# SSH HONEYPOT SIMULATION

A COURSE PROJECT REPORT

By

Rohan Kumar(RA2011030010050)

Rajya Vardhan(RA2011030010008)

Aniruddha Ghosh(RA2011030010038)

Under the guidance of

**Dr P. Visalakshi**

*In partial fulfilment for the Course*

of

**18CSC302J - COMPUTER NETWORKS**

in Department of Networking and Communication


**FACULTY OF ENGINEERING AND TECHNOLOGY SRM**

**INSTITUTE OF SCIENCE AND TECHNOLOGY**

**Kattankulathur, Chenpalpattu District**

NOVEMBER 2022

# SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

**(Under Section 3 of UGC Act, 1956)**

## BONAFIDE CERTIFICATE

Certified that this project report "SSH Honeypot Simulation" is the bonafide work of Students - Rohan Kumar(RA2011030010050), Rajya Vardhan (RA2011030010008), and Aniruddha Ghosh(RA2011030010038) who carried out the project work under my supervision.

**SIGNATURE**

**Dr P.Visalakshi**

**Faculty**

**Department of Networking and Communication**
SRM Institute of Science and Technology, Potheri, SRM Nagar, Kattankulathur, Tamil Nadu 603203

**SIGNATURE**

**Course Coordinator**

**Nwc Department**
SRM Institute of Science and Technology, Potheri, SRM Nagar, Kattankulathur, Tamil Nadu 603203

# 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 Dr.Annapurani Panaiyappan, Professor and Head, Department of NWC ,** and **Course Coordinator Dr.P.Visalakshi , Assistant Professor Networking and Communication** for their constant encouragement and support.

We are highly thankful for our Course project's Internal guide **Dr. P. Visalakshi , Assistant Professor ,Nwc Department** for **his/her** assistance, timely suggestion, and guidance throughout the duration of this course project.

We extend my gratitude to the **Student HOD NWC Department** 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.

# TABLE OF CONTENTS

# 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 categorised 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 behaviour 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?**

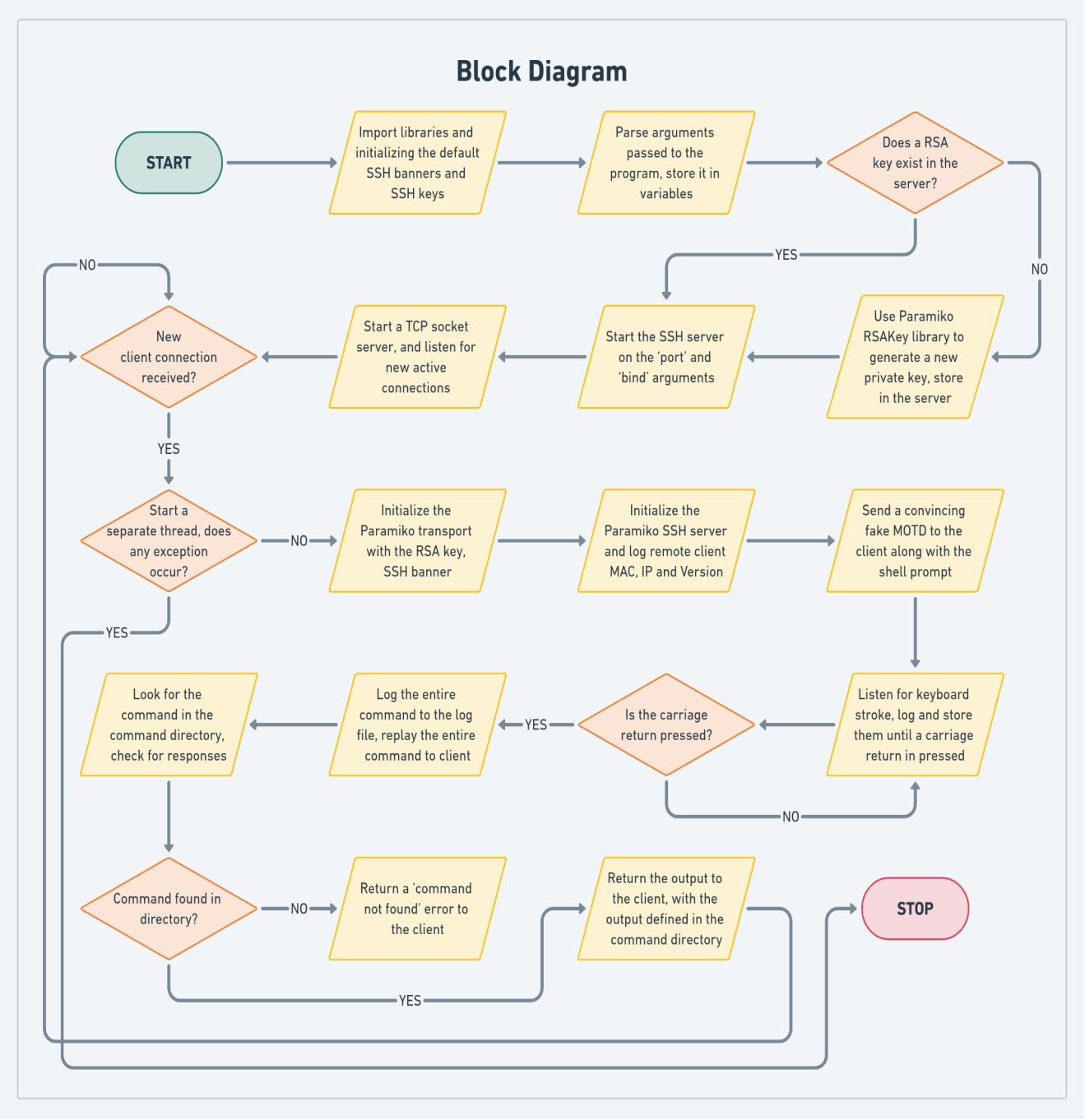
- Honeypots are used to capture information from unauthorized intruders that are tricked into accessing them because they appear to be a legitimate part of the network. Security teams deploy these traps as part of their network defence strategy. Honeypots are also used to research the behaviour of cyber 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 offences.
- 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. REQUIREMENT ANALYSIS

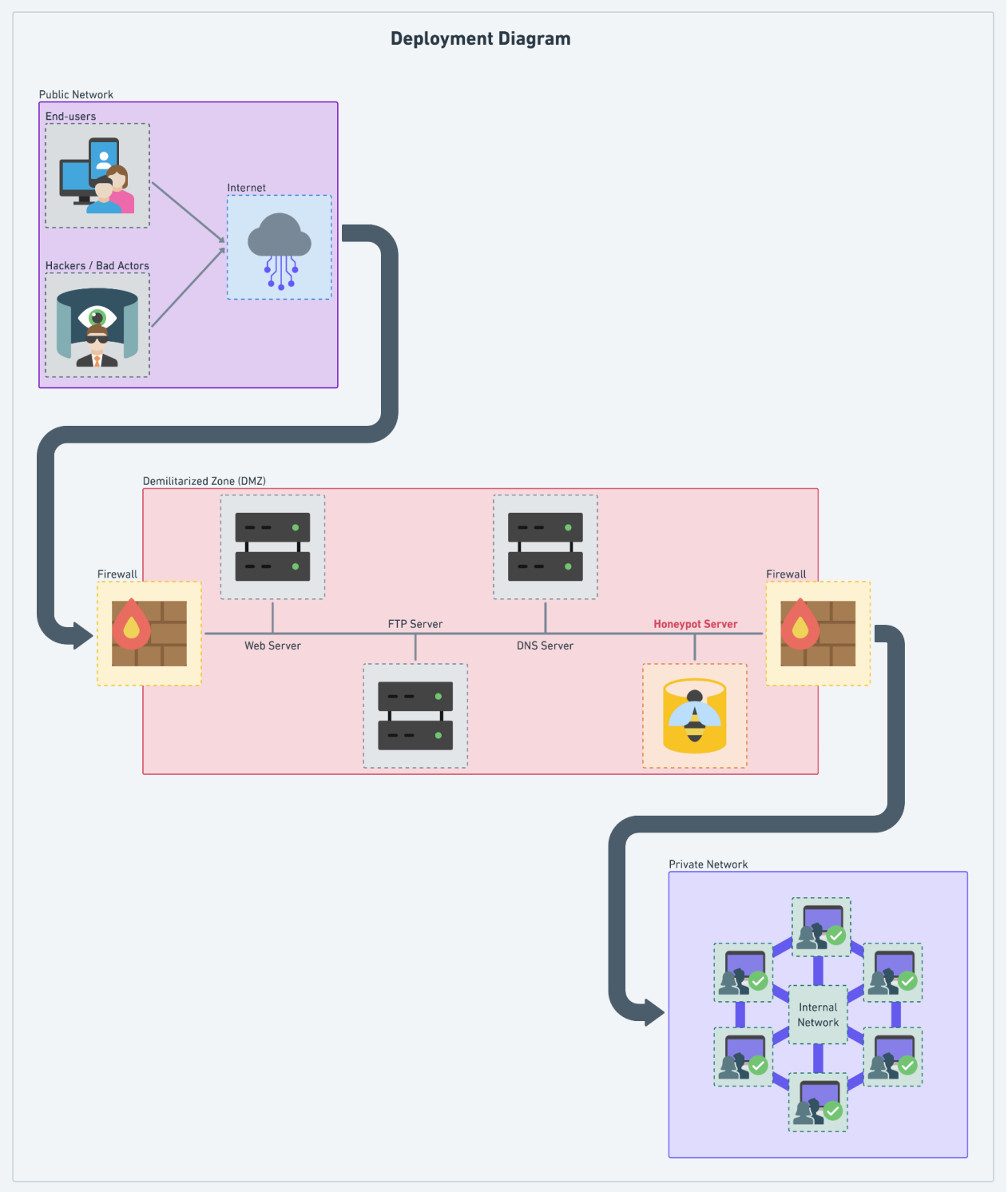
- **Hardware Requirements:**
    - **Processor:** Minimum 2.4 GHz Clock Speed, 4-Core CPU
    - **RAM:** Minimum 4GB DDR4 RAM
    - **Hard Disk:** 500GB SATA Drive
    - **Network:** 1000Mbps CAT6 Ethernet Cable
- **Software Requirements:**
    - **OS:** 64-bit Linux, Windows or Mac
    - **Software 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 & DESIGN

- **Block Diagram:**



- **Deployment Diagram:**



# 5. IMPLEMENTATION

**Code Implementation:**

**GitHub Repository Link: https://github.com/aghosh0605/ssh-honeypot**

**File Structure:**

```
├── 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
```

**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
Ubuntu-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/get-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**

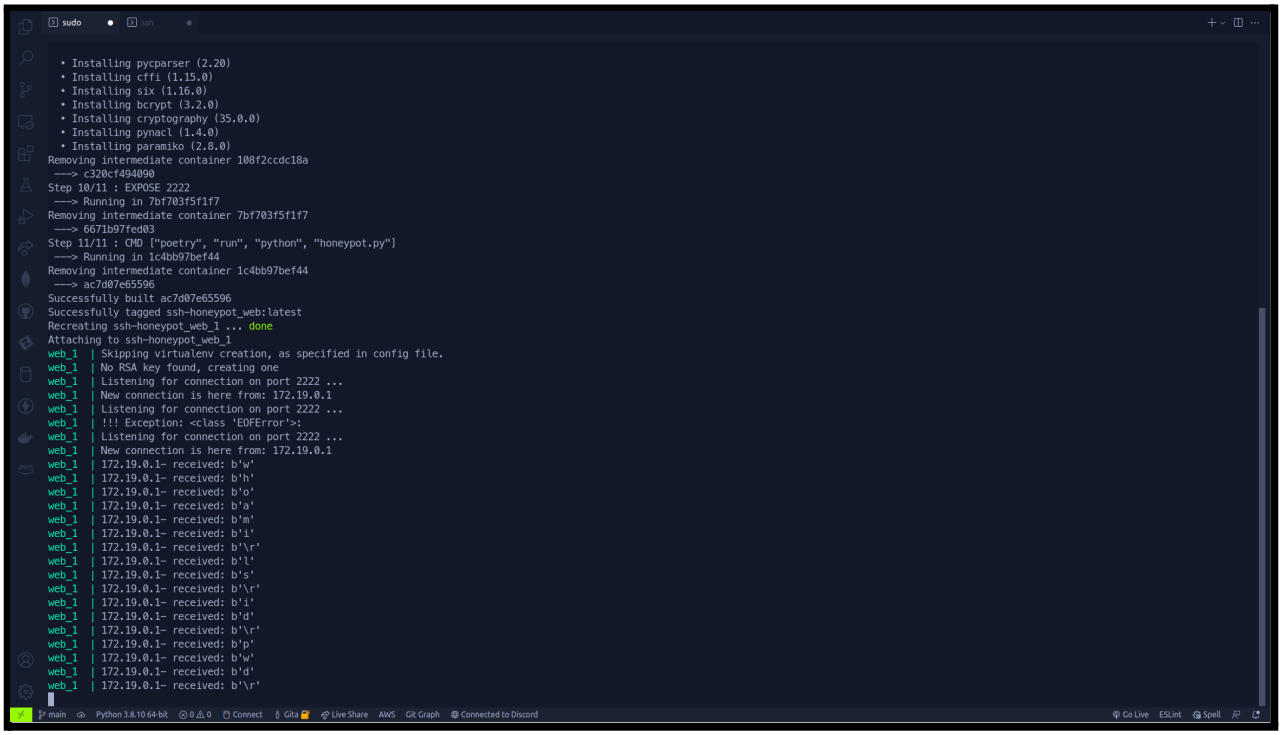
```
version: "3.9"
services:
 web:
   build: .
   ports:
     - "2222:2222"
```

# 6. EXPERIMENT RESULTS & ANALYSIS

## 6.1. RESULTS

**Execution Screenshots:**

- **Executing the honeypot server:**



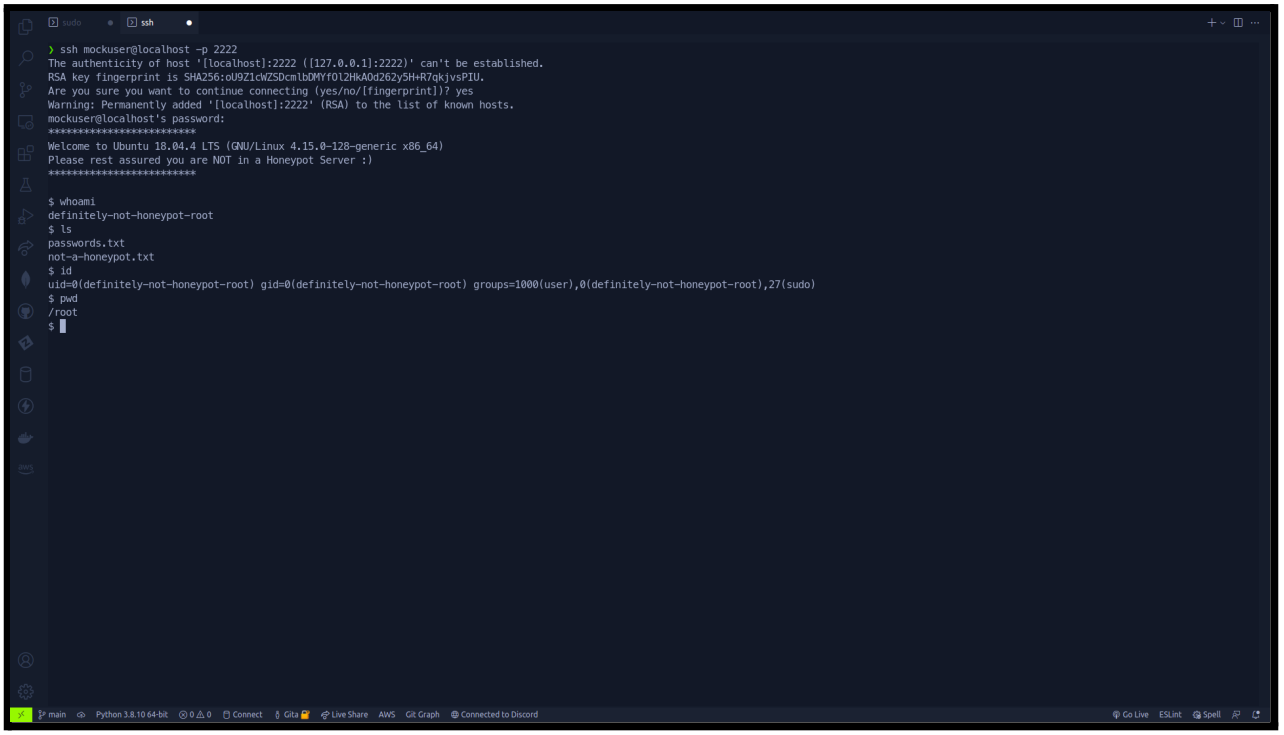

Our main requirements for the honeypot are to have an effective SSH server implementation, including emulated commands and some way of logging the usernames, passwords, and other metadata we can gather.

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:**

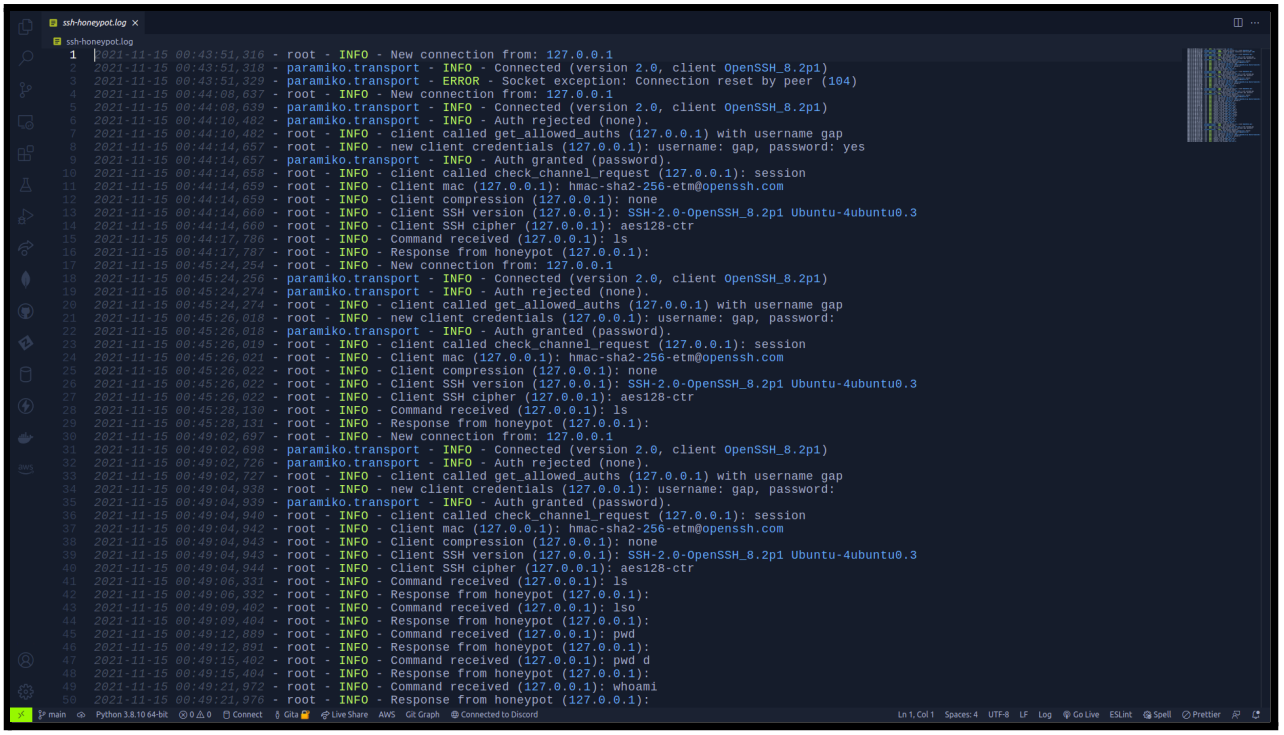



An essential function of our SSH honeypot is to respond to commands. The code below implements the ls (list directory contents) and pwd (print working directory) commands. The honeypot returns users.txt to the user (i.e., that's the only file in the

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.

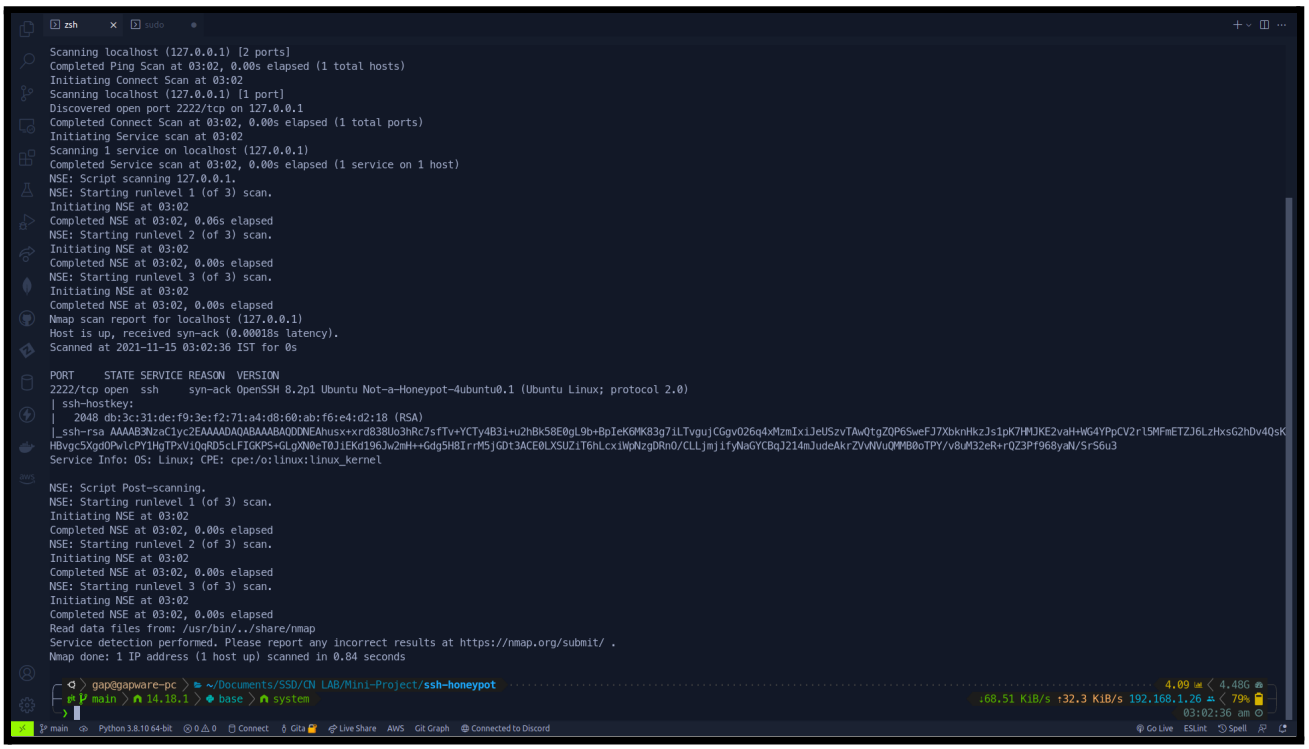
- **User activity being logged into the log file:**




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.

## 6.2. RESULT ANALYSIS

Based on the screenshots above, we can see that the SSH Honeypot server is successful 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.




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.

## 6.3. 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 organisations.

# 7. 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_Honeypot_Building_Deploying_and_Analysis.pdf
6. https://medium.com/acmvit/ssh-honeypot-build-your-own-6f508d535672
7. https://youtu.be/gtk2qphHKmA


----------X----------