
CYBER ADVERSARY PROFILING

USING SSH HONEYPOT SIMULATION FOR THREAT INTELLIGENCE

A COURSE PROJECT REPORT By

Ramya Manasa (RA2111027010011)

Akhila Angara (RA2111027010029)

Vyshnavi Nagella (RA2111027010034)

Snehal Sukundari (RA2111027010049)

Seyjuti Banerjee (RA2111027010052)

Shashi Kanikanti (RA2111027010053)

Under the Guidance of

Dr.R.Jayaraj

In partial fulfilment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE ENGINEERING

Specialization In Big Data Analytics



SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203 NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report "Cyber Adversary Profiling" is the bonafide work of Ramya Manasa (RA2111027010011), Akhila Angara (RA2111027010029), Vyshnavi Nagella (RA2111027010034), Snehal Sukundari (RA2111027010049), Seyjuti Banerjee (RA2111027010052) and Shashi Kumar Kanikanti (RA2111027010053) who carried out the project work under my supervision.

SIGNATURE

Dr.R.Jayaraj

Assistant Professor

Department of DSBS

SRMIST – KTR

SIGNATURE

Dr.M Lakshmi

Head of Department

Department of DSBS

SRMIST - KTR

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

CHAPTERS	CONTENTS
1.	ABSTRACT
2.	INTRODUCTION
3.	REQUIREMENT ANALYSIS
4.	ARCHITECTURE & DESIGN
5.	IMPLEMENTATION
6.	EXPERIMENT RESULTS & ANALYSIS
7.	CONCLUSION & FUTURE WORK
8.	REFERENCES

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.

Existing Problem: When a person has a proxy server, it can be tracked by few trackers.

For this SSH honeypot is stimulated, to track the behaviour of the attackers and their attacks. This will create a dummy server which will protect the actual proxy server.

And through this SSH honeypot we will be notified if anyone attempts by unauthorized users.

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

- **Software Requirements:**

- **OS:** 64-bit Linux, Windows or Mac ○

Software Installed:

- Git
- Docker
- Python 3.8+
- Visual Studio Code
- OpenSSH

- **Technology Stack:**

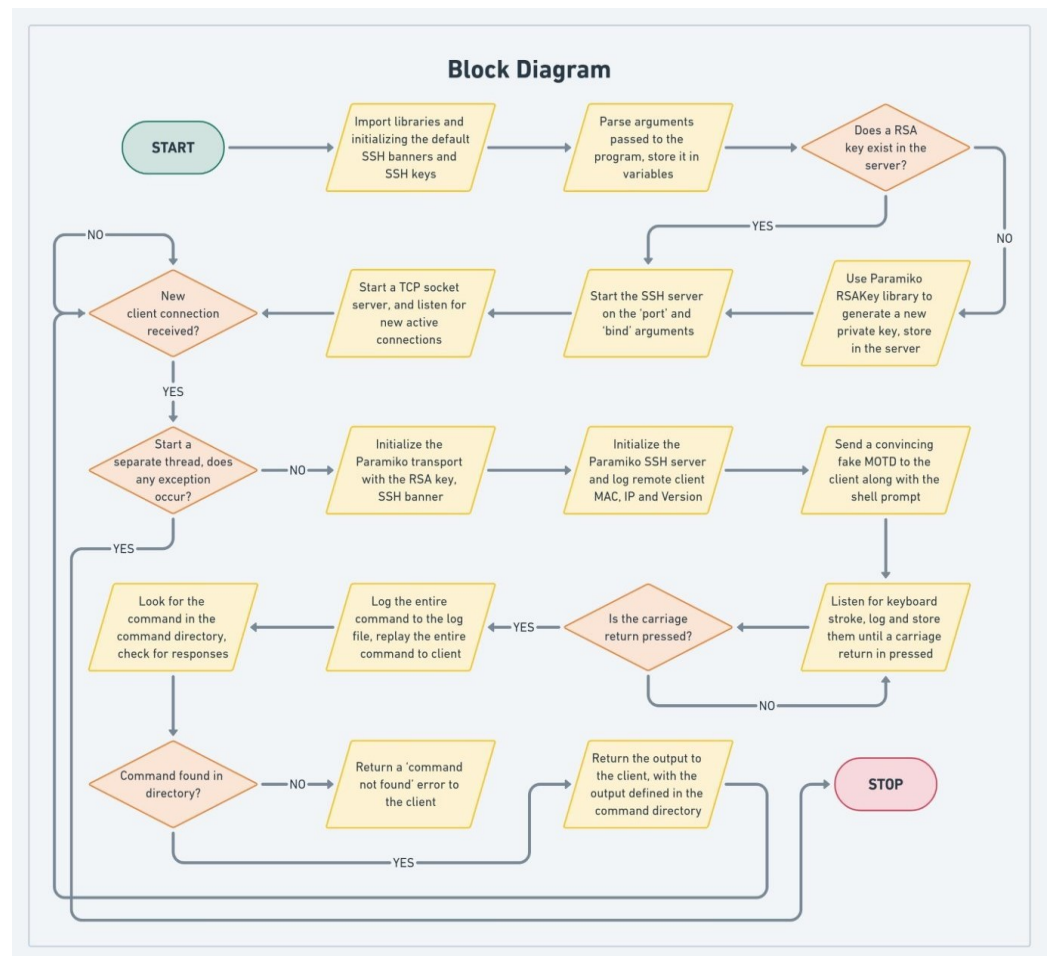
- **Language of choice:** Python 3.8
- **Package Manager:** Poetry
- **SCM and DevOps Platform:** GitHub ○ **Python**

Libraries Required:

- Paramiko

4. ARCHITECTURE & DESIGN

● Block Diagram



5. IMPLEMENTATION

Code Implementation:

```
import paramiko

import socket import

threading import

logging import os

# Configure logging

logging.basicConfig(filename='honeypot.log', level=logging.INFO,
format='%(asctime)s - %(levelname)s - %(message)s')

logger = logging.getLogger(__name__) class

FakeSSHServer(paramiko.ServerInterface):    def

    check_auth_password(self, username, password):

        logger.info(f'[*] Attracting attacker - Username: {username}, Password:
{password}')

        return paramiko.AUTH_FAILED

    def check_channel_request(self, kind, chanid):
```

```
logger.info(f'[*] Channel request - Kind: {kind}, Channel ID: {chanid}')

return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

def handle_connection(client, address, welcome_banner="Welcome to the
honeypot!"):

    try:

        transport = paramiko.Transport(client)

        transport.add_server_key(paramiko.RSAKey(filename='test_rsa.key'))

        server = FakeSSHServer()

        transport.start_server(server=server)

    # Wait for authentication attempts

    channel = transport.accept(10)

    if channel is not None:

        logger.info(f'[*] Connection accepted - Address: {address}')

        channel.send(welcome_banner + '\r\n')

    else:

        logger.warning("[-] No channel accepted within the timeout")
```

```
except paramiko.AuthenticationException as auth_error:
```

```
    logger.warning(f"[-] Authentication failed - Address: {address}, Error: {auth_error}")
```

```
except Exception as e:
```

```
    logger.error(f"[-] Error handling connection - Address: {address}, Error: {e}")
```

```
finally:
```

```
    client.close()
```

```
def start_honeypot(num_threads=5, welcome_banner="Welcome to the SSH
```

```
Honeypot!"):    server_socket = socket.socket(socket.AF_INET,
```

```
socket.SOCK_STREAM)
```

```
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
    server_socket.bind(('0.0.0.0', 2222))
```

```
    server_socket.listen(10)
```

```
    logger.info("[*] SSH Honeypot is actively attracting attackers on port 2222")
```

```
while True:
```

```
    try:
```

```
client, address = server_socket.accept()

logger.info(f"[*] Accepted connection from {address[0]}:{address[1]}")

# Handle the connection in a separate thread        client_handler =

threading.Thread(target=handle_connection, args=(client,

address, welcome_banner))

client_handler.start()

# Limit the number of concurrent threads

if threading.active_count() > num_threads:        logger.warning("[-\n\n] Maximum number of threads reached. Waiting for threads to finish.")

for thread in threading.enumerate():

    if thread != threading.current_thread():

        thread.join()

except KeyboardInterrupt:

    logger.info("[*] Honeypot shutting down.")

    break
```

except Exception as e:

logger.error(f'[-] Error accepting connection - Error: {e}') if

name == '__main__':

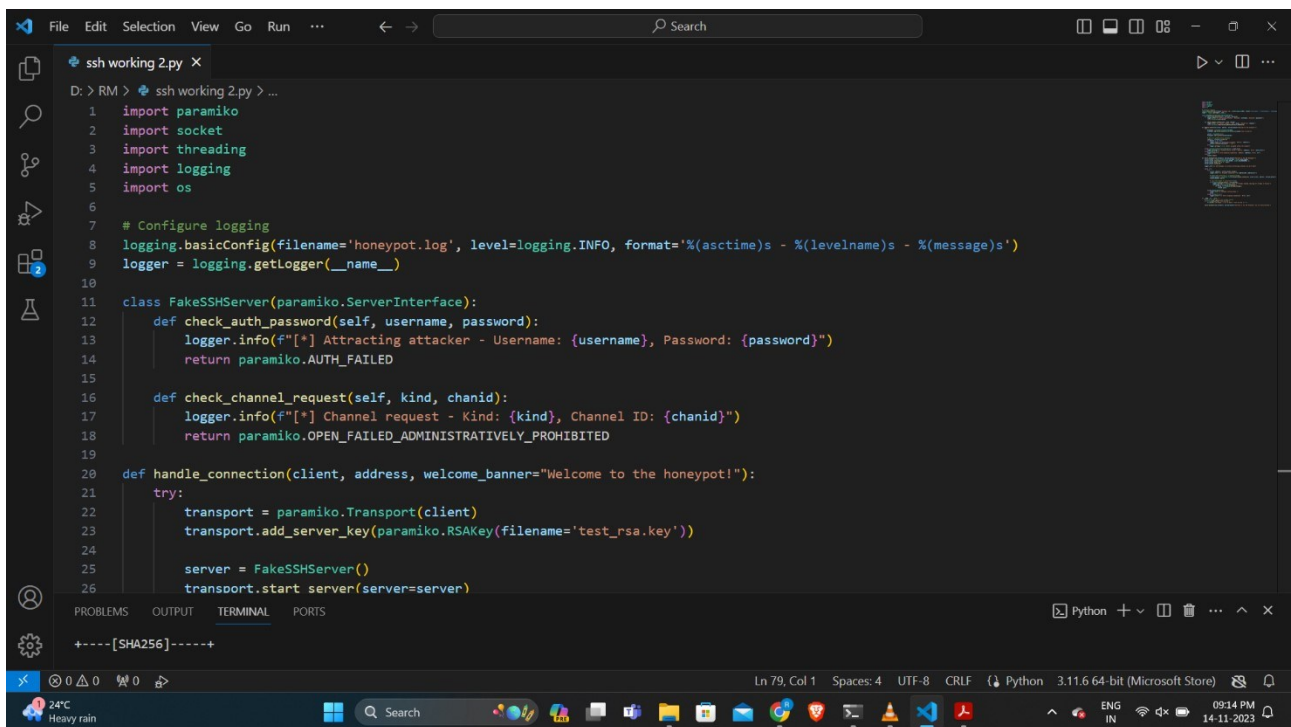
Set up the RSA key if not already present

if not os.path.isfile('test_rsa.key'):

os.system('ssh-keygen -t rsa -b 2048 -f test_rsa.key -N ""')

start_honeypot(num_threads=5, welcome_banner="Welcome to the SSH

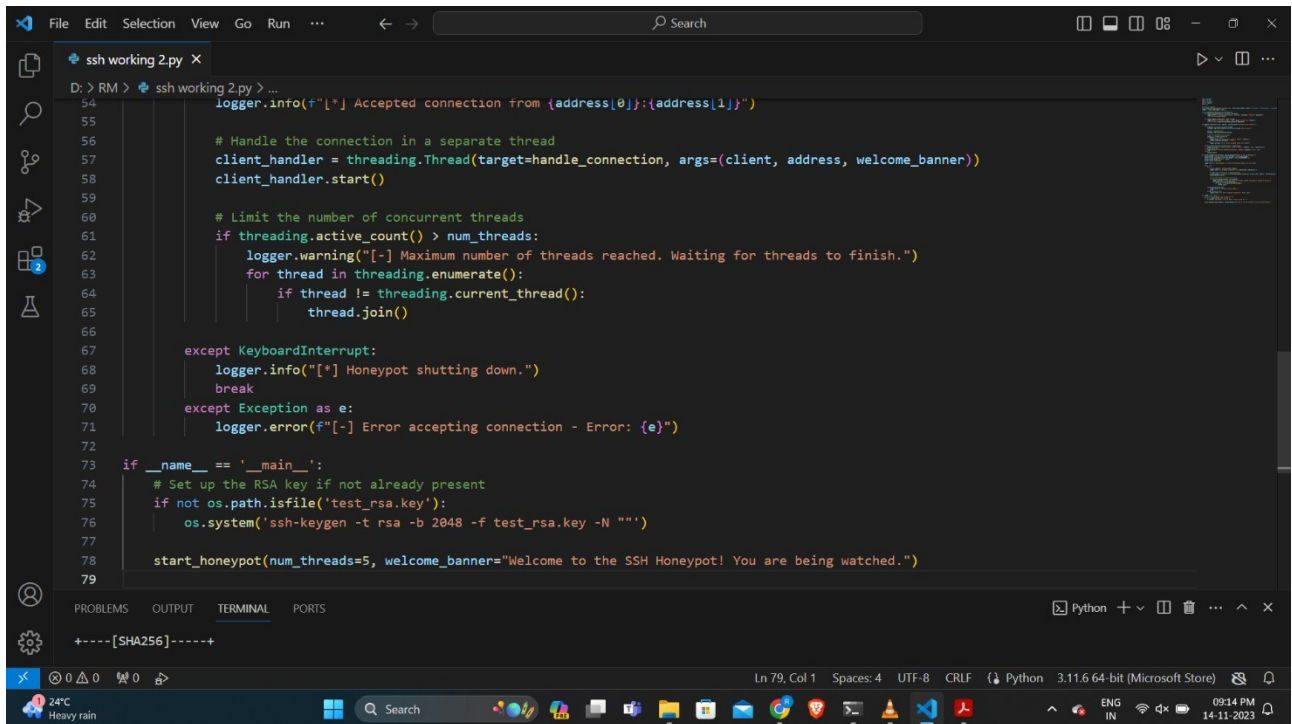
Honeypot! You are being watched.")



```
File Edit Selection View Go Run ... Search
ssh_working_2.py X
D:\> RM > ssh_working_2.py > ...
1 import paramiko
2 import socket
3 import threading
4 import logging
5 import os
6
7 # Configure logging
8 logging.basicConfig(filename='honeypot.log', level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
9 logger = logging.getLogger(__name__)
10
11 class FakeSSHServer(paramiko.ServerInterface):
12     def check_auth_password(self, username, password):
13         logger.info(f"[*] Attracting attacker - Username: {username}, Password: {password}")
14         return paramiko.AUTH_FAILED
15
16     def check_channel_request(self, kind, chanid):
17         logger.info(f"[*] Channel request - Kind: {kind}, Channel ID: {chanid}")
18         return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED
19
20     def handle_connection(client, address, welcome_banner="Welcome to the honeypot!"):
21         try:
22             transport = paramiko.Transport(client)
23             transport.add_server_key(paramiko.RSAKey(filename='test_rsa.key'))
24
25             server = FakeSSHServer()
26             transport.start_server(server=server)
```

The image shows a Windows 11 desktop with a dark theme. The primary focus is the Visual Studio Code (VS Code) editor, which is open to a file named 'ssh_working_2.py'. The script is a Python program designed to act as an SSH honeypot. It uses the 'paramiko' library for SSH handling and 'socket' for creating a listening server. The script includes a 'start_honeypot' function that sets up a server on port 2222, logs connection attempts, and handles authentication failures. The main execution part of the script runs a 'while True' loop to accept incoming connections. The VS Code interface includes a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The bottom of the editor shows a 'TERMINAL' tab with a shell prompt '+----[SHA256]-----+'. The Windows taskbar at the bottom displays the system clock as 09:14 PM on 14-11-2023, along with various background applications and system icons like network, volume, and battery.

```
File Edit Selection View Go Run ...  
ssh working 2.py X  
D:\> RM > ssh working 2.py > ...  
51 while True:  
52     try:  
53         client, address = server_socket.accept()  
54         logger.info(f"[*] Accepted connection from {address[0]}:{address[1]}")  
55  
56         # Handle the connection in a separate thread  
57         client_handler = threading.Thread(target=handle_connection, args=(client, address, welcome_banner))  
58         client_handler.start()  
59  
60         # Limit the number of concurrent threads  
61         if threading.active_count() > num_threads:  
62             logger.warning("[-] Maximum number of threads reached. Waiting for threads to finish.")  
63             for thread in threading.enumerate():  
64                 if thread != threading.current_thread():  
65                     thread.join()  
66  
67         except KeyboardInterrupt:  
68             logger.info("[*] Honeypot shutting down.")  
69             break  
70         except Exception as e:  
71             logger.error(f"[-] Error accepting connection - Error: {e}")  
72  
73 if __name__ == '__main__':  
74     # Set up the RSA key if not already present  
75     if not os.path.isfile('test_rsa.key'):  
76         os.system('ssh-keygen -t rsa -b 2048 -f test_rsa.key -N ""')  
PROBLEMS OUTPUT TERMINAL PORTS  
+----[SHA256]-----+  
Ln 79, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.6 64-bit (Microsoft Store) 09:14 PM 14-11-2023
```



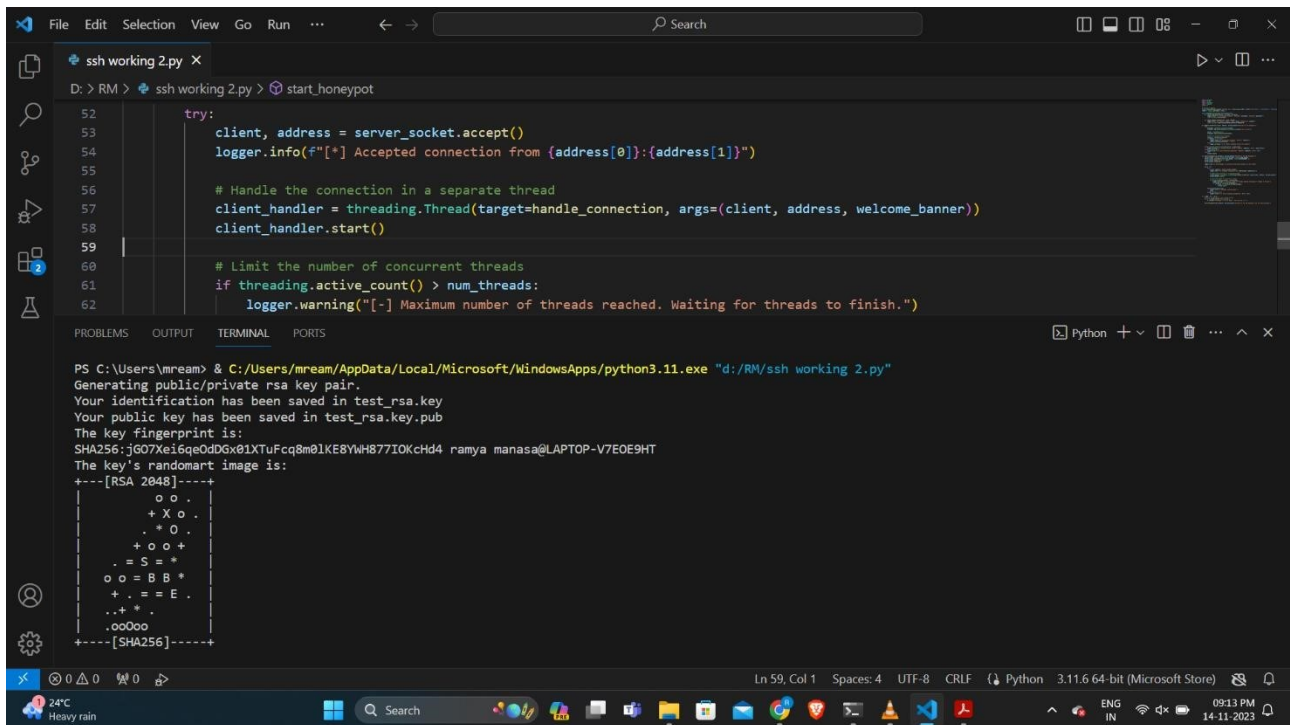
```
File Edit Selection View Go Run ... Search
ssh working 2.py X
D:\> RM > ssh working 2.py > ...
54 logger.info(f"[*] Accepted connection from {address[0]}:{address[1]}")
55
56 # Handle the connection in a separate thread
57 client_handler = threading.Thread(target=handle_connection, args=(client, address, welcome_banner))
58 client_handler.start()
59
60 # Limit the number of concurrent threads
61 if threading.active_count() > num_threads:
62     logger.warning(f"[-] Maximum number of threads reached. Waiting for threads to finish.")
63     for thread in threading.enumerate():
64         if thread != threading.current_thread():
65             thread.join()
66
67 except KeyboardInterrupt:
68     logger.info("[*] Honeypot shutting down.")
69     break
70 except Exception as e:
71     logger.error(f"[-] Error accepting connection - Error: {e}")
72
73 if __name__ == '__main__':
74     # Set up the RSA key if not already present
75     if not os.path.isfile('test_rsa.key'):
76         os.system('ssh-keygen -t rsa -b 2048 -f test_rsa.key -N ""')
77
78 start_honeypot(num_threads=5, welcome_banner="Welcome to the SSH Honeypot! You are being watched.")
79
PROBLEMS OUTPUT TERMINAL PORTS
+-----[SHA256]-----+
Ln 79, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.6 64-bit (Microsoft Store)
24°C Heavy rain Search 09:14 PM 14-11-2023
```

6. EXPERIMENT RESULTS & ANALYSIS

6.1. RESULTS Execution

Screenshots:

- Executing the honeypot server:



The screenshot shows a Visual Studio Code editor with a Python file named 'ssh_working_2.py'. The code implements an SSH server using the paramiko library. It sets up a server socket, listens for connections, and handles them in separate threads. It also includes a mechanism to limit the number of concurrent threads. The terminal window shows the command to run the script: `PS C:\Users\mream> C:\Users\mream\AppData\Local\Microsoft\WindowsApps\python3.11.exe "d:/RM/ssh_working_2.py"`. The output displays the generation of an RSA key pair, the key fingerprint, and a ASCII art representation of the key.

```
52     try:
53         client, address = server_socket.accept()
54         logger.info(f"[*] Accepted connection from {address[0]}:{address[1]}")
55
56         # Handle the connection in a separate thread
57         client_handler = threading.Thread(target=handle_connection, args=(client, address, welcome_banner))
58         client_handler.start()
59
60         # Limit the number of concurrent threads
61         if threading.active_count() > num_threads:
62             logger.warning("[!] Maximum number of threads reached. Waiting for threads to finish.")
```

```
PS C:\Users\mream> C:\Users\mream\AppData\Local\Microsoft\WindowsApps\python3.11.exe "d:/RM/ssh_working_2.py"
Generating public/private rsa key pair.
Your identification has been saved in test_rsa.key
Your public key has been saved in test_rsa.key.pub
The key fingerprint is:
SHA256:jG07Xe16qeOdDgX01XTuFcq8m01KE8YvH8771OKcHd4 ramya manasa@LAPTOP-V7EOE9HT
The key's randomart image is:
+---[RSA 2048]-----+
  o o .
  + X o .
  . * O .
  + o o +
  . = S = +
  o o = B B *
  + . = E .
  ..+ * .
  .ooOoo
+-----[SHA256]-----+
```

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`.

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.

ADVANTAGES:

Detection of Unauthorized Access: SSH honeypots can attract and detect unauthorized access attempts. By simulating vulnerable SSH services, they lure potential attackers, allowing security professionals to identify and analyze malicious activities.

Understanding Attack Techniques: Honeypots provide valuable insights into attackers' techniques and tactics. Analyzing the captured data helps security experts understand emerging threats, attack patterns, and vulnerabilities in SSH implementations.

DISADVANTAGES:

False Positives: Honeypots can generate false positives, flagging legitimate activities as malicious. This can lead to unnecessary alerts and potentially divert resources towards non-threatening events.

Resource Consumption: Running honeypots requires resources, both in terms of hardware and network bandwidth. This additional overhead may impact the performance of the systems hosting the honeypots.

7. 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.

8. REFERENCES

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