.

We left the account name of "richard" in the password file during Phase 1. During Phase 2, we altered the login database to only accept the username "richard". We only recorded one login during this phase from a persistent user who failed to login previously. This confirmed that users did not use the login credentials to fingerprint the honeypot during this phase.

We analyzed the effectiveness of different fingerprinting methods and incorporated some of these methods when we deployed the honeypot. Changes made between the experimental phases also added a degree of deception to the honeypot. Once we observed an attacker conducting their own fingerprinting experiment on our honeypot.

By creating the login database for Phase 4 based on the results from Phases 1-3, we could confirm that data such as file downloads by country, login credentials used, and login attempts from unique address were similar. We also could discover interesting datasets in the TCP/IP requests and gather more results.

B. FUTURE WORK

Future work could analyze the scripts downloaded by attackers to the honeypot. These scripts can be run in a virtual environment to dynamically analyze them. If any interesting behavior is observed, analyzing the binary code may be useful.

We observed an attacker who advertised it was conducting a Cowrie fingerprinting experiment. Future work could search for other methods to fingerprint the honeypot and apply the fixes accordingly. Afterward, running the updated honeypot may provide different results.

Other future work could compare results when the honeypot is deployed with the default file system and when it is deployed with a different file system. User interactions may change when they are presented with different files to inspect.

APPENDIX A. INSTALLING COWRIE

Installing Dependencies

$ sudo apt-get install git python-virtualenv libmpfr-dev libssl-dev libmpc-dev libffi-dev build-essential libpython-dev python2.7-minimal authbind

User Account

$ sudo adduser --disabled-password cowrie

$ sudo su – cowrie

Cowire Code

$ git clone http://github.com/micheloosterhof/cowrie

Virtual Environment

$ pwd

$ virtualenv cowrie-env

$ source cowrie-env/bin/activate

Starting Cowrie

$ bin/cowrie start

Redirection to port 2222

$ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. IP ADDRESSES TO COUNTRY NAMES

```
#Ryan McCaughey
#Takes IP addresses and writes the corresponding country to a new text file
import maxminddb
reader = maxminddb.open_database('GeoLite2-Country.mmdb')
InputFile=open('IPAddress.txt', 'r')          #File that is to be opened. Holds IPs
OutputFile =open('IPCountries.txt', 'w') #File that will have the counties written to
for data in InputFile:
        dataTest =(data.strip())
        output = (reader.get(dataTest))
        if output:
                newoutput = output['country']
                names = newoutput['names']
                country = names['en']
                #print(country)
                OutputFile.write(country + '\n')
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. ADDITIONAL TABLES AND FIGURES

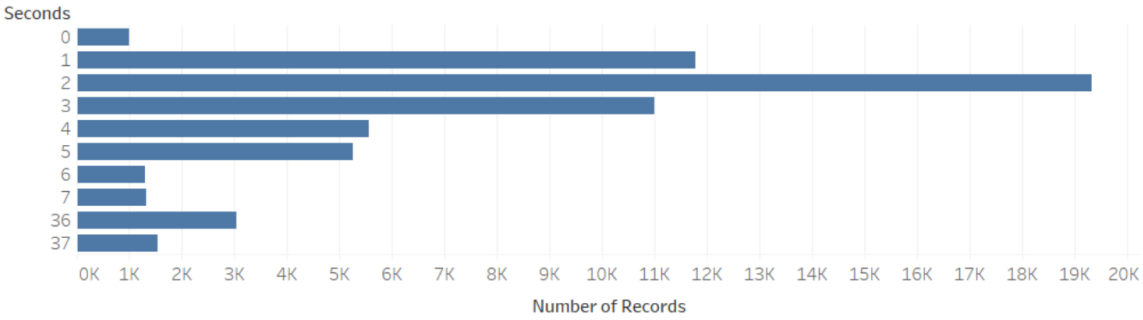

Figure 36. Top 10 times users remained on the honeypot

Table 11. Top 10 commands used

Command	**Number of records**
cd /tmp	827
cd /var/run	715
cd /	711
cd /mnt	711
cd /root	711
wget http://107.174.34.70/Swag.sh	338
cat /bin/ls	330
echo –en "//x31//x33//x33//x37"	330
free –m	203
uname	197

Timestamp	Src Ip	Message
2017-05-11T10:11:49.02..	212.156.72.102	New connection: 212.156.72.102:3081 (207.140.106.53:2222) [..
2017-05-11T10:11:49.27..	212.156.72.102	Remote SSH version: SSH-2.0-libssh2_1.7.0
2017-05-11T10:11:50.96..	212.156.72.102	login attempt [rpc/rpc] failed
2017-05-11T10:11:52.22..	212.156.72.102	login attempt [rpc/rpc] failed
2017-05-11T10:11:53.50..	212.156.72.102	login attempt [rpc/rpc] failed
2017-05-11T10:11:54.75..	212.156.72.102	Connection lost after 5 seconds
2017-05-11T10:25:52.92..	212.156.72.102	New connection: 212.156.72.102:3081 (207.140.106.53:2222) [..
2017-05-11T10:25:53.17..	212.156.72.102	Remote SSH version: SSH-2.0-libssh2_1.7.0
2017-05-11T10:25:54.89..	212.156.72.102	login attempt [administrador/administrador] failed
2017-05-11T10:25:56.15..	212.156.72.102	login attempt [administrador/administrador] failed
2017-05-11T10:25:57.41..	212.156.72.102	login attempt [administrador/administrador] failed
2017-05-11T10:25:58.67..	212.156.72.102	Connection lost after 5 seconds
2017-05-11T10:39:43.03..	212.156.72.102	New connection: 212.156.72.102:3081 (207.140.106.53:2222) [..
2017-05-11T10:39:43.28..	212.156.72.102	Remote SSH version: SSH-2.0-libssh2_1.7.0
2017-05-11T10:39:44.98..	212.156.72.102	login attempt [control/control123] failed
2017-05-11T10:39:46.26..	212.156.72.102	login attempt [control/control123] failed
2017-05-11T10:39:47.52..	212.156.72.102	login attempt [control/control123] failed
2017-05-11T10:39:48.77..	212.156.72.102	Connection lost after 5 seconds
2017-06-26T14:16:07.31..	212.156.72.102	New connection: 212.156.72.102:12041 (207.140.106.53:2222) ..
2017-06-26T14:16:07.55..	212.156.72.102	Remote SSH version: SSH-2.0-libssh2_1.7.0
2017-06-26T14:16:09.19..	212.156.72.102	login attempt [richard/richard123] succeeded
2017-06-26T14:16:12.71..	212.156.72.102	Connection lost after 5 seconds

Figure 37. Login attempts from 212.165.72.102

Timestamp	Src Ip	Message
2017-06-26T14:16:10.27..	212.156.72.102	CMD: unset HISTORY HISTFILE HISTSAVE HISTZO..
2017-06-26T14:16:10.28..	212.156.72.102	Command found: unset HISTORY HISTFILE HISTS..
2017-06-26T14:16:10.28..	212.156.72.102	Command found: history -n
2017-06-26T14:16:10.29..	212.156.72.102	Command found: export HISTFILE=/dev/null
2017-06-26T14:16:10.29..	212.156.72.102	Command found: export HISTSIZE=0
2017-06-26T14:16:10.30..	212.156.72.102	Command found: export HISTFILESIZE=0
2017-06-26T14:16:10.30..	212.156.72.102	Command found: rm -rf /var/log/wtmp
2017-06-26T14:16:10.31..	212.156.72.102	Command found: rm -rf /var/log/lastlog
2017-06-26T14:16:10.31..	212.156.72.102	Command found: rm -rf /var/log/secure
2017-06-26T14:16:10.32..	212.156.72.102	Command found: rm -rf /var/log/xferlog
2017-06-26T14:16:10.32..	212.156.72.102	Command found: rm -rf /var/log/messages
2017-06-26T14:16:10.33..	212.156.72.102	Command found: rm -rf /var/run/utmp
2017-06-26T14:16:10.33..	212.156.72.102	Command found: touch /var/run/utmp
2017-06-26T14:16:10.34..	212.156.72.102	Command found: touch /var/log/messages
2017-06-26T14:16:10.35..	212.156.72.102	Command found: touch /var/log/wtmp
2017-06-26T14:16:10.35..	212.156.72.102	Command found: touch /var/log/messages
2017-06-26T14:16:10.36..	212.156.72.102	Command found: touch /var/log/xferlog
2017-06-26T14:16:10.36..	212.156.72.102	Command found: touch /var/log/secure
2017-06-26T14:16:10.37..	212.156.72.102	Command found: touch /var/log/lastlog
2017-06-26T14:16:10.37..	212.156.72.102	Command found: rm -rf /var/log/maillog
2017-06-26T14:16:10.38..	212.156.72.102	Command found: touch /var/log/maillog
2017-06-26T14:16:10.38..	212.156.72.102	Command found: rm -rf /root/.bash_history
2017-06-26T14:16:10.39..	212.156.72.102	Command found: touch /root/.bash_history
2017-06-26T14:16:10.39..	212.156.72.102	Command found: history -r
2017-06-26T14:16:10.86..	212.156.72.102	CMD: uname
2017-06-26T14:16:10.87..	212.156.72.102	Command found: uname
2017-06-26T14:16:11.34..	212.156.72.102	CMD: free -m
2017-06-26T14:16:11.35..	212.156.72.102	Command found: free -m
2017-06-26T14:16:11.83..	212.156.72.102	CMD: ps -x
2017-06-26T14:16:11.83..	212.156.72.102	Command found: ps -x
2017-06-26T14:16:12.30..	212.156.72.102	CMD: cat /proc/cpuinfo
2017-06-26T14:16:12.31..	212.156.72.102	Command found: cat /proc/cpuinfo

Figure 38. Activity from 212.156.72.102

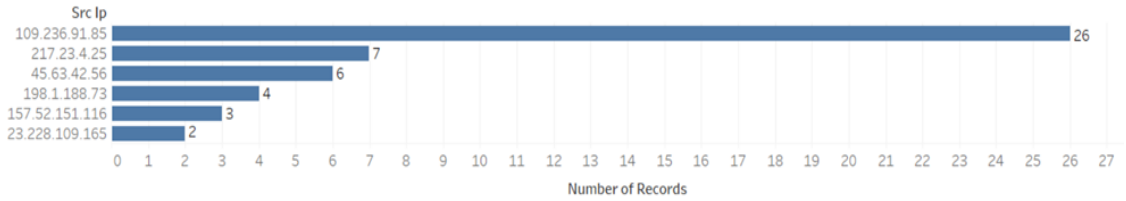

Figure 39. All file downloads by IP during Phase 4

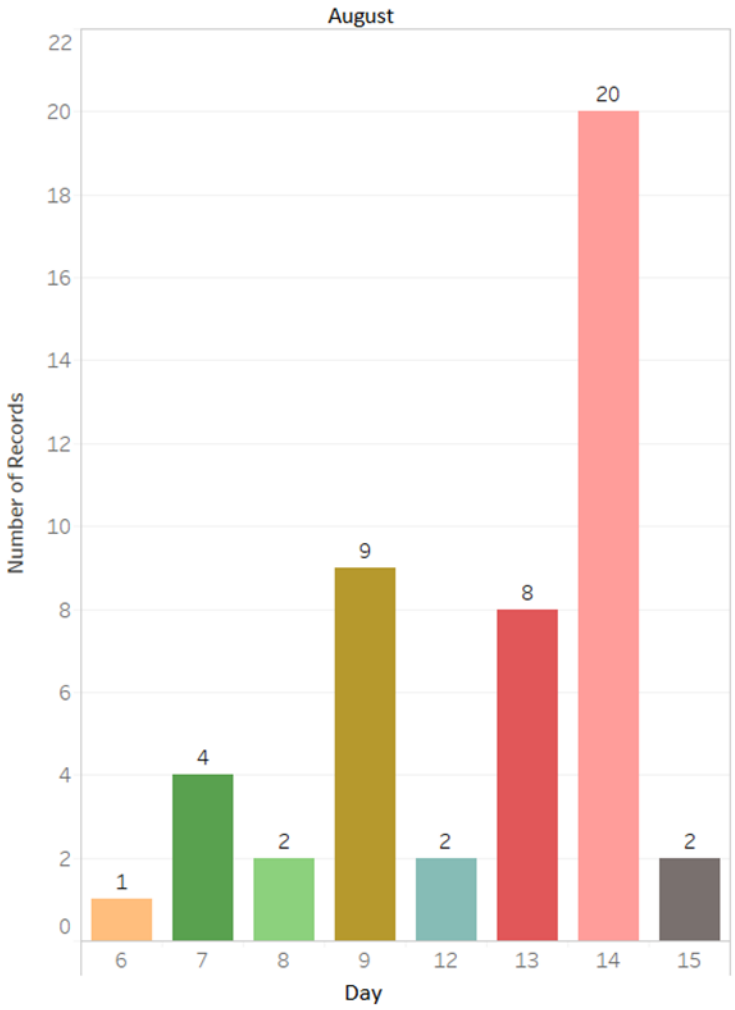

Figure 40. Command inputs by date during Phase 4

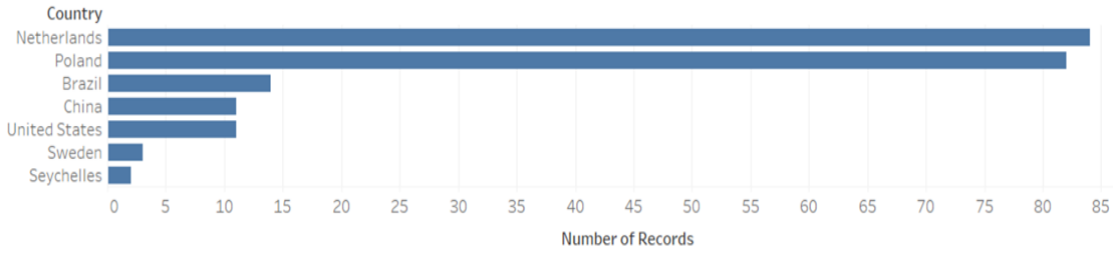

Figure 41. Command inputs by country during Phase 4

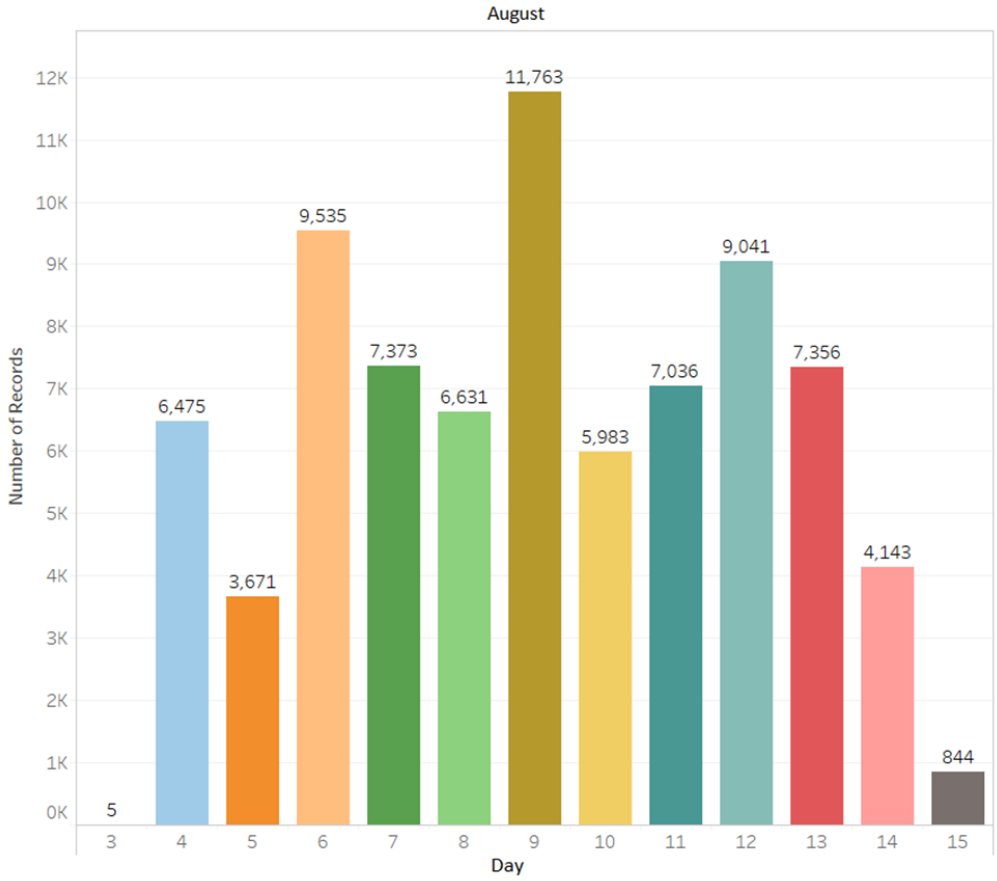

Figure 42. Top 10 connection requests from Phase 4

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Internet of things research study. (Nov. 2015). Hewlett Packard Enterprise. Palo Alto, CA. [Online]. Available: http://h20195.www2.hpe.com/V4/getpdf.aspx/4aa5-4759enw

[2] Q1 2017 state of the Internet / security report. (May 2017). Akamai Technologies. Cambridge, MA. [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-Internet/q1-2017-state-of-the-Internet-security-report.pdf

[3] E. Caltum and O. Segal. (Oct. 2016). SSHowDowN: Exploitation of IoT devices for launching mass-scale attack campaigns. Akamai Technologies. Cambridge, MA [Online]. Available: https://www.akamai.com/us/en/multimedia/documents/state-of-the-Internet/sshowdown-exploitation-of-iot-devices-for-launching-mass-scale-attack-campaigns.pdf

[4] L. Even. (2000, Jul. 12). What is a honeypot? SANS. [Online]. Available: https://www.sans.org/security-resources/idfaq/what-is-a-honeypot/1/9

[5] SSH Communications Security. (2017). SSH protocol. [Online]. Available: https://www.ssh.com/ssh/protocol/

[6] J. Riden and C. Seifert. (2008, Feb. 14). A guide to different kinds of honeypots. [Online]. Available: https://www.symantec.com/connect/articles/guide-different-kinds-honeypots

[7] L. Spitzner. (2013, Jan. 23). Honeypots—Definitions and value of honeypots. Windows Security [Online]. Available: http://www.windowsecurity.com/whitepapers/honeypots/Honeypots_Definitions_and_Value_of_Honeypots.html

[8] LM Security. (2016. Feb. 11). How malware detects virtualized environment (and its countermeasures).. [Online]. Available: http://resources.infosecinstitute.com/how-malware-detects-virtualized-environment-and-its-countermeasures-an-overview/#gref

[9] G. Kelly and D. Gan, "Analysis of attacks using a honeypot," in *Proceedings of Cyberforensics*, 2014, pp. 65–72.

[10] D. Watson. (2015, Aug. 06). Low interaction honeypots revisited. The Honeynet Project. [Online]. Available: https://www.honeynet.org/node/1267

[11] A. Yahyaoui, "Testing deceptive honeypots," M.S thesis, Dept. Comp. Sci., Naval Postgraduate School, Monterey, CA, 2014

[12] M. Oosterhof. Cowrie honeypot. [Online]. Available: http://www.micheloosterhof.com/cowrie/

[13] Honeynet Project. (2016, May 31). Know your enemy: Honeynets. [Online]. Available: http://old.honeynet.org/papers/honeynet/index.html

[14] R.C. Joshi and A. Sardana, "Honeypots," in *Honeypots: A New Paradigm to Information Security*, R.C. Joshi and A. Sardana, Eds. Boca Raton, FL: Science Publishers, pp. 1–37.

[15] E. Tan. (2014, Feb. 13). Dionaea—A malware capturing honeypot. [Online]. Available: https://www.edgis-security.org/honeypot/dionaea/

[16] R. N. Dahbul, C. Lim and J. Purnama, "Enhancing honeypot deception capability through network service fingerprinting," in *International Conference on Computing and Applied Informatics*, 2016, vol. 801, no. 1.

[17] A. Morris. (2014, Dec. 12). Detecting Kippo SSH honeypots, bypassing patches, and all that jazz. [Online]. Available: http://morris.sc/detecting-kippo-ssh-honeypots/

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

[19] M. Foster. (2013, Apr. 15). Running a Kippo honeypot: Part one. [Online]. Available: https://hackerific.net/2013/04/15/running-a-kippo-honeypot/

[20] M. Oosterhof. (n.d.). Cowrie SSH/telnet honeypot. [Online]. Available: https://github.com/micheloosterhof/cowrie

[21] M. Oosterhof. (n.d.). Forwarding.py [Online]. Available: https://github.com/micheloosterhof/cowrie/blob/master/cowrie/ssh/forwarding.py

[22] Tableau. (n.d.). Business intelligence and analytics. [Online] Available: https://www.tableau.com/

[23] MaxMind. (n.d). Python MaxMind DB reader extension. [Online] Available: https://github.com/maxmind/MaxMind-DB-Reader-python

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California