**HONEYPOT USING PYTHON**

Honeypot is an Host or a server that is impersonating an infrastructure in gerneral they are databases(employee and customer),work stations,servers and Files(also known as canary tokens).

* Multiple honeypots together is known as honeynet,it is majorly used for the simulation of a network at different levels

Development environtment:

* Python3.11+
* Virtual studios

Project Breakdown:

The following honeypot was made up as in modules mainly focusing on

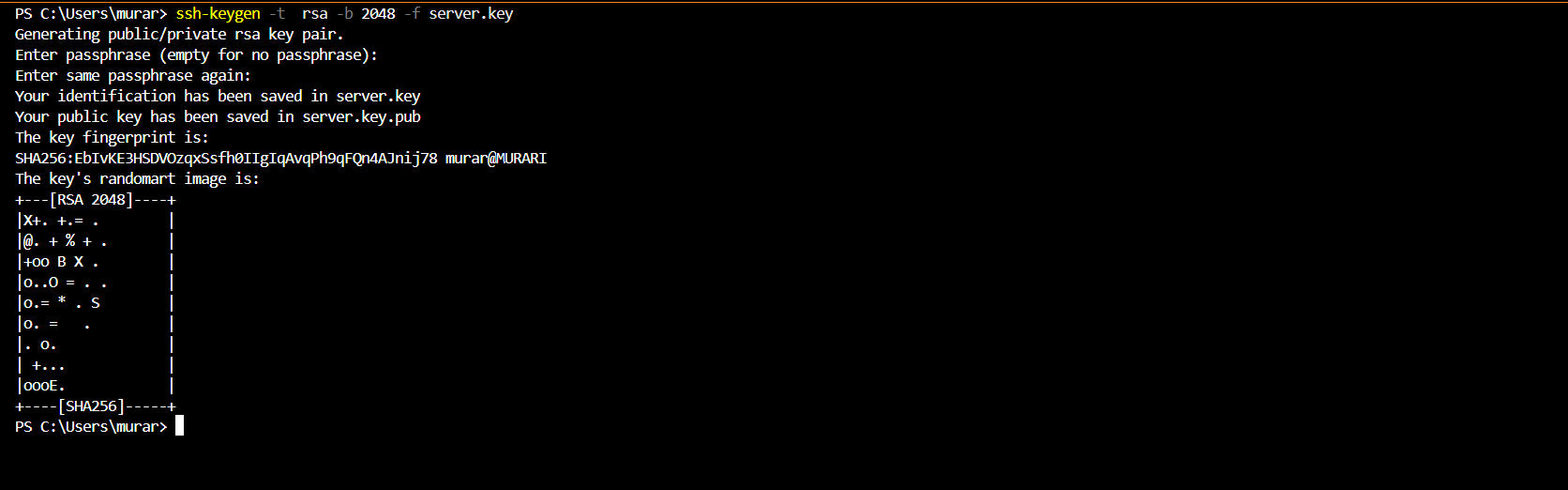
* SSH Honeypot

We are going to mainly use SSH(SSH stands for Secure Shell. It is a cryptographic network protocol that enables secure remote access to computers and servers over an unsecured network, such as the internet.) as our mail interface to lure the attackers into the Honeypot,

* Libraries used are

1. **Logging** : to log the IP address ,username and password that are used on the honeypot
2. **Socket**: the socket library in Python is used to create fake network services by listening on specific ports to capture and log malicious connection attempts from attackers
3. **Paramiko :** paramiko is a pure python implementation of sshv2 protocol providing both client-side and Server-side Functionality.it provides a foundation for high-level SSH library Fabric.
4. **Socket:**  The socket library is used to create and manage network connections. It allows the honeypot to listen for incoming connections and interact with attackers. basically it collects the ip addresses, requests and aps(access points) of the client who uses the honeypot.
5. **Threading :** he threading module is used to handle multiple connections at the same time. It lets the honeypot interact with several attackers without blocking.
6. **Pathlib**: module in Python provides an object-oriented way to work with filesystem paths.

* **We** are going to use RSA algorithm to generate the public private key pair in ssh. Because When you're building a honeypot that simulates an **SSH server**, you want it to **behave like a real SSH server** so attackers are tricked into interacting with it. One critical aspect of this simulation is the **SSH handshake**, which includes key exchange and authentication.
* **Command :** ssh-keygen -t rsa -b 2048 -f server.key

****



**Breakdown of functions used in the program**

**Global Variables**

* **audits\_logger** → Logs authentication attempts (username + password guesses).
* **creds\_logger** → Logs commands executed inside the honeypot shell.
* **standard\_banner** → The fake message shown after login (ssh successfully established...) to look realistic.
* **host\_key** → The server’s private RSA key used for SSH connections. If it doesn’t exist, one is generated.

**Functions**

**1. handle\_connection(client, addr)**

* **Purpose:** Handles a new incoming SSH connection.
* **Parameters:**
  + client: the socket object (raw TCP connection).
  + addr: tuple containing (IP, port) of the client.
* **Steps inside:**
  + Wraps socket with paramiko.Transport(client) → creates an SSH transport session.
  + Loads the honeypot RSA key: transport.add\_server\_key(host\_key).
  + Starts a custom server (transport.start\_server(...)) → uses our Server class.
  + Waits for authentication + channel request.
  + Calls emulated\_shell(channel, addr[0]) if the connection is valid.
  + Logs failures if exceptions occur.

**2. emulated\_shell(channel, client\_ip)**

* **Purpose:** Provides a fake shell to the attacker after login.
* **Parameters:**
  + channel: the SSH communication channel with the client.
  + client\_ip: attacker’s IP address.
* **Behavior:**
  + Sends the standard\_banner.
  + Enters an infinite loop:
    - Reads data from attacker (channel.recv).
    - If empty, closes the channel.
    - Decodes command, strips spaces.
    - Logs the command.
    - Handles known commands (ls, pwd, whoami) with fake outputs.
    - Unknown commands → shows bash: <command>: command not found.
  + Exits if client disconnects.

**3. main()**

* **Purpose:** Entry point of the honeypot.
* **Behavior:**
  1. Creates a TCP socket → socket.socket(socket.AF\_INET, socket.SOCK\_STREAM).
  2. Binds to 127.0.0.1:2025.
  3. Listens for new connections.
  4. For each connection, starts a new thread running handle\_connection.

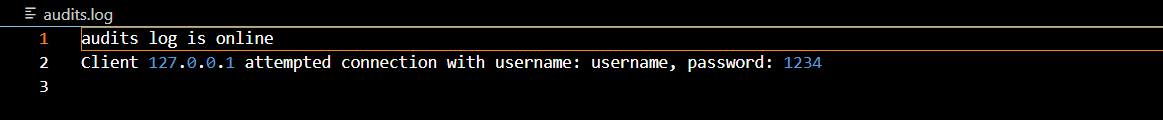
**4. Server (class, subclass of paramiko.ServerInterface)**

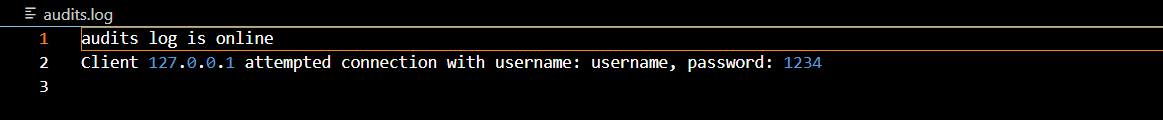
* **Purpose:** Defines SSH server behavior for authentication and channel requests.
* **Methods:**
  + check\_channel\_request(kind, chanid)
    - Allows session channel only → returns paramiko.OPEN\_SUCCEEDED.
    - Rejects others.
  + check\_auth\_password(username, password)
    - Always logs the username/password attempt.
    - Returns paramiko.AUTH\_SUCCESSFUL (lets attacker in, even with wrong creds).

**Important Variables Inside Functions**

* **cmd** (in emulated\_shell) → Stores the attacker’s latest command (decoded from bytes).
* **command** → Same as cmd, but kept as raw bytes. Used for echoing.
* **data** → Raw bytes received from client.
* **transport** → Paramiko object managing SSH session.
* **client\_handler** → Thread that runs handle\_connection for each attacker.
* **addr[0]** → Extracts client IP (ignores port).

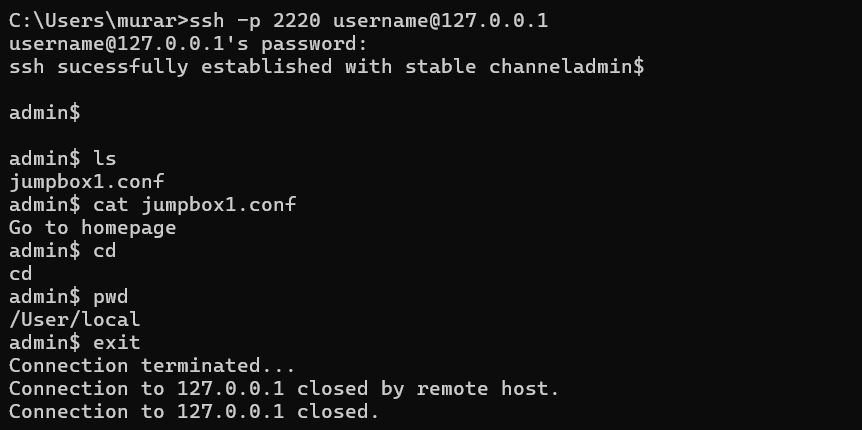
**Logging**

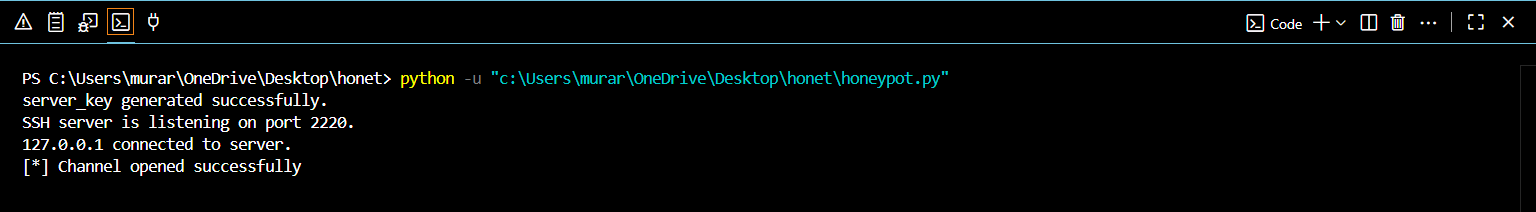
* audits.log → Username/password attempts.
* cmd\_audits.log → Commands executed.



**Note:**

You can increase the number of commands that can be executed in the emulated shell by adding more possible strings to the *#EMULATED SHELL* if–else block starting from line 43.

**sample output using localhost**

****