

SSH HONEYPOT SIMULATION

A COURSE PROJECT REPORT

By

RUDRA GARAI (RA2111003010321)
SAPTAPARNO PATRA (RA2111003010311)
ARZOB SEN (RA2111003010303)

Under the guidance of

Dr. R.S Ponmagal

In partial fulfillment for the Course

Of

18CSC302J - COMPUTER NETWORKS

In C-TECH



FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND

TECHNOLOGY

Kattankulathur, Chengalpattu District

OCTOBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this mini project report "**SSH HONEYPOT
SIMULATION**" is the bonafide work of **RUDRA GARAI
(RA2111003010321) SAPTAPARNO PATRA (RA2111003010311)
ARZOB SEN (RA2111003010303)** who carried out the project work under
my supervision.

SIGNATURE

Dr. R.S. Ponmagal
Associate Professor
Department of Computing Technologies
SRM Institute of Science and Technology

TABLE OF CONTENTS

CHAPTERS	CONTENTS
1.	ABSTRACT
2.	INTRODUCTION
3.	REQUIREMENT ANALYSIS
4.	ARCHITECTURE AND DESIGN
5.	IMPLEMENTATION
6.	CODE
7.	EXPERIMENT RESULTS & ANALYSIS
8.	CONCLUSION
9.	REFERENCES
10.	RESULT

ACKNOWLEDGEMENT

We express our heartfelt thanks to our honourable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavours.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy**, for his encouragement

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal**, for bringing out novelty in all executions.

We would like to express my heartfelt thanks to **Chairperson, School of Computing Dr. Revathi Venkataraman**, for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professor Dr. Annapurani Panaiyappan, Professor and Head, Department of Networking and Communications and Course Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr. Ponmagal R S, Associate Professor, Department of Computing Technologies**, for his/her assistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. Pushpalatha M, Professor and Head, Department of Computing Technologies** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

1. ABSTRACT

A honeypot is an intentionally created fake system that is designed as a trap for potential attackers. They deviate the attack to the artificial system rather than the original system, and even it helps you detect the malicious traffic and track them. It appears as part of a network but is actually isolated and closely monitored because there is no reason for legitimate users to access a honeypot, any attempts to communicate with it are considered hostile. They can be categorized as production or research honeypots. We have made a research honeypot in the mini-project.

We have implemented an SSH honeypot in this project, which will act as a proxy server for any central server and be used to track the behavior of attacks that are done on any main servers. The function of a honeypot is to represent itself on the internet as a potential target for attackers (usually a server or other high-value asset) and to gather information and notify defenders of any attempts to access the honeypot by unauthorized users. While providing fake garbage data to the hackers to keep them engaged, the honeypot will asynchronously log all hacker activity in the logger and even provide convincing fake banners for trapping automated scanners like NMap and Nessus.

2. INTRODUCTION

How do honeypots work?

Generally, a honeypot operation consists of a computer, applications, and data that simulate the behaviour of a natural system that would be attractive to attackers, such as a financial system, internet of things (IoT) devices, or a public utility or transportation network. It appears as part of a network but is actually isolated and closely monitored. Because there is no reason for legitimate users to access a honeypot, any attempts to communicate with it are considered hostile. Honeypots may also be put outside the external firewall facing the internet to detect attempts to enter the internal network. The exact placement of the honeypot varies depending on how elaborate it is, the traffic it aims to attract, and how close it is to sensitive resources inside the corporate network. No matter the placement, it will always have some degree of isolation from the production environment. Virtual machines (VMs) are often used to host honeypots. That way, if they are compromised by malware, for example, the honeypot can be quickly restored. Two or more honeypots on a network form a honeynet, while a honey farm is a centralized collection of honeypots and analysis tools.

What are honeypots used for?

- Honeypots are used to capture information from unauthorized intruders that are tricked into accessing them because they appear to be a legitimate part of the network. Security teams deploy these traps as part of their network defense strategy. Honeypots are also used to research the behavior of cyber attackers and the ways they interact with networks.
- Spam traps are also similar to honeypots. They are email addresses or other network functions set up to attract spam web traffic. Spam traps are used in Project Honey Pot, a web-based network of honeypots embedded in website software. Its purpose is to harvest and collect the Internet Protocol (IP) addresses, email addresses, and related information on spammers so web administrators can minimize the amount of spam on their sites. The group's findings are used for research and law enforcement to combat unsolicited bulk mailing offenses.
- Honeypots aren't always used as a security measure. Anyone can use them for network surveillance, including hackers. For instance, a Wi-Fi Pineapple lets users create a Wi-Fi honeypot. Wi-Fi Pineapples are relatively cheap because consumer devices make a fake Wi-Fi network that mimics a real one in the vicinity. Unsuspecting individuals mistakenly connect to the artificial Wi-Fi network, and the honeypot operator can then monitor their traffic.

3. REQUIREMENTS

- Hardware Requirements:

- Processor: Minimum 2.4 GHz Clock Speed, 4-Core CPU
- RAM: Minimum 4GB DDR4 RAM
- Hard Disk: 500GB SATA Drive
- Network: 1000Mbps CAT6 Ethernet Cable

- Software Requirements:

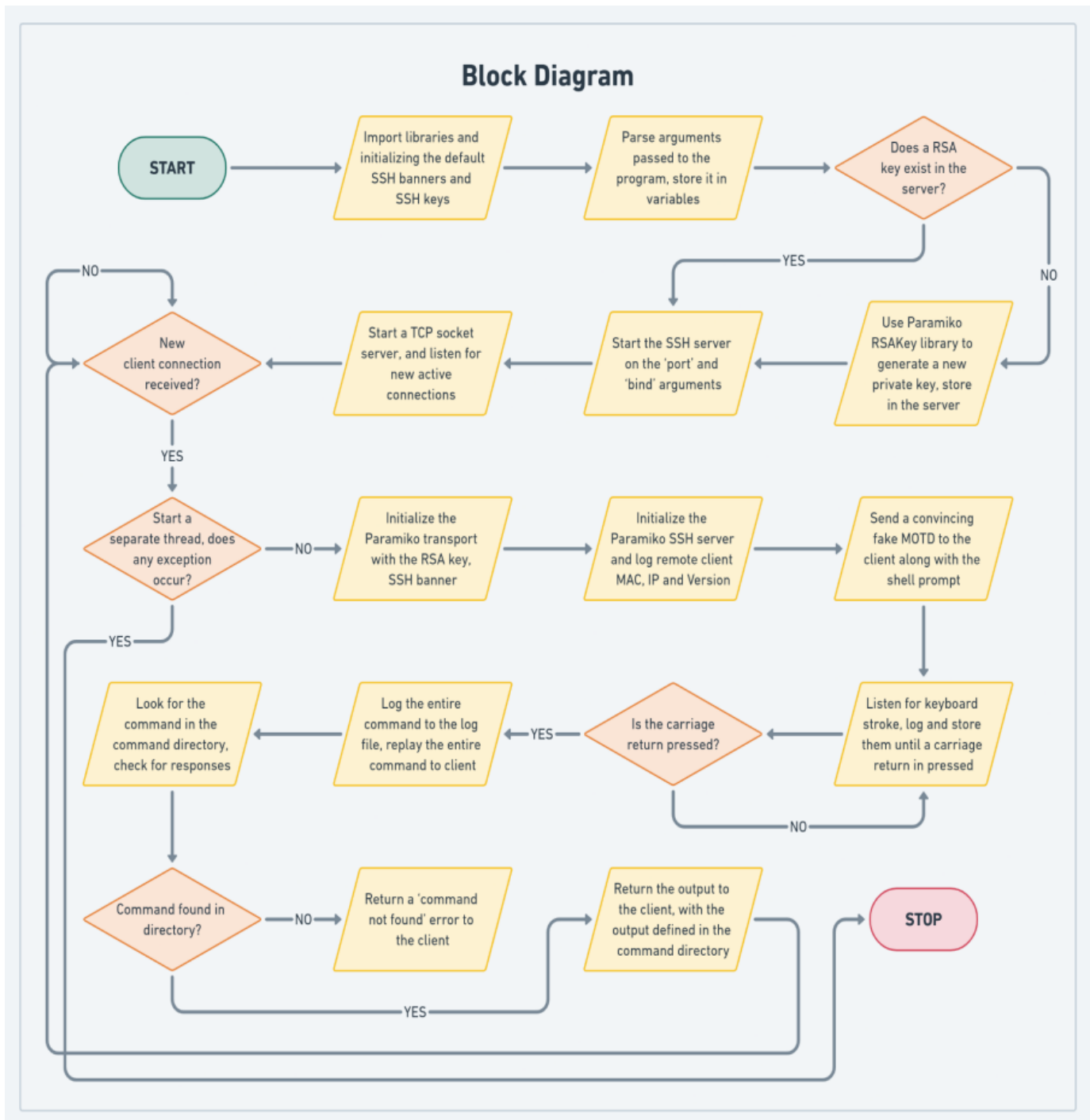
- OS: 64-bit Linux, Windows or Mac
- Software Installed:
 - Git
 - Docker
 - Python 3.8+
 - Visual Studio Code
 - OpenSSH

- Technology Stack:

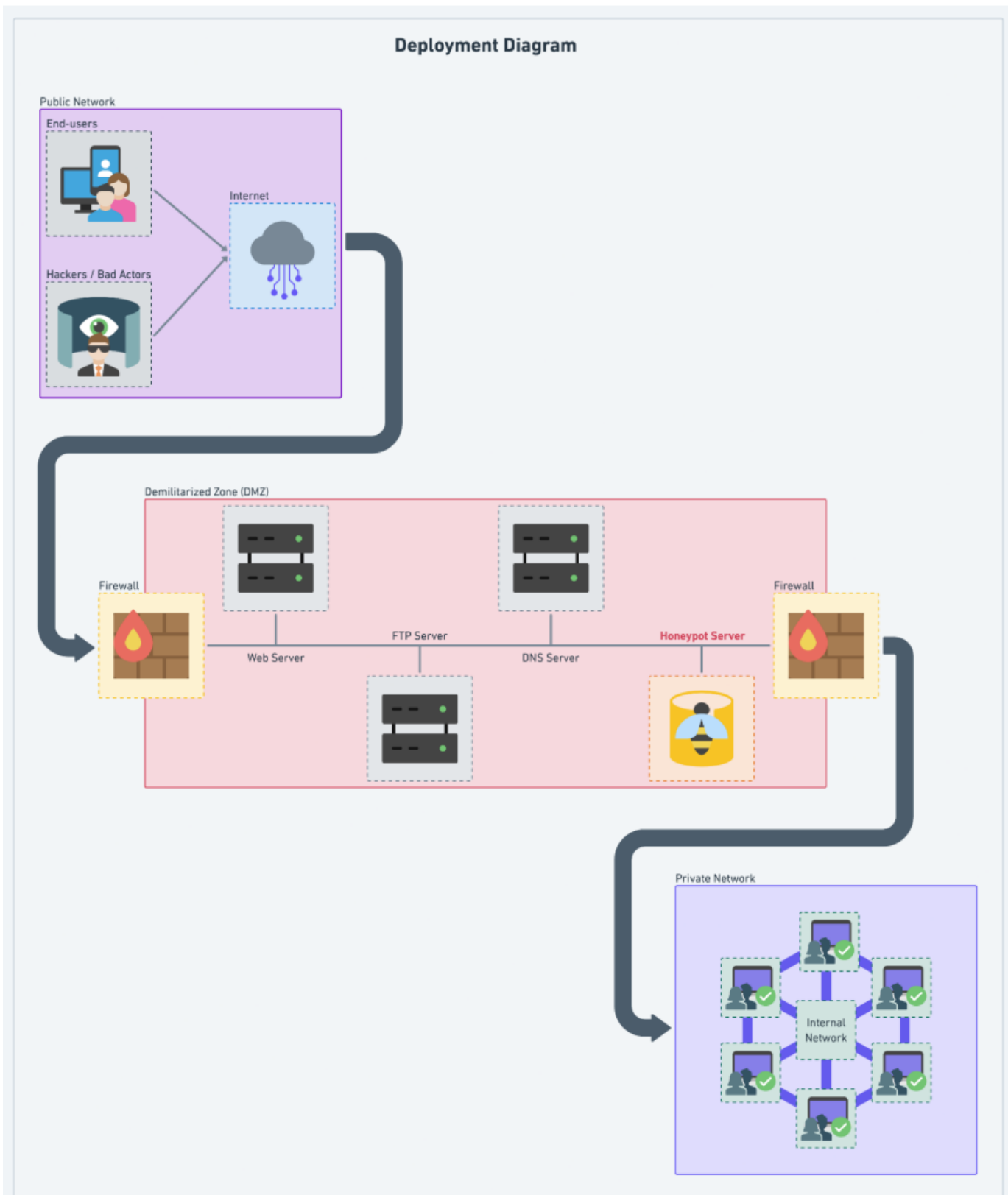
- Language of choice: Python 3.8
- Package Manager: Poetry
- SCM and DevOps Platform: GitHub
- Python Libraries Required:
 - Paramiko

4. ARCHITECTURE AND DESIGN

• BLOCK DIAGRAM



• DEPLOYMENT DIAGRAM



5. IMPLEMENTATION

Code Implementation: <https://github.com/techierudra/ssh-honeypot>

- assets**
 - Block-Diagram.png**
 - Demo-Video.mp4**
 - Deployment-Diagram-Vertical.png**
 - Log-Output.png**
 - Server-Output.png**
 - SSH-Output.png**
- cmd-directory.json**
- docker-compose.yml**
- Dockerfile**
- honeypot.py**
- LICENSE**
- poetry.lock**
- pyproject.toml**
- README.md**
- server.key**
- ssh-honeypot.log**

6. CODE

honeypot.py

```
#!/usr/bin/env python

# importing libraries
import argparse
import threading
import socket
import sys
from os.path import isfile
import traceback
import logging
import json
import paramiko
from paramiko.rsakey import RSAKey

# init SSH banner
SSH_BANNER = "SSH-2.0-OpenSSH_8.2p1\nUbuntu-Not-a-Honeypot-4ubuntu0.1"

HOST_KEY = None

# init arrow keys character sequence, to filter them out
UP_KEY = "\x1b[A".encode()
DOWN_KEY = "\x1b[B".encode()
RIGHT_KEY = "\x1b[C".encode()
LEFT_KEY = "\x1b[D".encode()
```

```

BACK_KEY = "\x7f".encode()

# init logger for logging all activity
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s -
%(message)s",
    level=logging.INFO,
    filename="ssh-honeypot.log",
)

def handle_cmd(cmd, chan, ip):
    """the function handling the different commands sent to the
SSH server"""
    response = ""
    cmd = cmd.strip()
    cmds = []
    if isfile("cmd-directory.json"):
        file = open("cmd-directory.json")
        cmds = json.load(file)["commands"]
    else:
        raise Exception("Command directory file not found!")
    if cmd in cmds:
        response = cmds[cmd]
    else:
        response = f"sh: 1: {cmd.split()[0]}: not found"

    if response != "":
        logging.info("Response from honeypot ({}): {}".format(ip,

```

```

response))
    response = response + "\r\n"
    chan.send(response)

class SshHoneypot(paramiko.ServerInterface):
    """the custom implementation of the Paramiko SSH server"""

    client_ip = None

    def __init__(self, client_ip):
        self.client_ip = client_ip
        self.event = threading.Event()

    def check_channel_request(self, kind, chanid):
        """determine if a channel request of a given type will
be granted, called in server when the client requests a channel,
after authentication is complete"""

        logging.info(
            "client called check_channel_request ({}):
{}".format(self.client_ip, kind)
        )
        if kind == "session":
            return paramiko.OPEN_SUCCEEDED

    def get_allowed_auths(self, username):
        """return a list of authentication methods supported by
the server"""

```

```

        logging.info(
            "client called get_allowed_auths ({{}}) with username
{{}}".format(
                self.client_ip, username
            )
        )
        return "password"

    def check_auth_password(self, username, password):
        """determine if a given username and password supplied
by the client is acceptable for use in authentication"""

        # Accept all passwords as valid by default
        logging.info(
            "new client credentials ({{}}): username: {{}},
password: {{}}".format(
                self.client_ip, username, password
            )
        )
        return paramiko.AUTH_SUCCESSFUL

    def check_channel_shell_request(self, channel):
        """determine if a shell will be provided to the client
on the given channel"""

        self.event.set()
        return True

```

```

def check_channel_pty_request(
    self, channel, term, width, height, pixelwidth,
    pixelheight, modes
):
    """determine if a pseudo-terminal of the given
    dimensions (usually requested for shell access) can be provided
    on the given channel"""

    return True

def check_channel_exec_request(self, channel, command):
    """determine if a shell command will be executed for the
    client"""

    command_text = str(command.decode("utf-8"))

    logging.info(
        "client sent command via check_channel_exec_request
    ({}): {}".format(
        self.client_ip, command_text
    )
    )
    return True

def handle_connection(client, addr):
    """the function handling the new client connections to the
    SSH server"""

```

```

client_ip = addr[0]
logging.info("New connection from: {}".format(client_ip))
print("New connection is here from: {}".format(client_ip))

try:
    transport = paramiko.Transport(client)
    transport.add_server_key(HOST_KEY)
    # changing banner to appear more convincing
    transport.local_version = SSH_BANNER
    server = SshHoneypot(client_ip)
    try:
        transport.start_server(server=server)

    except paramiko.SSHException:
        print("*** SSH negotiation failed.")
        raise Exception("SSH negotiation failed")

    # waiting 30 seconds for auth to complete
    chan = transport.accept(60)
    if chan is None:
        print("*** No channel (from " + client_ip + ").")
        raise Exception("No channel")

    chan.settimeout(None)

    if transport.remote_mac != "":
        logging.info("Client mac ({}): {}".format(client_ip,
transport.remote_mac))

```



```

if transport.remote_compression != "":
    logging.info(
        "Client compression ({}): {}".format(
            client_ip, transport.remote_compression
        )
    )

if transport.remote_version != "":
    logging.info(
        "Client SSH version ({}): {}".format(
            client_ip, transport.remote_version
        )
    )

if transport.remote_cipher != "":
    logging.info(
        "Client SSH cipher ({}): {}".format(client_ip,
transport.remote_cipher)
    )

server.event.wait(10)
if not server.event.is_set():
    logging.info("** Client ({}): never asked for a
shell".format(client_ip))
    raise Exception("No shell request")

try:
    # sending a convincing MOTD to the client
    chan.send(

```

```

        f"{' '*25}\n\rWelcome to Ubuntu 18.04.4 LTS
(GNU/Linux 4.15.0-128-generic x86_64)\n\rPlease rest assured you
are NOT in a Honeypot Server :)\n\r{' '*25}\r\n\r\n"
    )
    run = True
    while run:
        chan.send("$ ")
        command = ""
        while not command.endswith("\r"):
            transport = chan.recv(1024)
            print(client_ip + "- received:", transport)
            # echo input to pseudo-simulate a basic
terminal
            if (
                transport != UP_KEY
                and transport != DOWN_KEY
                and transport != LEFT_KEY
                and transport != RIGHT_KEY
                and transport != BACK_KEY
            ):
                chan.send(transport)
                command += transport.decode("utf-8")

        chan.send("\r\n")
        command = command.rstrip()
        logging.info("Command received ({}):
{}".format(client_ip, command))
        # handling commands to the SSH server
        if command == "exit":

```

```

        print(f"Connection closed (via exit command)
from: {client_ip}")
        logging.info(
            "Connection closed (via exit command): "
+ client_ip + "\n"
        )
        run = False

    else:
        handle_cmd(command, chan, client_ip)

except Exception as err:
    print("!!! Exception: {}: {}".format(err.__class__,
err))

    try:
        transport.close()
    except Exception:
        pass

    chan.close()

except Exception as err:
    print("!!! Exception: {}: {}".format(err.__class__,
err))

    try:
        transport.close()
    except Exception:
        pass

```

```

threads = []

def start_server(port, bind):
    """init and run the ssh server"""

    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1)
        sock.bind((bind, port))
    except Exception as err:
        print("*** Bind failed: {}".format(err))
        traceback.print_exc()
        sys.exit(1)

    # using multi-threading to handle parallel connections
    while True:
        try:
            sock.listen(100)
            print("Listening for connection on port {}".format(port))
            client, addr = sock.accept()
        except Exception as err:
            print("*** Listen/accept failed: {}".format(err))
            traceback.print_exc()
            new_thread = threading.Thread(target=handle_connection,
args=(client, addr))
            new_thread.daemon = True

```

```

        new_thread.start()
        threads.append(new_thread)

if __name__ == "__main__":
    # parse arguments passed to the executable
    parser = argparse.ArgumentParser(description="Run an SSH
HoneyPot Server")
    parser.add_argument(
        "--port",
        "-p",
        help="The port to bind the ssh server to (default:
2222)",
        default=2222,
        type=int,
        action="store",
    )
    parser.add_argument(
        "--bind",
        "-b",
        help="The address to bind the ssh server to (default:
0.0.0.0)",
        default="0.0.0.0",
        type=str,
        action="store",
    )
    args = parser.parse_args()
    # check for a server private key, if not generate a new one
    try:

```

```

    if isfile("server.key"):
        print("Server RSA key found")
        HOST_KEY = RSAKey(filename="server.key")
    else:
        print("No RSA key found, creating one")
        HOST_KEY = RSAKey.generate(bits=2048)

HOST_KEY.write_private_key_file(filename="server.key")
start_server(args.port, args.bind)
except KeyboardInterrupt as e:
    print("\r\nExiting program through keyboard interrupt.")
    sys.exit(0)

```

cmd-directory.json

```

{
  "commands": {
    "ls": "passwords.txt\n\rnot-a-honeypot.txt",
    "pwd": "/root",
    "whoami": "definitely-not-honeypot-root",
    "id": "uid=0(definitely-not-honeypot-root)
gid=0(definitely-not-honeypot-root)
groups=1000(user),0(definitely-not-honeypot-root),27(sudo)"
  }
}

```

Dockerfile

```
FROM python:3.8
```

```
# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install package manager
RUN curl -sSL
https://raw.githubusercontent.com/python-poetry/poetry/master/ge
t-poetry.py | python -
ENV PATH="${PATH}:/root/.poetry/bin"
RUN echo $PATH

# copy all files to the working directory
WORKDIR /usr/src/ssh-honeypot
COPY . .

# install packages through poetry
RUN poetry config virtualenvs.create false \
    && poetry install --no-interaction --no-ansi

# start the SSH honeypot server
EXPOSE 2222
CMD ["poetry", "run", "python", "honeypot.py"]
```

docker-compose.yml

```
version: "3.9"
services:
  web:
```

```
build: .
ports:
  - "2222:2222"
```

7. EXPERIMENT RESULTS & ANALYSIS

7.1. RESULTS

Execution Screenshots:

● Executing the honeypot server:

```

• Installing pycparser (2.20)
• Installing cffi (1.15.0)
• Installing six (1.16.0)
• Installing bcrypt (3.2.0)
• Installing cryptography (35.0.0)
• Installing pynacl (1.4.0)
• Installing paramiko (2.8.0)
Removing intermediate container 108f2ccdc18a
--> c320cf494090
Step 10/11 : EXPOSE 2222
--> Running in 7bf703f5f1f7
Removing intermediate container 7bf703f5f1f7
--> 6671b97fed03
Step 11/11 : CMD ["poetry", "run", "python", "honeypot.py"]
--> Running in 1c4bb97bef44
Removing intermediate container 1c4bb97bef44
--> ac7d07ed5596
Successfully built ac7d07ed5596
Successfully tagged ssh-honeypot_web:latest
Recreating ssh-honeypot_web_1 ... done
Attaching to ssh-honeypot_web_1
web_1 | Skipping virtualenv creation, as specified in config file.
web_1 | No RSA key found, creating one
web_1 | Listening for connection on port 2222 ...
web_1 | New connection is here from: 172.19.0.1
web_1 | Listening for connection on port 2222 ...
web_1 | !!! Exception: <class 'EOFError'>;
web_1 | Listening for connection on port 2222 ...
web_1 | New connection is here from: 172.19.0.1
web_1 | 172.19.0.1- received: b'q'
web_1 | 172.19.0.1- received: b'h'
web_1 | 172.19.0.1- received: b'o'
web_1 | 172.19.0.1- received: b'a'
web_1 | 172.19.0.1- received: b'm'
web_1 | 172.19.0.1- received: b'i'
web_1 | 172.19.0.1- received: b'\r'
web_1 | 172.19.0.1- received: b'l'
web_1 | 172.19.0.1- received: b's'
web_1 | 172.19.0.1- received: b'\r'
web_1 | 172.19.0.1- received: b'i'
web_1 | 172.19.0.1- received: b'd'
web_1 | 172.19.0.1- received: b'\r'
web_1 | 172.19.0.1- received: b'p'
web_1 | 172.19.0.1- received: b'w'
web_1 | 172.19.0.1- received: b'd'
web_1 | 172.19.0.1- received: b'\r'

```


Our main requirements for the honeypot are to have an effective SSH server implementation, including emulated commands and some way of logging the usernames, passwords, and other metadata we can gather. First, we need a way to log the attacks our honeypot receives. To keep things simple, we used Python's logging library. The code sets the logging library to save all logs to `ssh_honeypot.log`. We've selected the logging format to display helpful info such as the timestamp of when the event occurred, and then we'll provide messages to log later in the honeypot code.

For the basic SSH honeypot setup, we used Paramiko's `ServerInterface` class, implementing the following methods, where each process needs to return a specific response for the SSH server to work. This is also the ideal place to log things like authentication details. The method `check_auth_publickey` logs the client's public authentication key then returns `AUTH_PARTIALLY_SUCCESSFUL` (i.e., tells the client that a password is still required). The method `check_auth_password` logs the client's username, and password then returns `AUTH_SUCCESSFUL`.

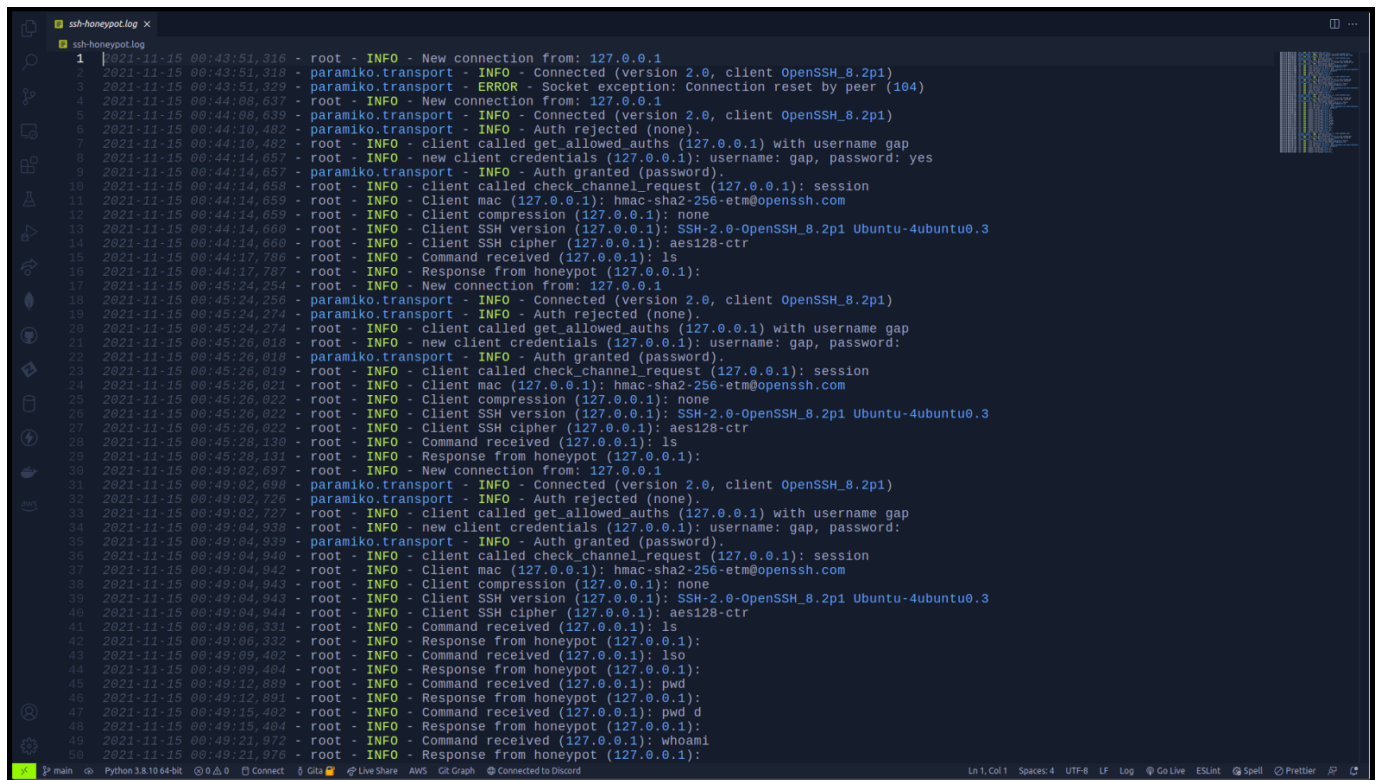
- SSH into the honeypot server:

```
ssh mockuser@localhost -p 2222
The authenticity of host '[localhost]:2222 ([127.0.0.1]:2222)' can't be established.
RSA key fingerprint is SHA256:0U9Z1cWZSDcmldMYf012HKA0d262y5H+R7qkjsvPIU.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2222' (RSA) to the list of known hosts.
mockuser@localhost's password:
*****
Welcome to Ubuntu 18.04.4 LTS (GNU/Linux 4.15.0-128-generic x86_64)
Please rest assured you are NOT in a Honeypot Server :)
*****

$ whoami
definitely-not-honey-pot-root
$ ls
passwords.txt
not-a-honey-pot.txt
$ id
uid=0(definitely-not-honey-pot-root) gid=0(definitely-not-honey-pot-root) groups=1000(user),0(definitely-not-honey-pot-root),27(sudo)
$ pwd
/root
$
```

An essential function of our SSH honeypot is to respond to commands. The code below implements the `ls` (list directory contents) and `pwd` (print working directory) commands. The honeypot returns `users.txt` to the user (i.e., that's the only file in the current directory). The command `pwd` returns `/home/root` as the current directory. To get the SSH honeypot server started, we used Python's `socket` library to open a port and bind it to our application. We also used Python's `threading` library to create a new thread to handle each connection. This allows our SSH honeypot server to handle multiple connections simultaneously.

- User activity being logged into the log file:



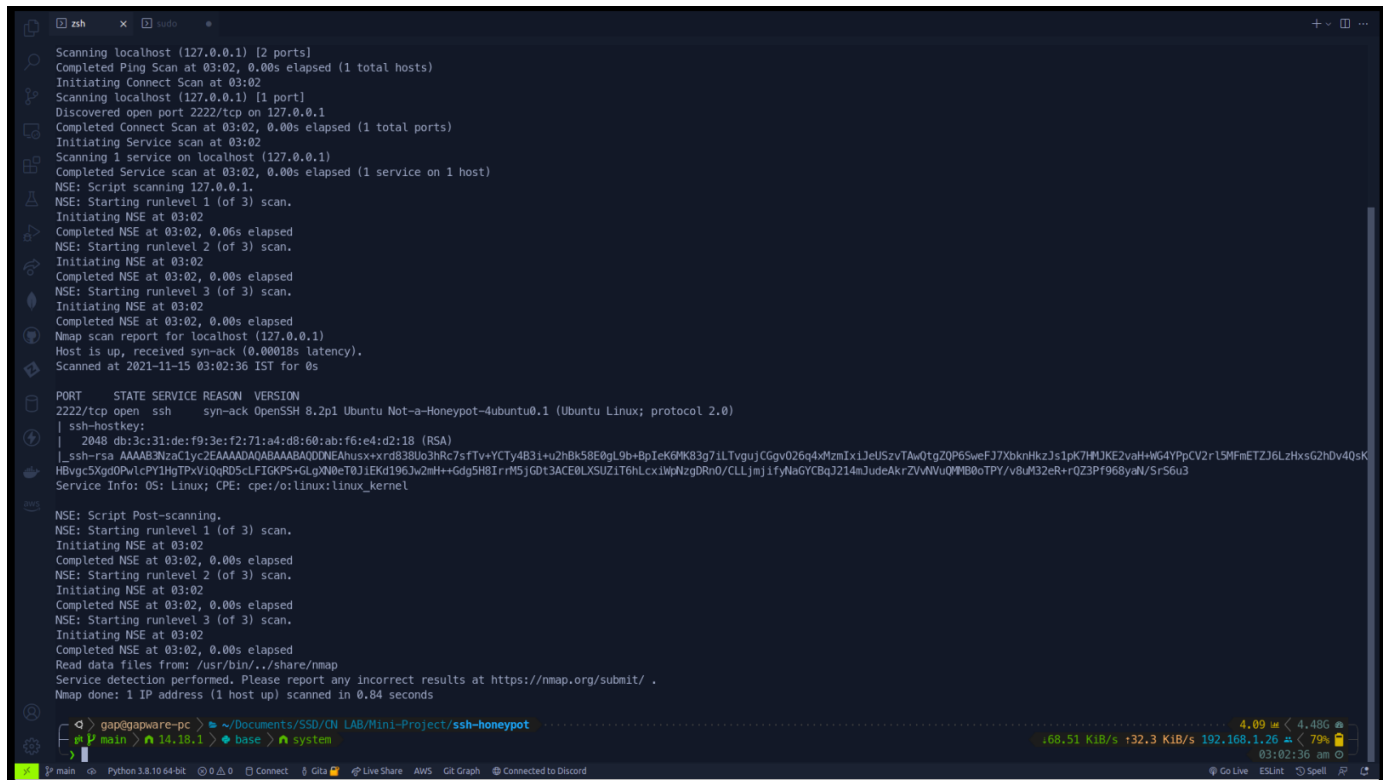
```
1 2021-11-15 09:43:51,316 - root - INFO - New connection from: 127.0.0.1
2 2021-11-15 09:43:51,318 - paramiko.transport - INFO - Connected (version 2.0, client OpenSSH_8.2p1)
3 2021-11-15 09:43:51,329 - paramiko.transport - ERROR - Socket exception: Connection reset by peer (104)
4 2021-11-15 09:44:09,637 - root - INFO - New connection from: 127.0.0.1
5 2021-11-15 09:44:09,639 - paramiko.transport - INFO - Connected (version 2.0, client OpenSSH_8.2p1)
6 2021-11-15 09:44:10,482 - paramiko.transport - INFO - Auth rejected (none).
7 2021-11-15 09:44:10,482 - root - INFO - client called get_allowed_auths (127.0.0.1): username gap
8 2021-11-15 09:44:14,657 - root - INFO - new client credentials (127.0.0.1): username: gap, password: yes
9 2021-11-15 09:44:14,657 - paramiko.transport - INFO - Auth granted (password).
10 2021-11-15 09:44:14,659 - root - INFO - client called check_channel_request (127.0.0.1): session
11 2021-11-15 09:44:14,659 - root - INFO - Client mac (127.0.0.1): hmac-sha2-256-etm@openssh.com
12 2021-11-15 09:44:14,659 - root - INFO - Client compression (127.0.0.1): none
13 2021-11-15 09:44:14,660 - root - INFO - Client SSH version (127.0.0.1): SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
14 2021-11-15 09:44:14,660 - root - INFO - Client SSH cipher (127.0.0.1): aes128-ctr
15 2021-11-15 09:44:17,786 - root - INFO - Command received (127.0.0.1): ls
16 2021-11-15 09:44:17,787 - root - INFO - Response from honeypot (127.0.0.1):
17 2021-11-15 09:45:24,254 - root - INFO - New connection from: 127.0.0.1
18 2021-11-15 09:45:24,256 - paramiko.transport - INFO - Connected (version 2.0, client OpenSSH_8.2p1)
19 2021-11-15 09:45:24,274 - paramiko.transport - INFO - Auth rejected (none).
20 2021-11-15 09:45:24,274 - root - INFO - client called get_allowed_auths (127.0.0.1): username gap
21 2021-11-15 09:45:26,018 - root - INFO - new client credentials (127.0.0.1): username: gap, password:
22 2021-11-15 09:45:26,018 - paramiko.transport - INFO - Auth granted (password).
23 2021-11-15 09:45:26,019 - root - INFO - client called check_channel_request (127.0.0.1): session
24 2021-11-15 09:45:26,021 - root - INFO - Client mac (127.0.0.1): hmac-sha2-256-etm@openssh.com
25 2021-11-15 09:45:26,022 - root - INFO - Client compression (127.0.0.1): none
26 2021-11-15 09:45:26,022 - root - INFO - Client SSH version (127.0.0.1): SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
27 2021-11-15 09:45:26,022 - root - INFO - Client SSH cipher (127.0.0.1): aes128-ctr
28 2021-11-15 09:45:28,139 - root - INFO - Command received (127.0.0.1): ls
29 2021-11-15 09:45:28,131 - root - INFO - Response from honeypot (127.0.0.1):
30 2021-11-15 09:49:02,697 - root - INFO - New connection from: 127.0.0.1
31 2021-11-15 09:49:02,698 - paramiko.transport - INFO - Connected (version 2.0, client OpenSSH_8.2p1)
32 2021-11-15 09:49:02,726 - paramiko.transport - INFO - Auth rejected (none).
33 2021-11-15 09:49:02,727 - root - INFO - client called get_allowed_auths (127.0.0.1): username gap
34 2021-11-15 09:49:04,938 - root - INFO - new client credentials (127.0.0.1): username: gap, password:
35 2021-11-15 09:49:04,939 - paramiko.transport - INFO - Auth granted (password).
36 2021-11-15 09:49:04,940 - root - INFO - client called check_channel_request (127.0.0.1): session
37 2021-11-15 09:49:04,942 - root - INFO - Client mac (127.0.0.1): hmac-sha2-256-etm@openssh.com
38 2021-11-15 09:49:04,943 - root - INFO - Client compression (127.0.0.1): none
39 2021-11-15 09:49:04,943 - root - INFO - Client SSH version (127.0.0.1): SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.3
40 2021-11-15 09:49:04,944 - root - INFO - Client SSH cipher (127.0.0.1): aes128-ctr
41 2021-11-15 09:49:06,331 - root - INFO - Command received (127.0.0.1): ls
42 2021-11-15 09:49:06,332 - root - INFO - Response from honeypot (127.0.0.1):
43 2021-11-15 09:49:09,402 - root - INFO - Command received (127.0.0.1): lso
44 2021-11-15 09:49:09,404 - root - INFO - Response from honeypot (127.0.0.1):
45 2021-11-15 09:49:12,889 - root - INFO - Command received (127.0.0.1): pwd
46 2021-11-15 09:49:12,891 - root - INFO - Response from honeypot (127.0.0.1):
47 2021-11-15 09:49:15,402 - root - INFO - Command received (127.0.0.1): pwd d
48 2021-11-15 09:49:15,404 - root - INFO - Response from honeypot (127.0.0.1):
49 2021-11-15 09:49:21,972 - root - INFO - Command received (127.0.0.1): whoami
50 2021-11-15 09:49:21,976 - root - INFO - Response from honeypot (127.0.0.1):
```

As an SSH server runs, it optionally produces log messages to describe what it is doing. Log messages aid the system administrator in tracking the server's behaviour and detecting and diagnosing problems. For example, if a server is mysteriously rejecting connections it should accept, one of the first places to seek the cause is the server's log output.

7.2. RESULT ANALYSIS

Based on the screenshots above, we can see that the SSH Honeypot server is successfully executed, providing garbage data value to any hacker who connects and tries to brute-force their unauthorized access. It even logs all the hacker activity to the log file, which can be later

analyzed by cyber forensics experts.



```
Scanning localhost (127.0.0.1) [2 ports]
Completed Ping Scan at 03:02, 0.00s elapsed (1 total hosts)
Initiating Connect Scan at 03:02
Scanning localhost (127.0.0.1) [1 port]
Discovered open port 2222/tcp on 127.0.0.1
Completed Connect Scan at 03:02, 0.00s elapsed (1 total ports)
Initiating Service scan at 03:02
Scanning 1 service on localhost (127.0.0.1)
Completed Service scan at 03:02, 0.00s elapsed (1 service on 1 host)
NSE: Script scanning 127.0.0.1.
NSE: Starting runlevel 1 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
NSE: Starting runlevel 2 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
NSE: Starting runlevel 3 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
Nmap scan report for localhost (127.0.0.1)
Host is up, received syn-ack (0.00018s latency).
Scanned at 2021-11-15 03:02:36 IST for 0s

PORT      STATE SERVICE REASON VERSION
2222/tcp  open  ssh      syn-ack  OpenSSH 8.2p1 Ubuntu Not-a-HoneyPot-4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|_ 2048 db3c31:de1f93e:f271a4:d8:ab:f6:e4:d2:18 (RSA)
|_ ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDIEAhussx+rrd838Uo3hRc7sFTv+YCTy4B31+u2hBk58E0g19b+BpIek6MK83g7iLTvqujCGv026q4xMzmIxLJeU5zvTAw0tgZOP6SweFJ7XbknHkzJslpK7HMDKE2vaHwG4YppCV2rLSMfMETZJ6LzHxsG2hDv4QsK
HBVg5Xqg0Pv1cPY1HgTPv1QqRD5cLFTGKPS+GLqN0eT0J1EKd1963w2mH++GdgSHBirrM5JG0t3ACE0LXSUZIT6hLcx1WpNzgDRn0/CLLj=j1fyNaGyCBqJ214mJudeAkrZVWVWuQMMB0oTPY/v8uM32eR+rQZ3Pf968yaN/SrS6u3
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

NSE: Script Post-scanning.
NSE: Starting runlevel 1 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
NSE: Starting runlevel 2 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
NSE: Starting runlevel 3 (of 3) scan.
Initiating NSE at 03:02
Completed NSE at 03:02, 0.00s elapsed
Read data files from: /usr/bin/./share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.84 seconds
```

In the above screenshot, it is also shown that the SSH HoneyPot server is also capable of providing fake SSH banners to any port scanners and automated reconnaissance tools. This keeps those tools engaged in trying to hack and discover this service, while the incident response team tries to locate where the automated scanning is coming from. Thus, the SSH server protects and safeguards actual production servers in the network from unauthorized attacks.

8. CONCLUSION & FUTURE WORK

The SSH Honeypot server implemented in this mini project serves as a prototype of a production-grade Honeypot. Thus, it lacks some features which we were not able to implement in the short time frame. For example, currently, we have implemented a hard-coded JSON file as our command directory which limits our SSH server responses which the server can provide back to the hacker. In the future, we would like to implement a dynamic command directory that can be vast in nature and be more convincing in responding to the hacker's commands. Next of all, the SSH Honeypot Server can also be extended to spoof more types of remote protocols like FTP, SMB and HTTP servers. This can provide a robust network protection system to organizations.

9. REFERENCES

1. <https://searchsecurity.techtarget.com/definition/honey-pot>
2. <https://www.rapid7.com/fundamentals/honeypots/>
3. <https://lwn.net/Articles/848291/>
4. <https://medium.com/@rockprofile/how-i-setup-my-ssh-honeypot-b34b2bd3fba9>
5. https://thesai.org/Downloads/Volume7No5/Paper_18-SSH_Honey_pot_Building_Deploying_and_Analysis.pdf
6. <https://medium.com/acmvit/ssh-honeypot-build-your-own-6f508d535672>
7. <https://youtu.be/gtk2qphHKmA>