HOW DO HACKERS HACK? – MOTIVATIONS, TECHNIQUES AND TOOLS

1 Introduction

In order to develop effective cyber-defences, it is crucial to first understand how hackers hack. Without this knowledge, mounting or building any defences may result in unexpectedly poor results. One way to obtain this knowledge is through the deployment of honeypots, a type of decoy system that lures attackers towards it by appearing vulnerable and exploitable [1]. However, honeypots that can be configured with ease and are quick and easy to deploy are lacking [2]. Honeypots also need to appeal to attacker motivations to effectively lure attackers. Hence, the recent example of CVE-2021-44228 or "Log4Shell" was used as a case study to understand more about current attacker behaviour to develop an effective honeypot.

Two Log4Shell research tools were also examined: Log4Pot [3] and log4shell-vulnerable-app [4]. Observations of their functionalities corroborate the opinions that most current honeypots are difficult to configure and deploy, as deploying Log4Pot requires multiple steps and additional dependencies [3]. Additionally, both tools suggest a lack of comprehensive logging within honeypots, as log4shell-vulnerable-app does not implement any form of logging and Log4Pot does not log the attacker's IP address. Furthermore, since log4shell-vulnerable-app runs a vulnerable version of Log4j, it also risks compromising the host computer's security by potentially downloading malware. It thus raises potential security issues with running actual vulnerable services within a honeypot.

To address the above flaws, a prototype honeypot was developed. It consists of several network namespaces within which, server emulation scripts can be executed. Doing so will result in the network namespaces appearing as either FTP or SSH servers. Logging of access attempts was implemented within the prototype, along with easy configuration. Due to constraints, the prototype developed was only tested in an isolated environment. Quantitative and qualitative evaluation of it was also performed to gauge its effectiveness.

2 CVE-2021-44228, "Log4Shell"

CVE-2021-44228, commonly referred to as "Log4Shell", was a vulnerability in Apache Log4j 2, a Java library used for logging [5]. Log4Shell enabled attackers to conduct Remote Code Execution (RCE) by submitting attack strings to services running Log4j, which would then be logged. The attack string would be formatted as such: "${jndi:ldap://query}", where "query" is a placeholder for a malicious domain and the exploit. This would prompt a JNDI lookup for the query at an LDAP server controlled by the attacker. Variations also later include using the RMI API instead of LDAP [6]. Log4Shell attacks were prevalent in December 2021 but were predicted to persist for a long time due to the large scale of the vulnerability.

The phases of a typical Log4Shell attack are as follows [6]:

1. The attack string is sent to and logged by the victim.
2. The attack string is resolved by the JNDI interface and performs a lookup at the attacker-controlled LDAP or RMI server.

3. The malicious server returns code, typically bash, that is executed on the victim's machine.
4. The malicious code then downloads and executes malware.

2.1 Attacker Motivations, Techniques and Tools

Researchers in [7] conducted a study on the use of scanners by both researchers and attackers in December and January 2021 with respect to Log4Shell. By collecting the payloads, they were able to classify the scanners based on their maliciousness. The study found that generally, malicious scanners were most deployed during December 2021 when the vulnerability was first discovered [7], indicating that attempted attacks peaked during that period.

Attacker motivations in the Log4Shell incident vary greatly due to the scale of the incident. However, attacks can be said to be motivated by the RCE capabilities of Log4Shell attacks. RCE allowed attackers to carry out other phases of an attack such as privilege escalation and persistence within the compromised network or system. It is easily used in conjunction with other types of malwares such as cryptocurrency mining tools and information stealers as seen in [8], a compilation of Log4Shell exploitation attempts and their payloads. Some recorded examples of payloads include XMRig and the Mirai botnet [8]. Hence, the additional attack opportunities that a successful Log4Shell attack provided appealed to attackers.

Attackers also employed obfuscation techniques to thwart defensive attempts to sanitise such user-controlled input for attack strings. By using JNDI nested strings instead of purely "${jndi:ldap://}, attack strings could pass through web-application firewalls. Examples of string obfuscation techniques include the usage of lower and upper lookups and system environment variables [9].

The study also discusses the ease of conducting such attacks in relation to the tools used by attackers, citing the widespread availability of public YouTube videos [7]. These tutorial videos explained how to conduct an attack in detail, thus allowing even inexperienced attackers to exploit the vulnerability. There were also many ready kits for setting up malicious LDAP servers publicly available on GitHub. Notably, JNDIExploit by feihong-cs can easily be used to set up a malicious LDAP server to conduct a Log4Shell attack, and also includes instructions and a list of supported payloads. While the original has been deleted, many copies can still be found on repositories around GitHub such as [10].

In summary, attackers are motivated to conduct an attack when doing so provides value to them, such as enabling further attack phases like privilege escalation. If there are readily available tools, attackers may also be influenced by the ease of carrying out a specific attack to target the corresponding vulnerability. These principles were subsequently applied in the development of a honeypot in order to strengthen its ability to lure attackers towards it and away from actual important systems.

3 Project Design

To address the flaws of current honeypots, the project aims to develop a honeypot using Python that consists of multiple network namespaces to emulate a busy network. A network namespace is a container within the Linux kernel with an isolated network environment with its own routing table and network interfaces [11] and to an attacker performing a network scan, each namespace should appear to be an actual machine operating on the network. Within the namespaces, server emulation scripts were executed, causing the namespaces to appear as either File Transfer Protocol (FTP) or Secure Shell Protocol (SSH) servers. Attempts to access the emulated servers would be logged and user configuration was also implemented within the honeypot. Additionally, the honeypot script was written in a way that allowed it to be deployed by running a single command in the terminal.

The materials used in the development of the honeypot prototype are Kali Linux[1], Wireshark, and the following Python modules: Scapy, Paramiko, psutil, subprocess, multiprocessing, csv, datetime, time and ast. The run_steps function in [12] was also used as reference for writing the code needed to create the network namespaces within Python.

3.1 Choice of Implemented Protocols

FTP was deliberately chosen because of its straightforwardness, thus making it easier to implement within the prototype, as well as its reputation for being very insecure. RFC 2577 lists several security vulnerabilities in FTP, such as a lack of password protection from malicious actors who may be listening on a conversation between server and client as payload data is unencrypted, and brute-force attacks [13]. An abundance of publicly-available tutorials on brute-forcing access into an FTP server such as [14] also makes attacking an FTP server appealing to attackers. Hence, an FTP server should be able to serve as an effective decoy for attackers.

Conversely, SSH is more secure due to its encryption payload data. However, it is still vulnerable to brute-force attacks and is a valuable target for attackers as it allows for remote login and command execution [15], which would facilitate further attack phases such as lateral movement to the other network namespaces.

[1] While Kali Linux was used for the development of the prototype, it is not specifically necessary for replication. Namespaces are a Linux kernel feature which any Linux machine should have.

3.2 Network Namespaces

Initially, the honeypot was developed on a smaller scale without the implementation of user-configurability. Three network namespaces named ns1, ns2 and ns3 were first created with the method described in [16]. This was done to emulate an internal network as seen below in Figure 1.

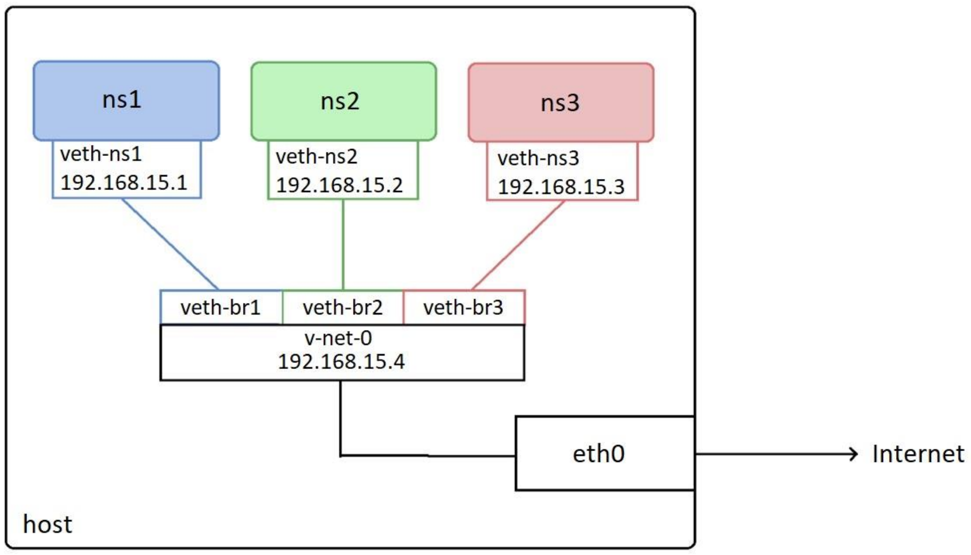


Figure 1: *Simplified network diagram of the namespaces within the host machine.*

The bridge, labelled "v-net-0" in **Figure 1**, acted as a gateway to allow each network namespace to communicate with each other and external networks. Using the commands in Listing 1, pairs of virtual ethernet devices were created for each namespace. The interface that is named in the format of <veth-nsx>, was placed in the corresponding namespace. Meanwhile, the interface that is named in the format of <veth-brx> would be attached to the bridge.

```
ip link add <veth-brx> type veth peer name <veth-nsx>
ip link set <veth-brx> master v-net-0
ip link set <veth-nsx> netns <nsx>
```

Listing 1: Shell commands to create virtual ethernet device pairs. x represents the corresponding number of the namespace.

To allow the network namespaces to reach an external network, a route was created within each network namespace using the bridge as the default gateway. Packet forwarding between eth0 and the bridge was enabled within the firewall for the route to work. This is because packets originating from the namespace must first be forwarded from the bridge to eth0 to reach external networks.

3.3 Server Emulation

Rather than running legitimate FTP and SSH servers within the namespaces, the vulnerable services were emulated through Python scripts. This is to minimise the security risks observed with the log4shell-vulnerable-app which ran a vulnerable version of Log4j and could potentially download malware that compromises the system [4]. Doing so also bypasses the difficulties in configuring commercially available or open-source FTP and SSH servers.

The Scapy Python library was used in writing a Python script to craft and send the appropriate packets back to an attacker who may be trying to open a session with what appears to be an FTP server from the attacker's point of view. First, a normal exchange between a legitimate FTP server (vsftpd 3.0.3) and a client was observed using Wireshark and outlined within **Figure 2**.

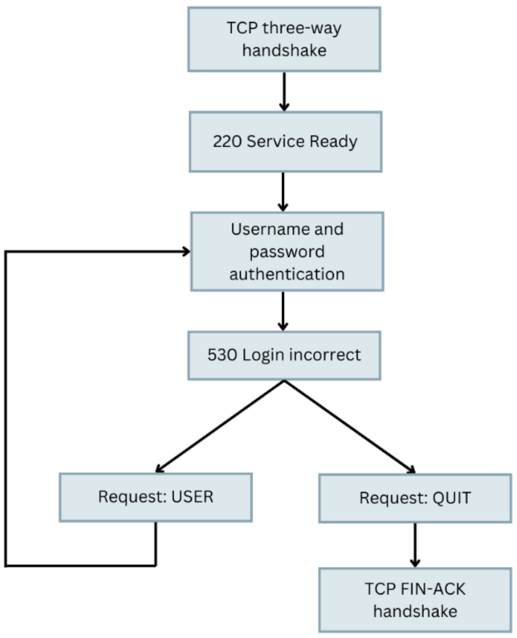


Figure 2:*Flowchart of a normal exchange between an FTP server and a client when incorrect credentials are provided.*

Only the case of a failed login was observed as the fake FTP server is not intended to have any accepted credentials; no combination of usernames and passwords will grant the attacker access. After a failed login attempt, the attacker can pass either the USER request to reattempt to log in, or the QUIT request to end the session.

The honeypot first crafts and sends a SYN-ACK packet after receiving SYN from the client. An iptables rule to drop all outbound RST packets from port 21 is added in order to allow the honeypot to receive ACK from the client before the kernel sends RST-ACK. Subsequently, server response packets are crafted and sent in response to client requests to replicate a normal FTP server-client conversation.

Unlike FTP, SSH has a readily available Python library that allows for the implementation of server functionality: Paramiko. Hence, rather than crafting individual response packets similar to emulating the FTP server, Paramiko is used. This is also due to the complexity of the SSH key exchange initialisation and generation which is hard to manually replicate using Scapy. [17] was used as a reference for writing a Paramiko server script and modified according to the needs of the project.

3.4 Configuration and Logging

To address the problem of configurability raised in [2], a configuration text file was created and integrated into the honeypot. By changing the values in the file, the user can easily choose the number of namespaces created, the network ID of the namespaces, the host IDs of the namespaces, and which server emulation scripts should run in each namespace. An example of the contents of the configuration file can be seen in Appendix A.

Additionally, attempts by attackers to connect to any of the "server" network namespaces would be logged and written to separate files for either FTP or SSH. A separate script was written to analyse the contents of the FTP logs. Running it would clean the payloads if needed and determine when a session started and ended, as well as how long it lasted. A feature to detect attempts to inject a Log4Shell attack string by searching for "${", "j", "n", "d", "i" in the payload was added. Obfuscation attempts can also be detected. Compared to Log4Pot, log analysis does not attempt to resolve any obfuscated strings and maintains the original in order to allow for obfuscation methods to be studied. Log analysis for Log4Shell attack strings was not implemented for the SSH "servers" as from an attacker's point-of-view, they are not Java-based and would not be running a vulnerable version of Log4j.

3.5 Validation of Prototype

The prototype was tested with the four network namespaces, ns1, ns2, ns4 and ns6, with the following IP addresses: 192.168.15.1, 192.168.15.2, 192.168.15.4 and 192.168.15.6, with 192.168.15.7 as the bridge IP address. The FTP server emulation script was executed in ns1 and ns2, while the SSH server emulation script was executed in ns1 and ns6. From a separate machine, attempts were made to open FTP and SSH sessions with the network namespaces that run server emulation scripts. Login attempts were then made after establishing a connection. Test results indicated that the honeypot was successful in replicating the FTP and SSH services (see Appendix B). Connectivity from an external machine to the network namespaces was also tested using the ping command. A route was added to the external machine to reach the network namespaces and test results indicated success in establishing connectivity (see Appendix C).

4 Evaluation Results

To assess the believability of the emulated servers, the time taken for the honeypot to send a packet was compared to the time taken for an actual server to send a packet.

The evaluative experimental procedure is described below:

1. Sniff a conversation between the honeypot and the client with Wireshark.
2. Record the times taken by the honeypot to send each packet within the conversation.
3. Calculate the average time taken for the honeypot to send a packet as $<t_1>$.
4. Repeat steps 1 to 3, this time recording the average time taken for the honeypot to send a packet as $<t_2>$.
5. Repeat steps 1 to 3, this time using an actual server instead of the honeypot. Record the average time taken for the actual server to send a packet as $<t_0>$.

The experiment was conducted once each for FTP and SSH. vsftpd 3.0.3 was used as the actual FTP server while OpenSSH 9.1 was used as the actual SSH server. Both versions were used as they were the latest.

Protocol	**$<t_1>$/s**	**$<t_2>$/s**	**$<t_0>$/s**

FTP	0.0441	0.0448	0.000301
SSH	0.0167	0.0127	0.00895

***Table 1**: Average time taken to send a packet*

All values are recorded to three significant figures in **Table 1**. The full dataset collected for FTP and SSH can be found in Appendix D and E respectively. The 8th data point was regarded as anomalous in the calculations for $<t_0>$ for FTP as it was abnormally large and can be attributed to intentional delay as part of the vsftpd settings [18]. For SSH, the 11th, 13th and 15th data points were abnormally large and regarded as anomalous in the calculations for $<t_0>$.

4.1 Discussion of Results

There is little difference between the average times taken for the honeypot to send a packet in both SSH conversations and the average time taken for OpenSSH 9.1 to send a packet. However, the average times taken for the honeypot to send a packet in both FTP conversations and the average time taken for vsftpd 3.0.3 to send a packet differ by two orders of magnitude. This can be attributed to the source code for the sendrecv class in Scapy. Every time a packet is sent, a raw socket is opened and then closed [19], slowing the process. While the difference is not noticeable by the average person, this deviation from normal behaviour may raise attacker suspicions if noticed. This can be overcome by either modifying the source code to keep the socket open or by manually creating a socket without the use of Scapy [19]. However, due to time constraints, this was not implemented within the current prototype. Furthermore, using a manually-created socket without Scapy requires each packet to be constructed as an array of raw bytes. This would negatively impact code readability.

Nevertheless, when a nmap scan is performed, namespaces that run the FTP server emulation script cannot be detected as up although attackers can successfully connect to them. This affects their discoverability and limits their effectiveness without the presence of other information that points towards the existence of the FTP "servers". However, the namespaces that run the SSH server emulation scripts can be detected through a nmap scan with no noticeable variations from the norm (see Appendix F).

Overall, the emulated SSH server is highly believable while the emulated FTP server is flawed, though adjustments can be feasibly made to increase its deceptive ability.

5 Conclusion

By studying the Log4Shell case study, it was understood that the ease of carrying out an attack is a primary motivator for attackers. A prototype honeypot was then created using network namespaces, where server emulation scripts for either the FTP or SSH service can be executed. Ultimately, while far from perfect, the prototype fulfils its objectives in comparison to most existing honeypots: not only is it easy to deploy and configure, it also minimises security issues by manually replicating the vulnerable services rather than running actual copies of them. Future work can include expanding the honeypot to include server emulation scripts of other protocols, as well as ensuring discoverability of the emulated servers via nmap. Further development of the prototype would also open the possibility of field-testing for a more comprehensive evaluation.

References

[1] Lutkevich, B., Clark, C., and Cobb, M. (2021). What is a honeypot? How it protects against cyber-attacks. https://www.techtarget.com/searchsecurity/definition/honey-pot. Date accessed: 22 December 2022

[2] Acosta, C.J., Basak, A., Kiekintveld, C., and Kamhoua, C. (2022). Lightweight On-Demand Honeypot Deployment for Cyber Deception. Digital Forensics and Cyber Crime: p294-312. https://doi.org/10.1007/978-3-031-06365-7_18

[3] Patzke, T. (2022). Log4Pot. [Computer Software]. GitHub. https://github.com/thomaspatzke/Log4Pot. Date accessed: 25 July 2022.

[4] Tafani-Dereeper, C. (2022). log4shell-vulnerable-app. [Computer Software]. GitHub. https://github.com/christophetd/log4shell-vulnerable-app. Date accessed: 6 August 2022.

[5] National Institute of Standards and Technology. 2022. CVE-2021-44228 Detail. https://nvd.nist.gov/vuln/detail/CVE-2021-44228. Date accessed: 14 July 2022.

[6] Burt, A., and Langton, A. (2021). Log4j Vulnerability: Attackers Shift Focus From LDAP to RMI. https://blogs.juniper.net/en-us/threat-research/log4j-vulnerability-attackers-shift-focus-from-ldap-to-rmi. Date accessed: 14 July 2022.

[7] Hiesgen, R., Nawrocki, M., Schmidt, T. C., and Wählisch, M. (2022). The Race to the Vulnerable: Measuring the Log4j Shell Incident. https://doi.org/10.48550/arXiv.2205.02544

[8] Curated Intelligence Trust Group. (2022). Log4Shell-IOCs. GitHub. https://github.com/curated-intel/Log4Shell-IOCs. Date accessed: 23 December 2022.

[9] Pulikowski, M., and Friman, J. (2022). CVE-2021-44228-PoC-log4j-bypass-words. GitHub. https://github.com/Puliczek/CVE-2021-44228-PoC-log4j-bypass-words. Date accessed: 2 August 2022.

[10] nil-malh. (2021). JNDI-Exploit. [Computer Software]. GitHub. https://github.com/nil-malh/JNDI-Exploit. Date accessed: 6 August 2022.

[11] Kerrisk, M. 2021. network_namespaces(7) — Linux manual page. https://man7.org/linux/man-pages/man7/network_namespaces.7.html. Date accessed: 26 August 2022.

[12] larsk. (2019). python-netns. (Version 1.0). [Computer Software]. GitHub. https://github.com/larsks/python-netns. Date accessed: 28 August 2022.

[13] Allman, M., and Ostermann, S. (1999). FTP Security Considerations. RFC 2577. https://www.rfc-editor.org/info/rfc2577.

[14] Null Byte. (2020). Brute-Forcing FTP Credentials for Server Access [Tutorial]. [Video]. YouTube. https://www.youtube.com/watch?v=hE_Kjav323U

[15] Ylonen, T. (2006). The Secure Shell Protocol (SSH) Architecture. RFC 4251. (C. Lonvick, Ed.) https://www.rfc-editor.org/info/rfc4251

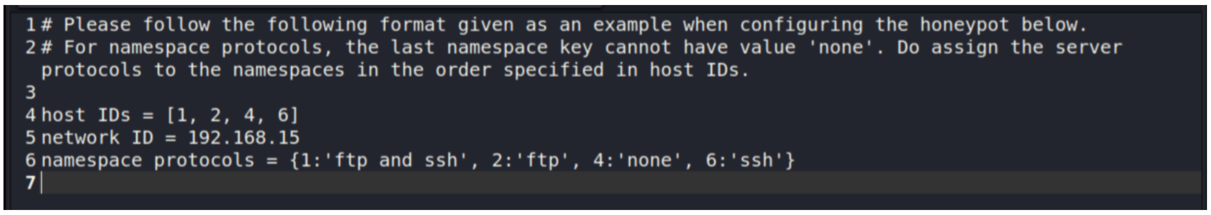
[16] KodeKloud. (2019). Network Namespaces Basics Explained in 15 Minutes. [Video]. YouTube. https://www.youtube.com/watch?v=j_UUnlVC2Ss.

[17] ysc3839. (2016). FakeSSHServer. [Computer Software]. GitHub. https://github.com/ysc3839/FakeSSHServer. Date accessed: 13 December 2022.

[18] redhat. n.d. 22.5. vsftpd Configuration Options. https://web.mit.edu/rhel-doc/5/RHEL-5-manual/Deployment_Guide-en-US/s1-ftp-vsftpd-conf.html. Date accessed: 16 December 2022.

[19] byt3bl33d3r. (2015). Mad-Max Scapy: Improving Scapy's packet sending performance. GitHub. https://byt3bl33d3r.github.io/mad-max-scapy-improving-scapys-packet-sending-performance.html. Date accessed: 16 December 2022.

Appendix A

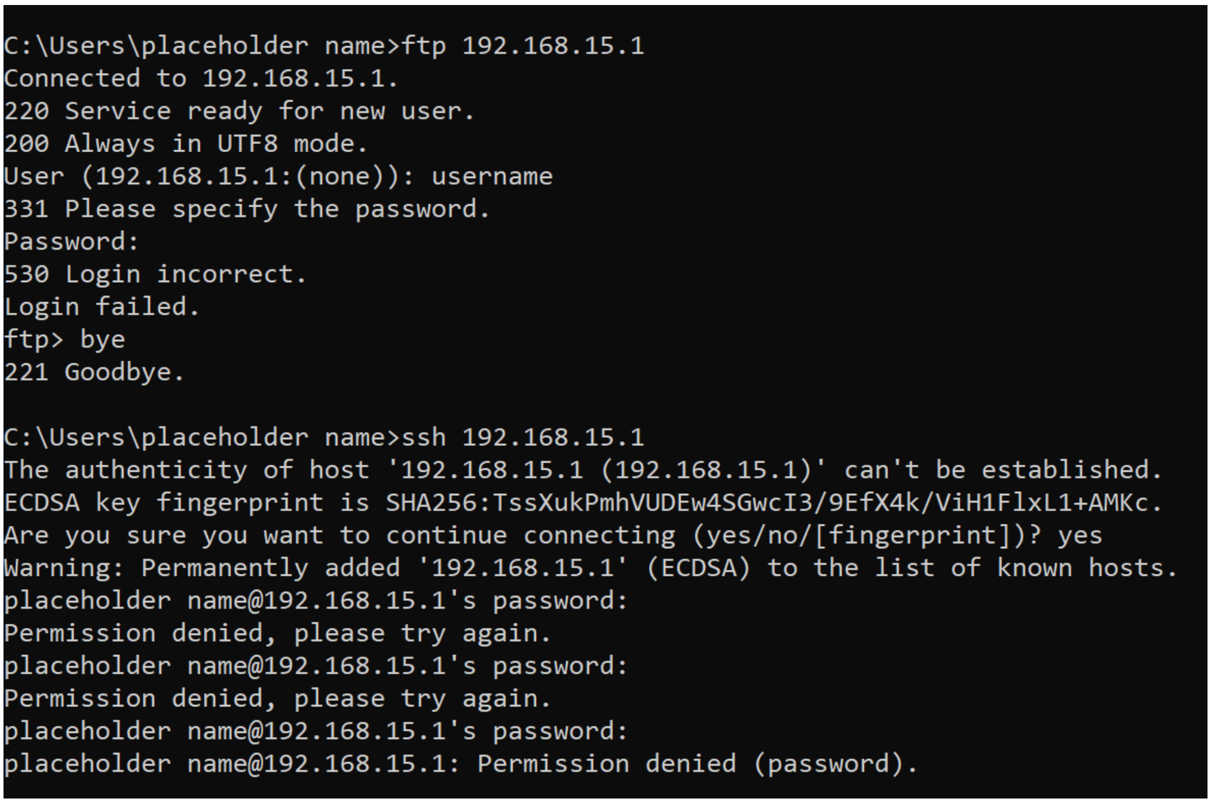


```
# Please follow the following format given as an example when configuring the honeypot below.
# For namespace protocols, the last namespace key cannot have value 'none'. Do assign the server
protocols to the namespaces in the order specified in host IDs.

host IDs = [1, 2, 4, 6]
network ID = 192.168.15
namespace protocols = {1:'ftp and ssh', 2:'ftp', 4:'none', 6:'ssh'}

```

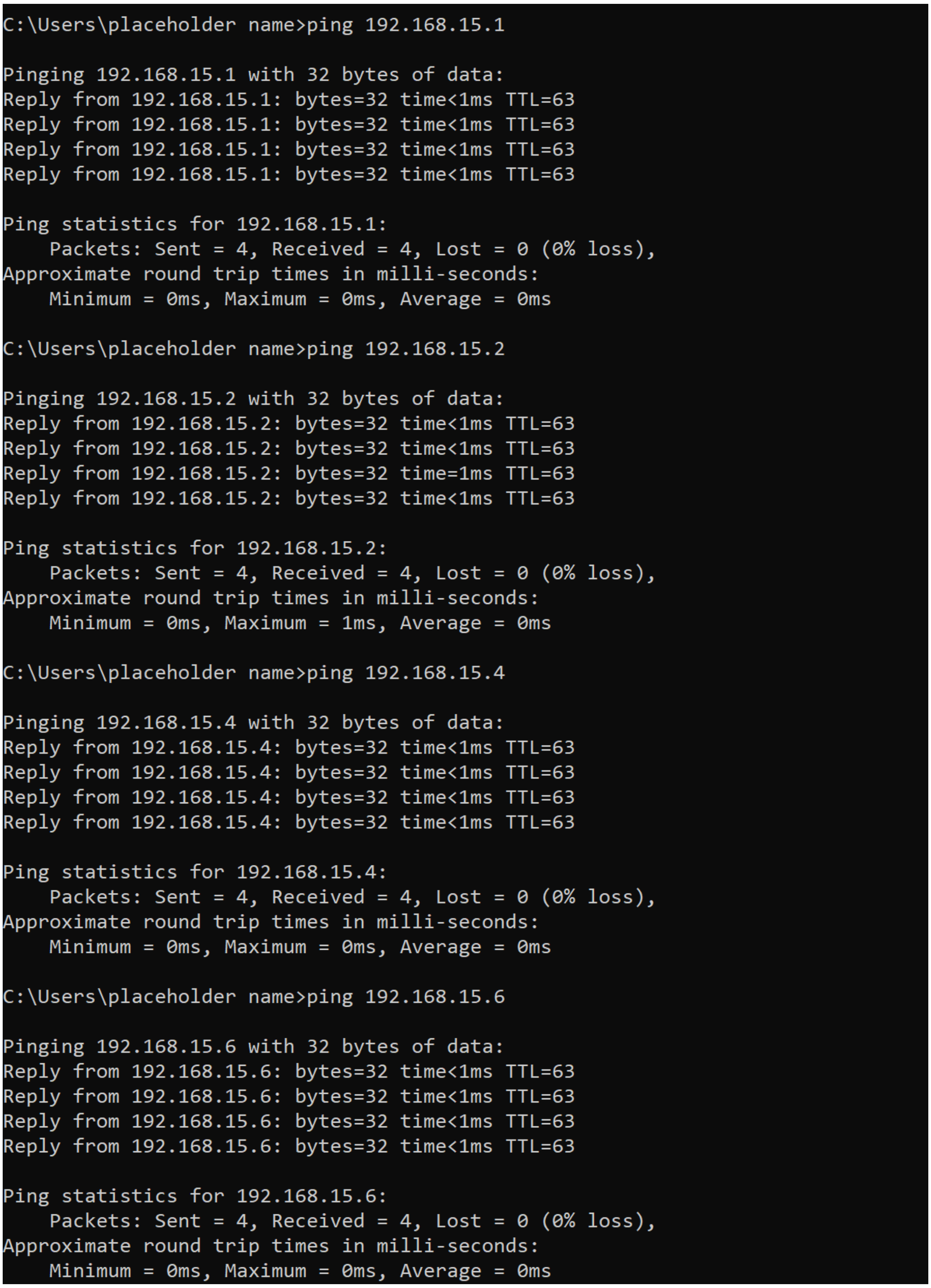
Appendix B



```
C:\Users\placeholder name>ftp 192.168.15.1
Connected to 192.168.15.1.
220 Service ready for new user.
200 Always in UTF8 mode.
User (192.168.15.1:(none)): username
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
ftp> bye
221 Goodbye.

C:\Users\placeholder name>ssh 192.168.15.1
The authenticity of host '192.168.15.1 (192.168.15.1)' can't be established.
ECDSA key fingerprint is SHA256:TssXukPmhVUDEw4SGwcI3/9EfX4k/ViH1FlxL1+AMKc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.15.1' (ECDSA) to the list of known hosts.
placeholder name@192.168.15.1's password:
Permission denied, please try again.
placeholder name@192.168.15.1's password:
Permission denied, please try again.
placeholder name@192.168.15.1's password:
placeholder name@192.168.15.1: Permission denied (password).
```

Appendix C



```
C:\Users\placeholder name>ping 192.168.15.1

Pinging 192.168.15.1 with 32 bytes of data:
Reply from 192.168.15.1: bytes=32 time<1ms TTL=63
Reply from 192.168.15.1: bytes=32 time<1ms TTL=63
Reply from 192.168.15.1: bytes=32 time<1ms TTL=63
Reply from 192.168.15.1: bytes=32 time<1ms TTL=63

Ping statistics for 192.168.15.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\placeholder name>ping 192.168.15.2

Pinging 192.168.15.2 with 32 bytes of data:
Reply from 192.168.15.2: bytes=32 time<1ms TTL=63
Reply from 192.168.15.2: bytes=32 time<1ms TTL=63
Reply from 192.168.15.2: bytes=32 time=1ms TTL=63
Reply from 192.168.15.2: bytes=32 time<1ms TTL=63

Ping statistics for 192.168.15.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Users\placeholder name>ping 192.168.15.4

Pinging 192.168.15.4 with 32 bytes of data:
Reply from 192.168.15.4: bytes=32 time<1ms TTL=63
Reply from 192.168.15.4: bytes=32 time<1ms TTL=63
Reply from 192.168.15.4: bytes=32 time<1ms TTL=63
Reply from 192.168.15.4: bytes=32 time<1ms TTL=63

Ping statistics for 192.168.15.4:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\placeholder name>ping 192.168.15.6

Pinging 192.168.15.6 with 32 bytes of data:
Reply from 192.168.15.6: bytes=32 time<1ms TTL=63
Reply from 192.168.15.6: bytes=32 time<1ms TTL=63
Reply from 192.168.15.6: bytes=32 time<1ms TTL=63
Reply from 192.168.15.6: bytes=32 time<1ms TTL=63

Ping statistics for 192.168.15.6:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

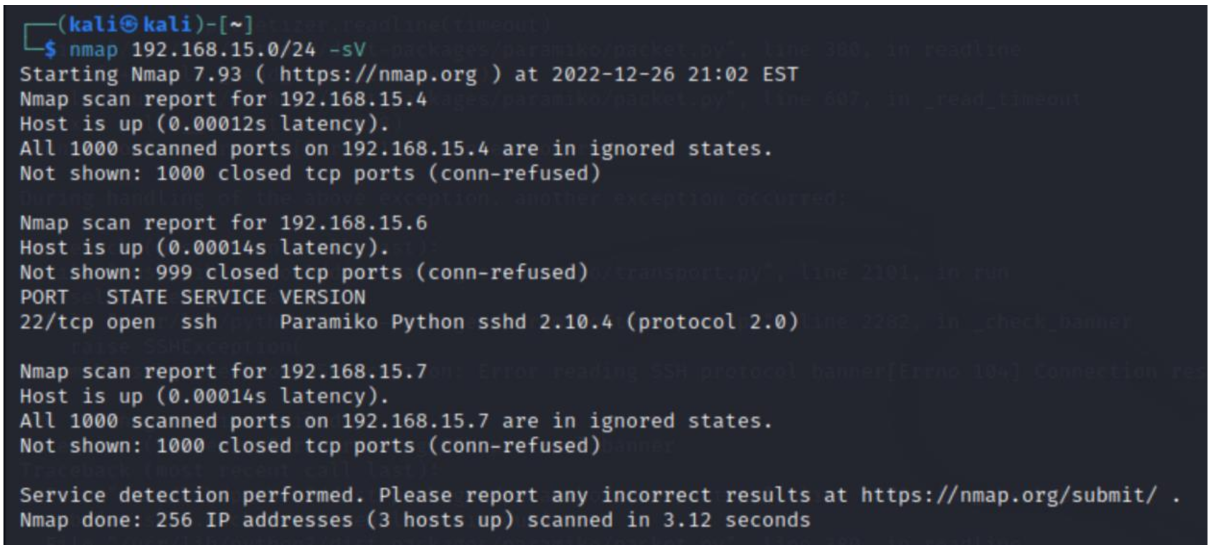
Appendix D

t_1/s	t_2/s	t_0/s
0.074759843	0.079432678	0.000044027
0.039856076	0.039918821	0.002313942
0.052834112	0.050268735	0.000030856
0.040283050	0.040641387	0.000084681
0.037144439	0.034666087	0.000077256
0.034436012	0.033571932	0.000247274
0.040502623	0.037276313	0.000076506
0.045547667	0.036962810	2.896050058
0.055824055	0.043706311	0.000065371
0.040506942	0.037899277	0.000178723
0.032445725	0.043648937	0.000162134
0.035589999	0.060133811	0.000031478

Appendix E

t_1/s	t_2/s	t_0/s
0.000225317	0.000305760	0.000058802
0.000341773	0.001197591	0.000018143
0.000155247	0.000363456	0.006594756
0.000280465	0.000904525	0.003133064
0.001479525	0.000469562	0.002114612
0.041389949	0.001560465	0.040463249
0.046187327	0.004326112	0.000045493
0.000297305	0.000220167	0.000204319
0.000398089	0.000187800	0.009406813
0.000468169	0.054243481	0.045231670
0.043924493	0.049813667	2.284267687
0.042061073	0.048364606	0.000072566
0.055067892	0.003690925	2.329394052
0.001516435	-	0.000072026
-	-	2.329585104

Appendix F



```
┌──(kali㉿kali)-[~]
└─$ nmap 192.168.15.0/24 -sV
Starting Nmap 7.93 ( https://nmap.org ) at 2022-12-26 21:02 EST
Nmap scan report for 192.168.15.4
Host is up (0.00012s latency).
All 1000 scanned ports on 192.168.15.4 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 192.168.15.6
Host is up (0.00014s latency).
Not shown: 999 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     Paramiko Python sshd 2.10.4 (protocol 2.0)

Nmap scan report for 192.168.15.7
Host is up (0.00014s latency).
All 1000 scanned ports on 192.168.15.7 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 256 IP addresses (3 hosts up) scanned in 3.12 seconds
```