Name: Stephan Mingoes
ID: 620140933
Course Code: INFO3165

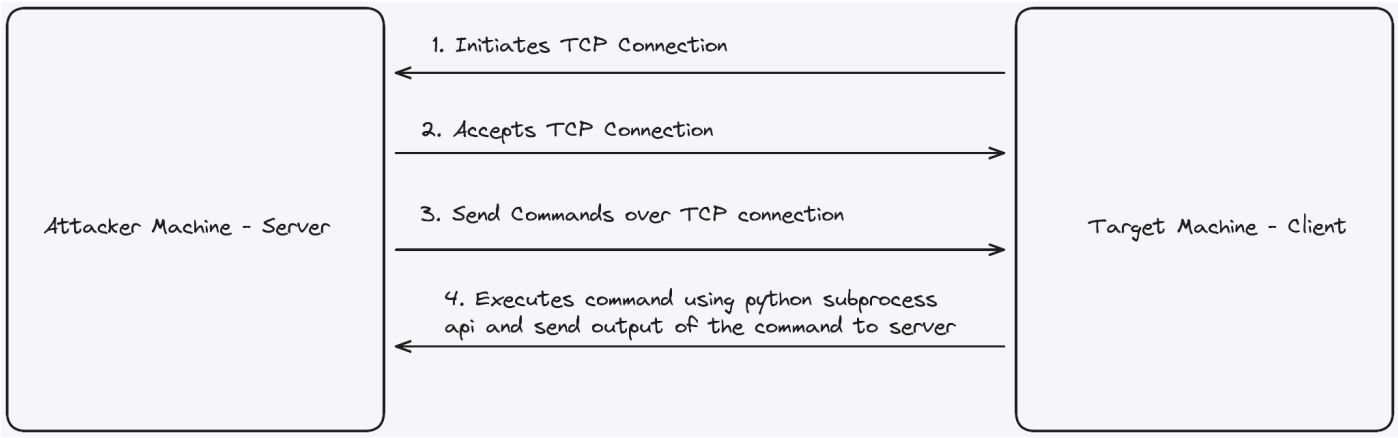
Part 1: Design [5 Marks]

Attackers typically aim to gain interactive shell access on compromised systems, allowing them to execute commands and potentially elevate their privileges. However, accessing systems directly via remote shell connections is often prevented by firewalls. One way to overcome this limitation is by using a reverse shell. A shell serves as a user interface for accessing operating system services. In a reverse shell scenario, the connection originates from the system offering services to the client seeking to utilize them remotely. The target machine initiates the connection to the user's computer, which then listens for incoming connections on a specified port.

Reverse shells play a crucial role in cybersecurity as they are a common technique used by attackers to bypass firewall restrictions and gain unauthorized access to compromised systems. Firewalls are essential components that act as a barrier between an internal network and external networks, filtering incoming traffic based on a set of security rules; they typically block incoming connections on most ports, allowing only specific services to accept connections. This is what leads to the significance of reverse shells because it enables attackers connections from the compromised system back to their own system rendering the firewall restriction of inbound connections obsolete.

In the context of network management, reverse shells can be useful for legitimate purposes. Network administrators can utilize reverse shells to remotely access systems behind firewalls and perform tasks such as monitoring, maintenance, diagnostics, software updates etc, without needing to be physically present at the location of the system.

Diagram detailing interaction between client and server



The diagram depicts two machines communicating: the Attacking machine (server) and the Target machine (client). The server, acting as the attacker's machine, accepts the TCP connection initiated by the client. The client, on the other hand, is the target machine where the client code is injected to facilitate remote access by the attacker.

1. **Initiates TCP Connection** - The client initiates a TCP connection to the server. This involves creating a socket and specifying the server's IP address and port number to connect to. In the reverse shell, this connection is established from the target machine (client) to the attacker's machine (server)

2. **Accepts TCP Connection -** On the server side, the server socket listens for incoming connections. When a client (target machine) connects, the server accepts the connection, creating a new socket specifically for communication with that client. This new socket is used to send and receive data between the client and server.

3. **Send Commands over TCP Connection** - Once the connection is established, the server can send commands to the client over the TCP connection (e.g., ls, pwd, whoami) that the server wants the client to execute on the target machine.

4. **Executes Command using Python Subprocess API and Sends Output to Server** - On the client side, when a command is received from the server, the client uses the Python subprocess module to execute the command on the target

machine's operating system. The output of the command (e.g., the result of ls or pwd) is captured by the client, after which it is sent to the server

Part 2: TCP Server [5 Marks]

Code for the TCP Server - Also attached to the submission of this project

Part 3: TCP Client [5 Marks]

Code for the TCP Client - Also attached to the submission of the project

Operation Instructions [5 Marks]

To run the code, first start the server by running the command **python server.py** then start the client by running **python client.py**.

Both the server and the client use the same IP address and port number to establish a connection. To configure these settings, you can modify the **settings.json** file, where the **server_ip** and **server_port** are stored. When choosing a port for your reverse shell, it's recommended to select a common port such as **80, 443, 8080, 139, or 445**. These ports are more likely to allow traffic, making it easier to establish a connection.

One known issue in the application is that commands that don't return any output can cause the program to freeze. This happens because the client is expecting output from the command to send back to the server, but when there's no output, the client waits indefinitely for something to send. To address this, you can implement a mechanism in the client to handle commands that don't produce output. This way, the client can continue to receive and execute commands from the server even if some commands don't return any output.

There is a custom command **search_password <path to directory>** that prompts the client to execute a bash script that searches for a password file in the given directory. This script written in bash makes use of the **find** command to recursively look for a specified file that has the text “password” in the name and returns a list of file paths to the server of files found under that criteria.