國立臺灣大學電機資訊學院電機工程學研究所

碩士論文

Department of Electrical Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

現代物聯網僵屍網路行為的深度分析：利用多個蜜罐進行研究

Deep Analysis of Modern IoT Botnet Behavior with Multiple Honeypots.

王棠葳

Tang-Wei Wang

指導教授: 雷欽隆 博士

Advisor: Chin-Laung Lei, Ph.D.

中華民國 112 年 8 月

August, 2023

doi:10.6342/NTU202303073

摘要

隨著物聯網（IoT）設備數量的急速增加，它們已成為網路攻擊的目標，例如 botnet 中的「Mirai」和「Gafgyt」。這些攻擊通常利用物聯網設備的漏洞，並透過 SSH 或 Telnet 傳播，這是許多路由器和 IP 攝像頭等設備常用的通訊協定。為了防範這類攻擊，蜜罐（honeypot）被證明是一個有用的工具。它通過模擬能被攻擊的平台、機器或服務，作為誘餌來吸引攻擊者。蜜罐不僅能夠誤導攻擊者，使其轉移對真實目標的注意力，而且還使防禦者能夠更深入地分析攻擊者使用的策略和技術。在這個實驗中，我們同時部署了四個不同配置的 SSH/Telnet 蜜罐，使我們能夠更深入地了解攻擊者的行為。結果，我們在實驗中識別出至少 54 種不同類型的攻擊，並依據其出現頻率從中挑選了 14 種攻擊，進行更深入的分析。我們指出了一些有用的特徵，並整理了一份簡單的指南，以幫助人們抵禦當前的 botnet 威脅。

關鍵字：蜜罐、反蜜罐、SSH、Telnet、物聯網、殭屍網路、Cowrie

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

Abstract

With the rapid increase in the number of IoT devices, they have become targets for cyber attacks, such as the botnets ”Mirai” and ”Gafgyt.” These attacks often exploit vulnerabilities in IoT devices, particularly through SSH or telnet connections commonly used by devices like routers and IP cameras. To defend against such attacks, a honeypot proves to be a valuable tool. It operates by simulating a vulnerable platform, machine, or service, serving as a decoy to lure attackers. Not only does the honeypot mislead attackers by diverting their attention from the real target, but it also enables defenders to analyze the strategies and techniques employed by the attackers in greater depth. In our case, we deployed four SSH/telnet honeypots with different configurations simultaneously, allowing us to gain deeper insights into the behavior of attackers. As a result, we identified at least 54 distinct types of attacks that occurred during our experiment and picked 14 of them based on appearance frequency to perform deeper analysis. We pointed out some useful features and arranged a simple guideline to help people defend against the current botnet

doi:10.6342/NTU202303073

threats.

Keywords: Honeypot, Anti-honeypot, SSH, Telnet, IoT, Botnet, Cowrie

doi:10.6342/NTU202303073

Contents

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

List of Figures

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

List of Tables

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

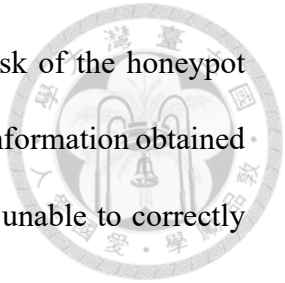
Chapter 1 Introduction

In 2016, a new type of botnet called “Mirai” emerged. This botnet was designed to target the large number of IoT (Internet of Things) devices. The original version of Mirai took advantage of a weakness where manufacturers often set up a default username and password for their devices, which consumers should change as soon as they set up their devices. For example, the older version of Raspberry Pi OS had a default username and password of “pi/raspberrypi”. Mirai used a list of default usernames and passwords to perform dictionary attack and gain access to many devices through SSH connection. Once a device was infected, the botmasters could use it to launch denial-of-service attack. Mirai was responsible for one of the biggest DDoS attacks in September 2016. Honeypot is useful for detecting and capturing botnets like Mirai. IOTPOT, developed by [18], captured 106 malware samples in 42 days.

A honeypot is a tool that is designed to mimic a vulnerable device in order to attract attackers. There are various types of honeypots that target different protocols and services. For instance, there are HTTP honeypots and SQL honeypots. Honeypots can be classified based on the degree of interaction they provide.

- Low Interaction Honeypot(LIH): This type of honeypot limits access to the system for attackers. Typically, it simulates commonly used features or operations. Because

doi:10.6342/NTU202303073



attackers cannot directly communicate with the system, the risk of the honeypot being compromised and used in an attack is low. However, the information obtained from this type of honeypot may be limited since attackers are unable to correctly execute their exploits.

- High Interaction Honeypot(HIH): This type of honeypot provides attackers with almost complete access to the system. As a result, security researchers need to carefully configure the network connection of the honeypot to prevent other machines on the internet from being endangered. In theory, a High Interaction Honeypot (HIH) has the ability to capture zero-day attacks.

Some people consider a third class of honeypot called a ”Medium Interaction Honeypot (MIH)”. It offers more simulated functions than a LIH, but it is not an actual operating system. There is no universally agreed upon definition for LIH or MIH, but the idea is straightforward.

In this study, we will concentrate on SSH/Telnet honeypots because numerous botnets, including Mirai and Gafgyt, spread through the SSH/telnet protocol. The process by which a botnet infects a machine can be broadly divided into three steps, as described in [18]: intrusion, infection, and monetization.

1. Intrusion: During this stage, attackers attempt to gain control of the system. They may exploit vulnerabilities in the system or employ brute-force techniques to guess passwords and gain unauthorized access.

2. Infection: Once attackers have successfully infiltrated the system, they proceed to check and configure the environment to suit their objectives. At this stage, attack-

doi:10.6342/NTU202303073

ers often download scripts or binary files and execute them, preparing for the next phase.

3. Monetization: In the final stage, the downloaded malware becomes active and carries out malicious activities. This may include activities such as launching distributed denial-of-service (DDoS) attacks or mining cryptocurrencies.

There are various tools available to set up SSH/telnet honeypots. ssh-honeypot[12] is designed to capture the usernames and passwords used by attackers during the intrusion phase. Another popular tool, Cowrie[17], can log the commands made by the attacker after logging in and capture some malware samples in the infection phase. Additionally, when combined with a sandbox, a honeypot can analyze the behavior of malware in the monetization phase. With higher levels of customization, a honeypot can potentially locate the C&C server of botnets or even become a spy C&C server.

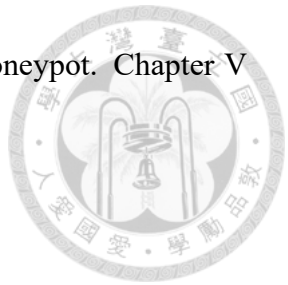
On the other hand, it is actually possible for attackers to identify the presence of a honeypot. Numerous studies have proposed various anti-honeypot methods to counteract these defensive measures. While it is true that most open-source honeypot tools can eventually be detected, there is still value in deploying such honeypots, even if they are discovered after fulfilling their intended purpose.

In our study, we deployed 4 honeypots simultaneously, each with a different level of disguise or camouflage, in order to observe attackers' behavior in different environments. This allowed us to gather valuable insights into how attackers adapt and respond to varying honeypot configurations.

The remainder of the paper is organized as follows: Chapter II discusses some related work on honeypots and honeypot detection techniques. Chapter III outlines the details of

doi:10.6342/NTU202303073

our experiment. Chapter IV analyzes the attacks observed by our honeypot. Chapter V draws conclusions, and chapter VI describes future work.



doi:10.6342/NTU202303073

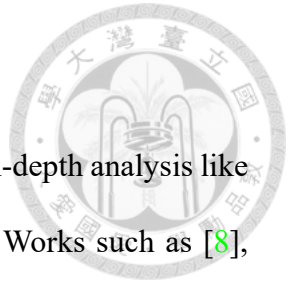
Chapter 2 Related work

[24] introduced a hybrid honeypot named IoTCMAL that enables attackers to launch attacks not only through SSH and Telnet connections but also through exploiting actual vulnerabilities in IoT devices. They collected 1435 malware samples between March 1, 2019, and August 31, 2019. Their primary focus was to develop a method for mapping the traffic of vulnerable IoT services to the public network.

[18] proposed a honeypot that is combined with a sandbox, allowing them to capture and analyze malware at the same time. Their honeypot supports multiple CPU architectures, such as x86, ARM, and MIPS. They deployed their system on 87 different IP addresses for 39 days in 2015 and analyzed the malicious behavior captured by the honeypot. They categorized botnets based on their login credentials, command sequences, and downloaded files. Their work provides a good opportunity to learn basic knowledge about SSH/Telnet botnet ecology.

[9] developed a script to automatically and randomly configure a Cowrie honeypot. They deployed three honeypots, one default and two configured. They found that attackers tend to stay longer and send more commands in the configured honeypots. However, they didn't conduct a deep analysis of the data they captured. Furthermore, their honeypots were deployed at different time periods. In our work, we configure the Cowrie in a similar

doi:10.6342/NTU202303073



way to theirs and deploy them simultaneously.

There are several works worth mentioning that did not provide in-depth analysis like [18] and [24], but they are still valuable contributions to the field. Works such as [8], [14], and [22] utilized Cowrie or its predecessor Kippo to capture SSH and Telnet botnet activity. These studies focused on providing statistics related to the login credentials used by attackers, the geographical locations of their IP addresses, daily attack counts, and other relevant information. Similarly, [15] employed a self-made SSH honeypot and conducted a similar analysis. The main distinction between the two previously mentioned works and these studies is that the former ones focused on classifying the attackers and conducting cross-analysis of the introduced features.

In our research, we conducted a thorough analysis comparable to the works presented in [24] and [18]. While we did not employ a sandbox environment for dynamic analysis of downloaded files, we identified and highlighted some unique features about attackers that had not been previously introduced, such as the SSH client identification string and parallel attacks.

Additionally, in contrast to other studies that deployed multiple honeypots and listened on multiple IP addresses primarily to capture more data, our objective was to perform cross-analysis of attackers' behavior in different environments. This approach allowed us to gain insights into how attackers adapt and respond to varying honeypot configurations, shedding light on their strategies and techniques.

By focusing on these distinct aspects, we contribute to the existing body of knowledge on honeypot-based research and provide a comprehensive understanding of attackers' behavior in different scenarios.

doi:10.6342/NTU202303073

Chapter 3 Background

3.1 SSH and Telnet protocol

Before explaining the configuration and captured data of our honeypot, let's provide a brief introduction to the SSH[16] and Telnet[2] protocols. SSH and Telnet are similar protocols in that they both allow users to access a remote host's command-line interface (CLI). They transmit data through a TCP/IP connection, with SSH servers typically listening on port 22 and Telnet on port 23. In most cases, when connecting to an SSH/Telnet server, users are prompted to enter a username and password for authentication. Upon successful login, users are granted access to a CLI where they can execute commands.

The main difference between SSH and Telnet lies in the way data is transferred. Telnet transfers data in plaintext, while SSH connections are encrypted. Consequently, during the initial stages of the SSH protocol, several options need to be exchanged between the client and server. These include the public-key cryptosystem, key exchange method, and symmetric encryption algorithm. The SSH protocol can be broadly described by referring to table 3.1. On the other hand, Telnet is simpler, as some servers allow users to authenticate immediately after establishing the TCP connection.

doi:10.6342/NTU202303073

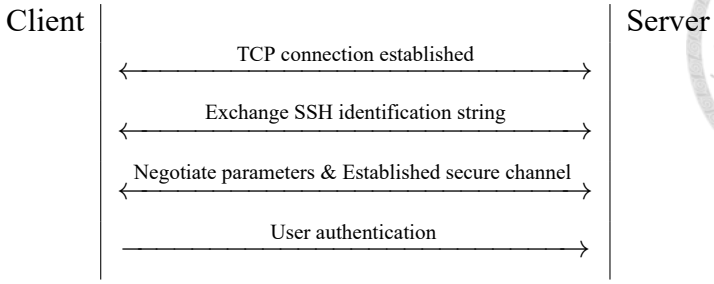


Figure 3.1: Summary of SSH protocol

3.2 Cowrie

Cowrie is a Python-based SSH/Telnet honeypot that is designed to simulate SSH/telnet services. It provides attackers who successfully log in with a Unix shell system and implements 99 commands with a complete file system, which are listed in Table 3.1. Cowrie records the usernames/passwords used by attackers and the commands they execute. When attackers attempt to download files with commands such as `wget` and `curl`, Cowrie saves the files, enabling users to perform further analysis. Additionally, Cowrie can be configured to function as an SSH/telnet proxy that connects to real or virtual machines, making it a high interaction honeypot. Cowrie is currently one of the most popular SSH honeypots available. In 2017, [23] discovered more than 3,000 Cowrie honeypots, as well as its predecessor, Kippo, by scanning IP addresses.

doi:10.6342/NTU202303073

Table 3.1: The commands supported by Cowrie

adduser	exit	halt	export	:
dd	cd	ifconfig	python	uname
apt	logout	reboot	alias	do
du	rm	iptables	scp	uniq
awk	clear	history	jobs	done
env	cp	last	service	unzip
whoami	reset	yes	kill	base64
ethtool	mv	ls	sleep	uptime
users	hostname	bash	pkill	busybox
free	mkdir	dir	ssh	wc
help	ps	sh	killall	cat
grep	rmdir	nc	sudo	wget
w	id	php	killall5	chmod
fgrep	pwd	netstat	tar	dget
who	passwd	chattr	su	chpasswd
egrep	touch	nohup	tee	which
echo	shutdown	umash	chown	crontab
tail	ftpget	perl	tftp	yum
printf	poweroff	unset	chgrp	curl
head	gcc	ping	ulimit	

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

Chapter 4 Methodology

4.1 Reconnaissance

Our work focuses on providing attackers with honeypots that have different levels of disguise. Once attackers gain access, they typically investigate the environment by executing commands. However, configuring each of the approximately hundred commands implemented by Cowrie individually can be an exhausting task. To establish a preliminary direction for configuring the honeypots, we initially set up a completely default Cowrie honeypot. The commands provided to this default Cowrie instance are potential indicators that attackers may use to detect a honeypot.

During the honeypot's operational period from March 29, 2023, to April 1, 2023, we recorded the usage of Cowrie-supported commands by different IP addresses. The results are displayed in table 4.1. Most of these commands primarily serve to perform operations rather than display information. Examples include commands like `rm`, `cp`, `mkdir`, and `cd`. In the following discussion, we will focus on the commands that attackers may utilize to gather information.

- `cat`: It can display file content.
- `cd`, `ls`: They give the information about file system.

doi:10.6342/NTU202303073

Table 4.1: Commands captured by default Cowrie

commands	counts	commands	counts
`cat`	346	`uname`	20
`cp`	163	`grep`	7
`cd`	151	`sudo`	4
`wget`	143	`ps`	4
`tftp`	139	`export`	4
`rm`	80	`scp`	3
`echo`	72	`curl`	3
`sh`	70	`unset`	2
`exit`	67	`ls`	2
`done`	63	`ifconfig`	2
`do`	63	`bash`	2
`dd`	63	`ssh`	1
`chmod`	24	`perl`	1
`uname`	20	`mkdir`	1

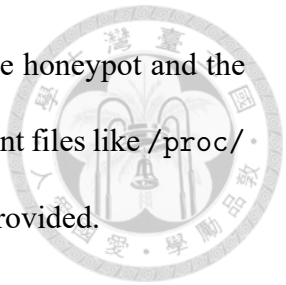
- `ps`: It show the information about processes.
- `ifconfig`: It can show the network interfaces status.
- `uname`: It print system information.

Cowrie already provides sufficient randomness for the `ifconfig` command. The output of the `ps` command, which can be used to detect Cowrie, is rarely captured by the honeypot. Therefore, we decided to focus on configuring the settings related to the `uname` command and file system to improve our honeypot's effectiveness.

4.2 Configuration

Rather than configuring the Cowrie honeypot randomly like in the previous work by [9], we chose to configure it to imitate a specific machine and developed a Python script to automate this task. The script is based on Cabral et al.'s work, but we modified it to establish SSH connection to the target machine. This enables the script to adjust

doi:10.6342/NTU202303073



the settings related to the `uname` command so that the output from the honeypot and the target machine are identical. Additionally, the script can clone important files like `/proc/cpuinfo` and `/etc/passwd`, and even copy the entire file system if provided.

We created an OpenWRT image and ran it on Qemu to serve as the target machine. We also set up four honeypots with different levels of disguise, which are described in detail in table 4.2 and 4.3. To attract as many attacks as possible, we configured the honeypots to accept any username/password combination for login. We didn't want there to be too much of difference between the target machine and the default Cowrie, as this would make it difficult to determine whether an attacker was detecting the honeypot or simply selecting the environment. Therefore, we chose OpenWrt, which is also a Linux distribution and runs on an x86_64 CPU. Additionally, because we ran virtual machines to set up a high-interaction honeypot, the lightweight nature of OpenWrt helped reduce system load.

Deploying multiple honeypots allows us to capture a wider range of attacker behaviors. More honeypots make analysis easier and provide us with a larger dataset. Honeypot B offers a simple disguise in the early stages of intrusion and infection, while Honeypot C is highly disguised and difficult to detect. Honeypot D closely resembles a real machine, making it challenging for attackers to distinguish it from a legitimate target. The combination of these honeypots provides valuable insights into attackers' behavior and helps improve our overall understanding of their strategies and techniques.

Note that we implemented a high-interaction honeypot using Cowrie's tools. However, honeypot D encountered engineering issues and didn't enable Telnet connections.

doi:10.6342/NTU202303073

Table 4.2: Configuration of honeypots

Name	SSH identification string	Login Banner	Hostname	`uname -a`	file system	Interaction
A	default	default	default (`svr04`)	default	default	medium
B	OpenWrt	OpenWrt	default (`svr04`)	default	default	medium
C	OpenWrt	OpenWrt	OpenWrt (`OpenWrt`)	OpenWrt	OpenWrt	medium
D	OpenWrt	OpenWrt	OpenWrt (`OpenWrt`)	OpenWrt	OpenWrt	high

Table 4.3: Content of Configuration

	SSH identification string
default	`SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2`
OpenWrt	`SSH-2.0-dropbear`
	Login Banner
default	The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in /usr/share/doc/*/copyright. Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.
OpenWrt	BusyBox v1.35.0 (2023-01-03 00:24:21 UTC) built-in shell (ash) _______ ________ __ | |.-----.-----.-----.| | | |.----.| |_ | - || _ | -__| || | | || _|| _| |_______|| __|_____|__|__||________||__| |____| |__| W I R E L E S S F R E E D O M ----- OpenWrt 22.03.3, r20028-43d71ad93e -----
	uname -a
default	`Linux svr04 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86_64 GNU/Linux`
OpenWrt	`Linux OpenWrt 5.10.161 #0 SMP Tue Jan 3 00:24:21 2023 x86_64 GNU/Linux`

doi:10.6342/NTU202303073

4.3 Networking

In our setup, we have 4 honeypots running on a single physical machine. The machine obtains 4 IP addresses from our ISP using PPPoE on a daily basis. Attackers in the public network can access these IP addresses on port 22 for SSH and port 23 for Telnet. The incoming traffic is redirected to different ports that correspond to each honeypot, as shown in 4.1

doi:10.6342/NTU202303073

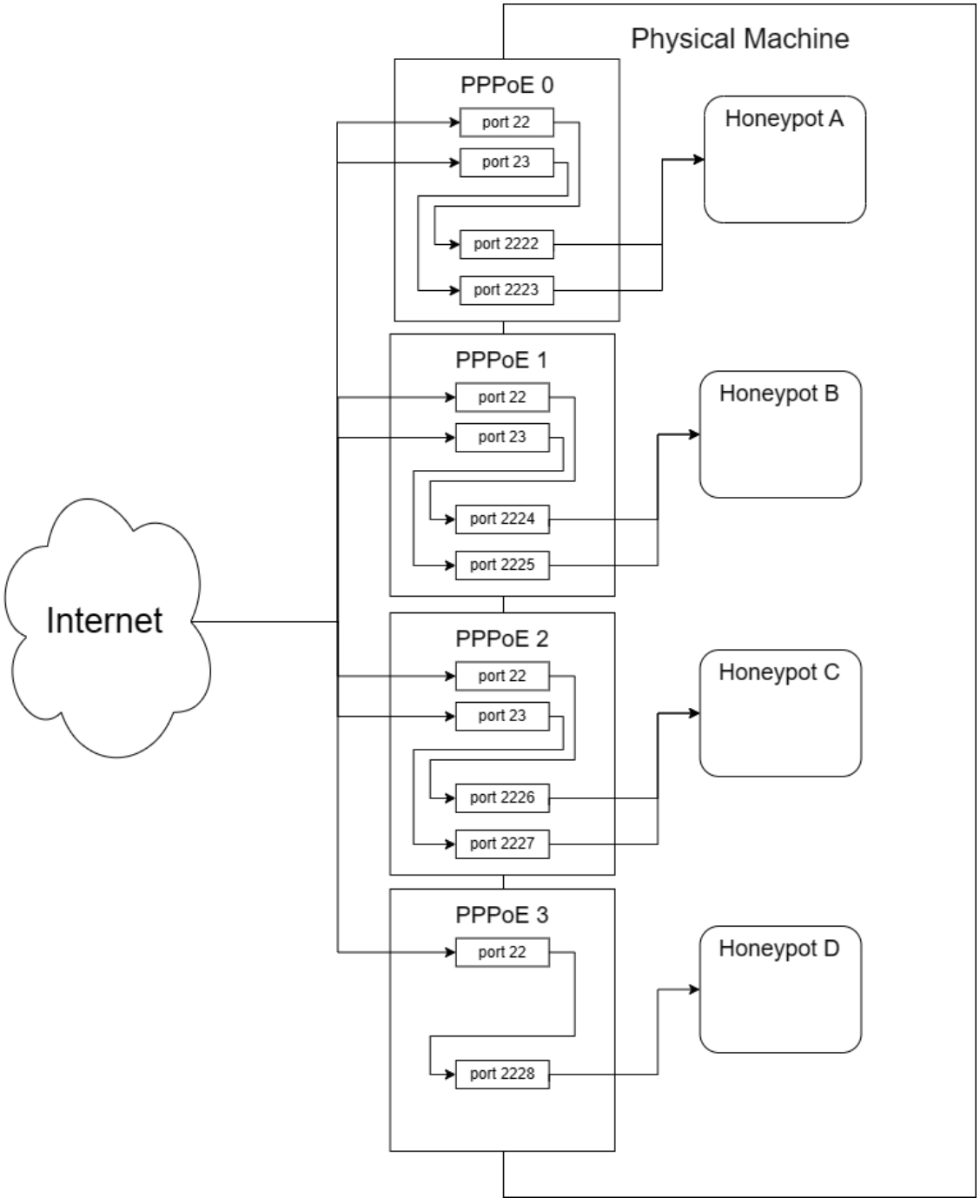


Figure 4.1: Network structure of our honeypots

doi:10.6342/NTU202303073

Chapter 5 Analysis

5.1 Overview of captured data

During the 30-day period from April 15, 2023, to May 14, 2023, we deployed our honeypots on a public network. Throughout this duration, we captured a total of 176,908 sessions from 14,783 unique IP addresses. The statistics for each honeypot are presented in Table 5.1 and 5.2.

As observed, honeypots A and B captured a significantly higher number of sessions compared to honeypots C and D. However, the number of unique IP addresses is similar across all honeypots, suggesting that each IP tends to initiate more connections to honeypots A and B. This observation aligns with the fact that attackers tend to spend more time in honeypot C, as demonstrated in Table 5.3.

If we narrow our focus to sessions that involve command inputs, the discrepancy becomes even more pronounced. This result indicates that our honeypot configurations indeed create a more realistic environment for attackers.

doi:10.6342/NTU202303073

Table 5.1: Number of sessions captured by each honeypot

sensor	protocol	number of sessions
A	ssh	51,630
	telnet	11,739
B	ssh	47,817
	telnet	15,556
C	ssh	21,319
	telnet	14,369
D	ssh	21,971
total	ssh	142,737
	telnet	41,664

Table 5.2: Number of IPs captured by each honeypot

sensor	protocol	number of IP
A	ssh	1,872
	telnet	4,368
B	ssh	2,288
	telnet	4,448
C	ssh	2,215
	telnet	4,937
D	ssh	1,886
total	ssh	4,383
	telnet	11,272
	ssh+telnet	15,526

Table 5.3: Average duration for all IPs in second

sensor	average duration for all sessions	average duration for sessions with command
A	29.52	16.59
B	29.69	16.78
C	38.65	56.52
D	19.52	27.27

doi:10.6342/NTU202303073

5.2 Classification based on command sequence

In our research, we classified attackers based on their command inputs for two main reasons. Firstly, we observed that many IP addresses used similar or even identical sequences of commands in the captured data. This allowed us to classify attackers by employing simple string matching techniques. Secondly, command sequences are easily understandable and can be manually classified by human analysts. We identified approximately 50 unique patterns of command sequences, but we filtered out some patterns that provided little information. For example, certain attackers only sent a single command such as "`uname -a`" or "`uname -s -v -n -r -m`". We selected 14 major patterns for further analysis, and their descriptions are provided in Table 5.4. We assigned names to each pattern based on the unique string it contains. If a pattern didn't have a representative string, we assigned it a unique number.

Table 5.4: Descriptions of each commands pattern

Pattern name	protocol	Descriptions
pattern 02	SSH	1. Collect OS name, version, release number and hostname using "`uname`". 2. Collect CPU information in file "`/proc/cpuinfo`". 3. Print number of processors using "`nproc`".
pattern 03	SSH	1. Check and start shell. 2. Print file "`/bin/echo`" or "`/proc/self/exe`".

doi:10.6342/NTU202303073

Table 5.4: Descriptions of each commands pattern

Pattern name	protocol	Descriptions
Gayfgt	SSH	1. Check and start shell. 2. Set `95.214.27.202` as a DNS server. 3. Print file "`/bin/echo`" or "`/proc/self/exe`".
		1. Check and start shell. 2. Remove all SSH authorized keys and add its own key. 3. Check download tools "`wget`" and "`curl`". 4. Print "gaygft" using "`echo`" command. 5. Download a file from `http://95[.]214.27.202/x86_64` and run it.
		1. Print file "`/bin/echo`" and "`/proc/mounts`". 2. Write "rppr/dev" into `/dev/.dabag` and remove it. 3. Print "MOUNTS_DONE" using `echo` command. 4. Check download tools "`wget`" and "`curl`". 5. Download a file from `http://95[.]214.27.202/x86` and run it. 6. Print "rppr" using `echo` command.

doi:10.6342/NTU202303073

Table 5.4: Descriptions of each commands pattern

Pattern name	protocol	Descriptions
pattern 39	Telnet	1. Check and start shell. 2. Print files "`/proc/mounts`" and "`/bin/echo`". 3. "`/ bin/ busybox {random string}`" at the end of each command.
Senpai	SSH	1. Check and start shell. 2. Write "senpai" in "`{path}/rootsenpai`" and delete it. `{path}` is usually `/tmp`. 3. Download a file named "`nig`", run it and remove it.
PingPong	SSH	1. Upload file to "`/tmp/{random}`" using scp. 2. Exit and reconnect. 3. Run the file.
Dred	SSH	1. Get system information using "`uname -a`". 2. Check graphic card information. 3. Download a file named "dred" from http://39[.]165.53.17:8088/ iposzz/ dred and run it with Perl.

doi:10.6342/NTU202303073

Table 5.4: Descriptions of each commands pattern

Pattern name	protocol	Descriptions
Hive	SSH	1. Get system information using ”`uname -a`”. 2. Change password. 3. Print the file ”`/hive-config/rig.conf`”
pattern 09	SSH	1. Remove all SSH authorized key and add its own keys. 2. Collect many information, such as CPU, memory, crontab and file system. 3. Change password.
Oto	SSH	1. Get system information using ”`uname -a`”. 2. Print ”`/etc/shadow`” and ”`/etc/passwd`”. 3. Get CPU information using ”`lscpu`”. 4. Change password. 5. Download a SSH authorized key, a Perl script and two binary files from http://124[.]70.7.7/. 6. Run Perl script and binary files.

doi:10.6342/NTU202303073

Table 5.4: Descriptions of each commands pattern

Pattern name	protocol	Descriptions
C3pool	SSH	1. Get system information using ”`uname -a`”. 2. Check GPU information. 3. Download a file from https:// raw[.]githubusercontent[.]com/ C3Pool/ xmrig_setup/ master/ setup_c3pool_miner.sh and run it.
Bot	Telnet	1. Check and start shell. 2. Remove all files under ”`/tmp`”. 3. Dowload a file from http:// 95[.]214.27.136/ bot.mips .
pattern 14	SSH	1. Download 4 files from 85[.]217.144.207 using `wget`, `curl`, `ftp` and run them.
Mikrotik	SSH	1. Use MikroTik RouterOS OS command ”`/ip cloud`” and ”`ifconfig`” to collect network information. 2. Get system information using ”`uname -a`”. 3. Print file ”`/proc/cpuinfo`”. 4. List information of many files.

doi:10.6342/NTU202303073

5.3 SSH identification string

As previously mentioned, the identification string is the first piece of information exchanged between the client and server in an SSH connection. It has been suggested in previous studies that attackers can detect the Cowrie honeypot by recognizing its default SSH identification string [9][21] [23]. However, we believe that the identification string can also be utilized by defenders as a means of defense.

A valid SSH identification string should start with "`SSH-2.0`" and end with "`\r\n`". For example, when using Putty, a widely used software for SSH, SFTP, and Telnet clients, the identification string sent to the server would be something like "`SSH-2.0-PuTTY_Release_0.78\r\n`".

In the captured data, it is common to find identification strings that are either unusual or extremely outdated. Since the identification string exchange occurs at the beginning of an SSH connection, blacklisting suspicious identification strings can be a simple yet effective defense mechanism against botnets. By maintaining a blacklist of known suspicious identification strings, defenders can identify and block potential malicious connections.

Moreover, analyzing the identification strings can provide insights into the language, package, or library used by the attackers. This information can be valuable for understanding the tools and techniques employed by the attackers and can aid in the development of stronger defense strategies.

Table 5.5 displays the top 10 identification strings captured by our honeypots, along with their basic information, sorted by the number of captures. Among these identification strings, 80% of them are "`SSH-2.0-Go`". Interestingly, sessions using this identification

doi:10.6342/NTU202303073

Table 5.5: Top 10 captured identification string.

identification string	Note	count
SSH-2.0-Go	Golang's SSH package [13]	100,834
SSH-2.0-libssh2_1.10.0	A C language SSH library [5]	6,232
SSH-2.0-HELLOWORLD	Reported to be used by botnet "RapperBot"[20]	3,723
SSH-2.0-dropbear	Software package used by OpenWrt	3,188
SSH-2.0-libssh2_1.7.0	A C language SSH library [5]	2,064
SSH-2.0-libssh2_1.4.3	A C language SSH library [5]	1,680
SSH-2.0-AsyncSSH_2.1.0	A Python SSH package [7]	1,217
SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2	Cowrie default	931
GET / HTTP/1.1	Wrong protocol	729
SSH-2.0-libssh_0.9.6	A C language SSH library [4]	714

string typically only send a "uname" command. It is worth noting that some identification strings include software versions that are outdated. For instance, the identification string "SSH-2.0-libssh2_1.4.3" corresponds to a version released in 2012, which is ten years ago[5].

An interesting observation is that some attackers reflected the identification string provided by the server side. For example, certain IP addresses sent "SSH-2.0-OpenSSH_6.0p1 Debian-4+deb7u2" when accessing honeypot A, but sent "SSH-2.0-dropbear" when accessing honeypots B, C, and D. This alignment between the attackers' identification strings and our honeypots' configurations serves as evidence supporting our assumptions.

doi:10.6342/NTU202303073

5.4 Parallel intrusion

During our experiment, we noticed a distinction in the behavior of attackers regarding their connection initiation strategies. Some attackers attempted to establish multiple connections simultaneously within a short timeframe, while others initiated new connections sequentially after the previous one had ended. This discrepancy is attributed to the program structure employed by the attackers.

In theory, botnets utilizing a parallel intrusion strategy have the potential for a higher infection rate. This approach allows them to brute-force passwords at a faster pace and infect multiple machines simultaneously. As a result, it is crucial to prioritize the defense against such botnets in advance.

By identifying and understanding these different connection initiation strategies employed by attackers, defenders can enhance their security measures and allocate resources effectively. This knowledge enables them to focus on mitigating the risks posed by botnets that employ parallel intrusion strategies, ultimately bolstering their overall defense against cyber threats.

To measure this aspect, we compute the ”overlap rate” for each IP address. The ”overlap rate” is a metric that we define in definition 5.4.1. For instance, let’s consider an IP address that visited our honeypots three times, as illustrated in the timeline depicted in figure 5.1. The total connection time is $9 + 3 + 4 = 16$, and the total connection time overlapping with other connection is $3 + 3 + 1 + 1 = 8$, so the overlap rate is 50%.

When we examine the IP addresses that have made at least 10 connections, we observe a highly polarized distribution, as depicted in figure 5.2. To simplify the analysis, we

doi:10.6342/NTU202303073

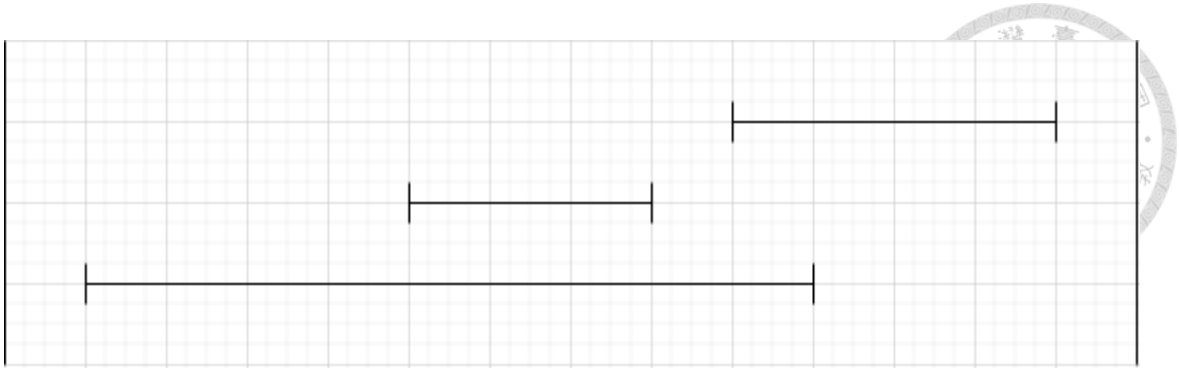


Figure 5.1: Example of calculating overlap rate

set a threshold at 50%. If the average overlap rate for IP addresses that have made at least 10 connections under a specific command pattern exceeds this threshold, we consider the pattern to indicate the use of a parallel intrusion strategy. Out of the 14 patterns listed in table 5.4, 5 of them exhibit this parallel intrusion feature. Further details will be presented subsequently.

Definition 5.4.1 (Overlap rate)**.** $\textit{overlap rate} = \frac{\text{Total connection time overlapping with other connection}}{\text{Total connection time}}$

5.5 Password length

The attackers who visited our honeypots attempted to infect victims by brute-forcing usernames and passwords. In practice, the password is often the last line of defense for a machine. We chose a simple attribute of the password for analysis: its length. It is natural to assume that longer passwords are stronger, but there is a trade-off between security and usability[10]. This analysis can help us determine how long a password should be considered sufficient. Figure 5.3 shows the statistics of the password lengths provided by attackers. It is observed that 94% of the passwords were not longer than 10 characters, which is shorter than the recommended advice of 12 characters from the United States Federal Trade Commission [19].

doi:10.6342/NTU202303073

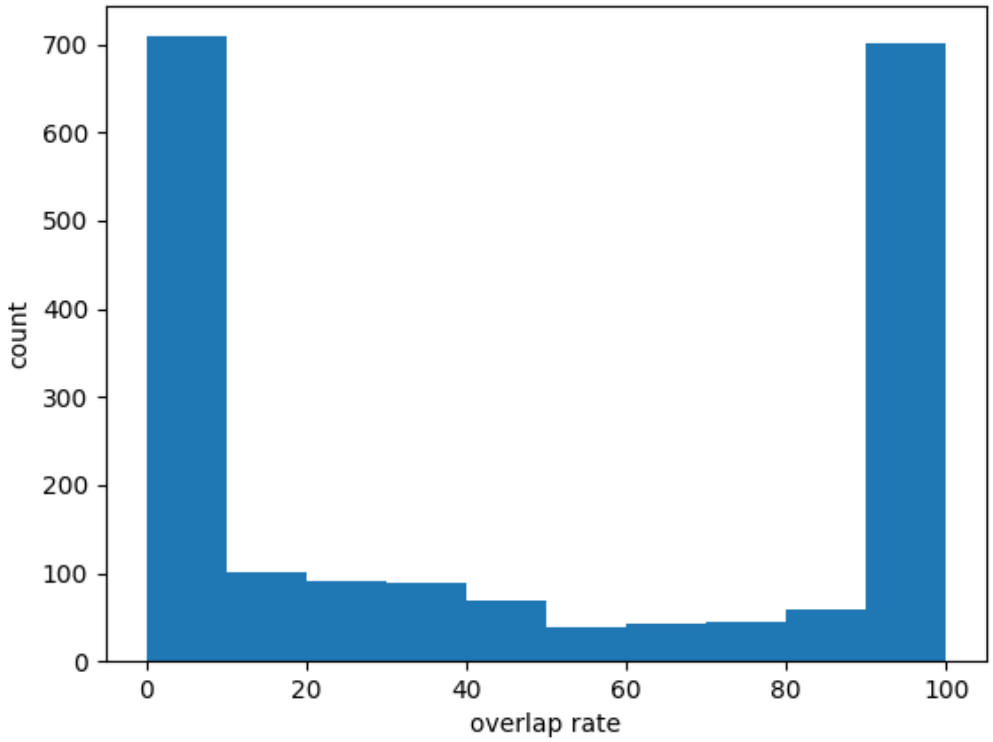


Figure 5.2: Histogram of overlap rate for IP addresses that make at least 10 connections

5.6 Cooperative intrusion

In the data we captured, we noticed that certain command patterns only occurred in specific honeypots. This observation is reasonable due to the different configurations of our honeypots. However, we also observed a similar phenomenon from the perspective of IP addresses. Some hosts made numerous connections over multiple days to certain honeypots but never visited others, not even initiating a TCP connection. This behavior is peculiar because our IP address changes daily, making it highly unlikely for an attacker to miss a specific honeypot during their internet scanning activities. The only plausible explanation is that these hosts are cooperating with other machines that perform reconnaissance on their behalf.

doi:10.6342/NTU202303073

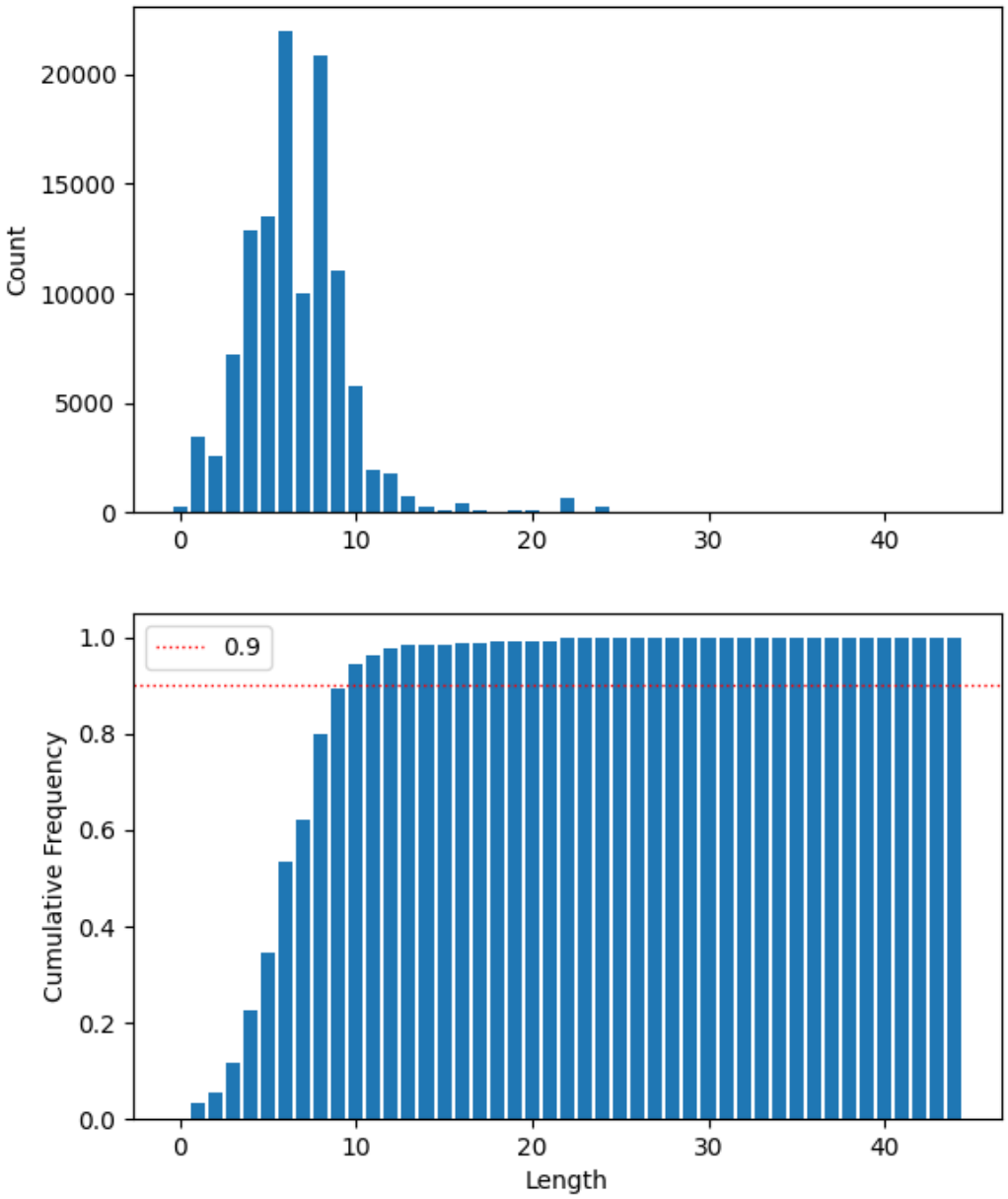


Figure 5.3: Password length histogram and cumulative frequency graph

doi:10.6342/NTU202303073

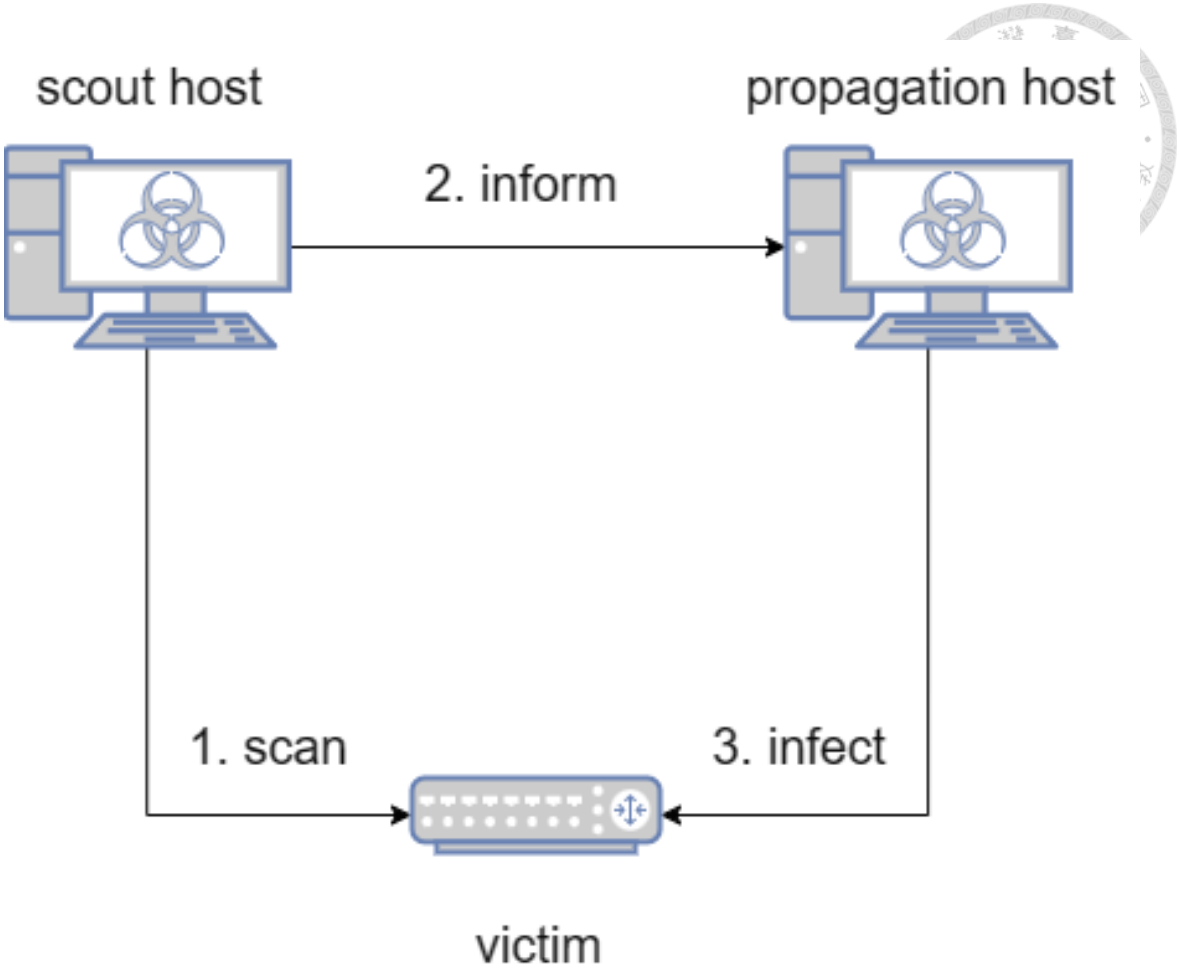


Figure 5.4: Cooperative intrusion

These ”scout” hosts visit our honeypots initially to gather information, which they subsequently transmit to ”propagation” hosts. The propagation hosts then decide whether to proceed with infection or not, as illustrated in figure 5.4. This division of labor is reminiscent of the behavior observed in the Mirai botnet [6]. A similar observation was made by [18], who noticed that some attackers successfully guessed passwords in only one attempt.

One concrete example from our experiment is the ”Gayfgt”, which selectively visited honeypots A, B, and C. Fortunately, we were able to identify its scout, pattern 03, easily. By examining the username-password sets used by each pattern, we discovered that pattern Gayfgtemployed a small set of 8 different username-password pairs, which was identical

doi:10.6342/NTU202303073

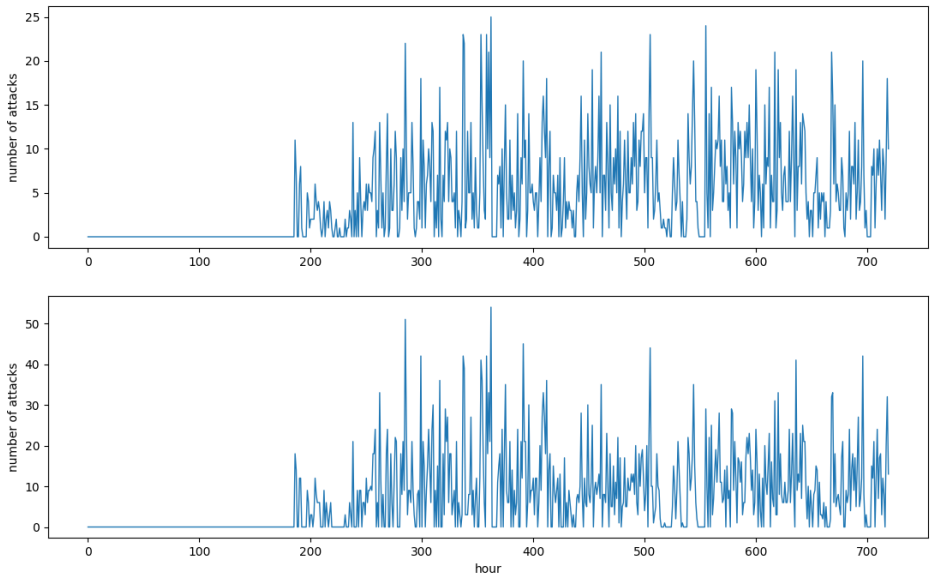
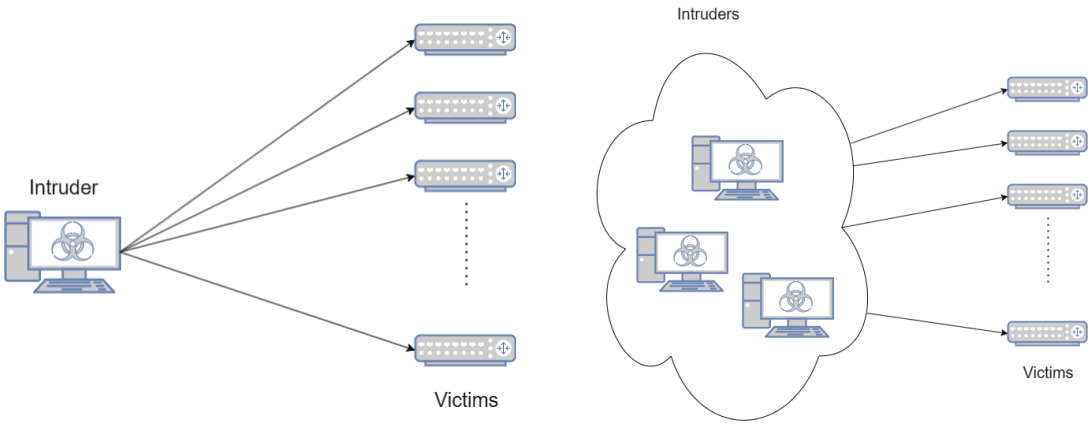


Figure 5.5: Attacks per hours of pattern 03(above) and Gayfgt(below)

to the set used by pattern 03. Additionally, both patterns started on the same day, and the peaks of attacks per hour aligned, as depicted in figure 5.5.

5.7 Single source intrusion and download



(a) Single source

(b) Multiple source

Figure 5.6: Difference between single source and multiple source

While analyzing the captured data, we observed that most of the command patterns were executed by multiple unique hosts. However, there were certain patterns that were

doi:10.6342/NTU202303073

performed by a single host, as depicted in the figure 5.6. This difference in behavior is indicative of variations in botnet architecture.

A similar observation can be made regarding the sources from which attackers download malicious files. Although the botmaster, who manages the botnet system, can utilize the Domain Name System (DNS) to dynamically allocate C&C (Command and Control) servers [11], the URLs for file downloads we encountered were predominantly written as IP addresses. For example, "`http://95[.]214.27.202/x86`". This allows us to trace the source of the downloaded files. The sources of the downloaded files can be categorized into four types:

1. Self-propagating: The host that has compromised the victim system sends the file itself.

2. Single IP: The file is downloaded from a fixed IP address.

3. Multiple IP: The file is downloaded from multiple different IP addresses.

4. Single domain name: The file is downloaded from a fixed domain name.

This information is valuable because blacklisting the IP addresses of those single-source attackers would be a straightforward defense strategy for individual machines. From a broader perspective, Internet Service Providers (ISPs) can choose to reject forwarding packets to these IP addresses, effectively limiting the propagation of botnets.

doi:10.6342/NTU202303073

Table 5.6: Comparison with other works

work	[18]	[24]	[15]	[22]	[8]	[14]	Our work
self made honeypot	o	o	o				
parallel intrusion							o
SSH identification string			o				o
username/password	o		o	o	o	o	o
commands classification	o	o				o	o
download file classification	o	o	o	o		o	o
botnet architecture	o					o	o
coorperate intrusion	o						o
cross analysis of each features	o	o					o
cross analysis of multiple honeypots							o

5.8 Summary

We summarize our observation in table 5.7. We can found that every single source intrusion attackers also apply parallel intrusion, this is predictable because otherwise the infect rate will be highly limited. Many of them use SSH libraries whose versions are older than five years. The fact that most of them download files from a single source indicates attackers may hard code the URL in the program. We classified the downloaded binary using free tool VirusTotal[1], the result include Mirai, Tsunami and so on.

5.9 Comparison with other works

We provide a similar depth of analysis as [18] and [24], while offering more detailed insights compared to [15], [22], [8], and [14]. Our emphasis is on identifying features that are valuable for security researchers and network administrators in designing effective defense strategies. The detail comparison is shown in 5.6.

doi:10.6342/NTU202303073

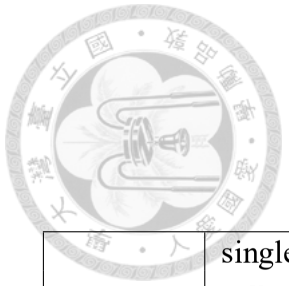


Table 5.7: Summary of analysis

	single source	parallel	protocal	SSH identification string		Download source	Malware Family
				String	Discription		
pattern 02			SSH	SSH-2.0-libssh2_1.10.0	Up to date		
pattern 03			SSH		Reflect		
Gayfgt	○	○	SSH	SSH-2.0-HELLOWORLD	Custom	single IP	Mirai
				SSH-2.0-libssh2_1.7.0	> 5 years		
pattern 39			Telnet				
Senpai			SSH	SSH-2.0-libssh2_1.10.0	Up to date	multiple IP	Mirai
PingPong		○	SSH	SSH-2.0-OpenSSH_X.Xp1 {OS version}	> 1 year	self-propagating	IRCBot
Dred			SSH	SSH-2.0-libssh2_1.4.3	> 10 years	single IP	Shellbot
Hive			SSH	SSH-2.0-libssh2_1.4.3	> 10 years		
pattern 09			SSH	SSH-2.0-libssh_0.9.6	> 1 year		
				SSH-2.0-libssh-0.6.3	> 5 years		
Oto	○	○	SSH	SSH-2.0-libssh2_1.4.3	> 10 years	single IP	Tsunami
C3pool			SSH	SSH-2.0-libssh2_1.4.3	> 10 years	single domain name	Minerdownloader
Bot	○	○	Telnet			single IP	Mirai
pattern 14		○	SSH	SSH-2.0-Go		single IP	Medusa
Mikrotik			SSH	SSH-2.0-libssh2_1.8.2	> 1 year		
				SSH-2.0-libssh2_1.10.0	Up to date		
				SSH-2.0-libssh2_1.9.0	> 1 year		

doi:10.6342/NTU202303073

5.10 Guideline

Based on our data, we have formulated a simple guideline for individuals to protect their IoT devices:

1. Avoid enabling Telnet: Telnet transmits data in plain text, allowing attackers to eavesdrop on usernames and passwords. It is recommended to disable Telnet to enhance security.

2. Whitelist SSH identification strings: Users can create a whitelist for the SSH identification strings of the software they commonly use. For example, PuTTY [3] utilizes an identification string starting with ”SSH-2.0-PuTTY.” While only 0.2% of the captured sessions used this string, the occurrence of another popular software, OpenSSH, was 1.3%. Implementing this rule can protect machines from SSH brute-force attacks.

3. Use passwords longer than 10 characters: As mentioned in the previous chapter, 94% of the login attempts used passwords that were not longer than 10 characters. It is advisable to select longer passwords to enhance security.

For internet service providers (ISPs), implementing IP address bans for those found in attackers' commands can be an effective and cost-efficient way to prevent the propagation of botnets. In our data, we observed approximately 4,500 download attempts using tools like `wget` and `curl`, but only 29 download hosts were identified. Additionally, only one of these hosts was specified using a domain name, while the rest were all IP addresses.

doi:10.6342/NTU202303073

doi:10.6342/NTU202303073

Chapter 6 Conclusion and Future Work

In our study, we have showcased the benefits of deploying multiple diverse honeypots in gaining valuable insights into botnet attacks. We have highlighted novel features about attackers that can greatly enhance the strategies of network security defenders. Over the course of our 30-day deployment, we observed approximately 50 unique command patterns, focusing on 14 major patterns for detailed analysis. This work not only contributes to the understanding of the current botnet ecosystem but also provides readers with a foundational understanding of botnet ecology.

We have discovered intriguing behaviors exhibited by attackers that were not addressed in our current study. However, these behaviors merit further investigation due to their potential significance and relevance.

1. Repeatedly infection: We have observed that certain attackers repeatedly infect a single honeypot within a short period of time. This behavior may be a result of either flaws in the botnet's program design or the statelessness of the Cowrie honeypot, which consistently offers a fresh environment to visitors. Repeatedly infecting the same honeypot is undoubtedly an inefficient use of resources for attackers. We

doi:10.6342/NTU202303073

believe that it is possible to leverage this characteristic as a means to counteract and exploit vulnerabilities in botnets.

2. Waiting for response: In our captured data, we have observed instances where attackers send input based on the output of their previous commands. For example, they enter a directory that appears in the file ”/proc/mounts”. This behavior suggests that we can potentially make the attacker wait longer by deliberately delaying our response. Exploring how to effectively implement this delay in a subtle and strategic manner could be an interesting topic for further investigation.

doi:10.6342/NTU202303073

References

[1] Virustotal. https://www.virustotal.com/.

[2] Telnet Protocol Specification. RFC 854, May 1983.

[3] Putty. https://www.putty.org/, 2022.

[4] libssh: The ssh library! https://www.libssh.org/, 2023.

[5] libssh2: the ssh library. https://www.libssh2.org/, 2023.

[6] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In 26th {USENIX} security symposium ({USENIX} Security 17), pages 1093–1110, 2017.

[7] AsyncSSH. Asynchronous ssh for python. https://asyncssh.readthedocs.io/en/latest/, 2023.

[8] M. Başer, E. Y. Güven, and M. A. Aydın. Ssh and telnet protocols attack analysis using honeypot technique: Analysis of ssh and telnet honeypot. In 2021 6th International Conference on Computer Science and Engineering (UBMK), pages 806–811. IEEE, 2021.

doi:10.6342/NTU202303073

[9] W. Z. Cabral, C. Valli, L. F. Sikos, and S. G. Wakeling. Advanced cowrie configuration to increase honeypot deceptiveness. In ICT Systems Security and Privacy Protection: 36th IFIP TC 11 International Conference, SEC 2021, Oslo, Norway, June 22–24, 2021, Proceedings, pages 317–331. Springer, 2021.

[10] K. Chanda. Password security: an analysis of password strengths and vulnerabilities. International Journal of Computer Network and Information Security, 8(7):23, 2016.

[11] D. Dagon, C. C. Zou, and W. Lee. Modeling botnet propagation using time zones. In NDSS, volume 6, pages 2–13, 2006.

[12] droberson. ssh-honeypot. https://github.com/droberson/ssh-honeypot, 2022.

[13] Go. ssh. https://pkg.go.dev/golang.org/x/crypto/ssh, 2023.

[14] S. Kemppainen and T. Kovanen. Honeypot utilization for network intrusion detection. Cyber Security: Power and Technology, pages 249–270, 2018.

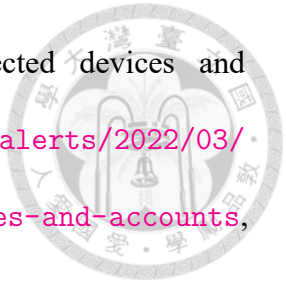
[15] M. Knöchel and S. Wefel. Analysing attackers and intrusions on a high-interaction honeypot system. In 2022 27th Asia Pacific Conference on Communications (APCC), pages 433–438. IEEE, 2022.

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

[17] Oosterhof. cowrie. https://github.com/cowrie/cowrie, 2023.

[18] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow. Iotpot: Analysing the rise of iot compromises. Emu, 9(1), 2015.

doi:10.6342/NTU202303073



[19] A. Puig. Cybersecurity advice to protect your connected devices and accounts. https://consumer.ftc.gov/consumer-alerts/2022/03/cybersecurity-advice-protect-your-connected-devices-and-accounts, 2022.

[20] J. Salvio and R. Taya. Rapperbot ddos botnet expands into cryptojacking. https://www.fortinet.com/blog/threat-research/rapperbot-ddos-botnet-expands-into-cryptojacking, 2023.

[21] O. Surnin, F. Hussain, R. Hussain, S. Ostrovskaya, A. Polovinkin, J. Lee, and X. Fernando. Probabilistic estimation of honeypot detection in internet of things environment. In 2019 International Conference on Computing, Networking and Communications (ICNC), pages 191–196. IEEE, 2019.

[22] A. Z. Tabari and X. Ou. A first step towards understanding real-world attacks on iot devices. arXiv preprint arXiv:2003.01218, 2020.

[23] A. Vetterl, R. Clayton, and I. Walden. Counting outdated honeypots: Legal and useful. In 2019 IEEE Security and Privacy Workshops (SPW), pages 224–229. IEEE, 2019.

[24] B. Wang, Y. Dou, Y. Sang, Y. Zhang, and J. Huang. Iotcmal: Towards a hybrid iot honeypot for capturing and analyzing malware. In ICC 2020-2020 IEEE International Conference on Communications (ICC), pages 1–7. IEEE, 2020.

doi:10.6342/NTU202303073