Prakhar Sinha
22BCI0127

Course Code: BCSE309P
Course Title: Cryptography and Network Security Lab
Class Number: VL2024250505094 (L35+L36)
Faculty: DEVIPRIYA A

Develop a simple client server model using telnet and capture the packets transmitted with tshark Analyze the pcap file and get the transmitted data (plain text) using any packet capturing library. Implement the above scenario using SSH and observe the data

Starting the Telnet Server

```
C:\Users\Prakhar\Desktop\22BCI0127>python a.py
Server started on 0.0.0.0:8023
Connection from: 192.168.1.5
Received from 192.168.1.5: Hello Server
Received from 192.168.1.5: This is a secret message
Received from 192.168.1.5: My password is password123
Received from 192.168.1.5: exit
Connection closed: 192.168.1.5

C:\Users\Prakhar\Desktop\22BCI0127>`
```

Capturing Packets with tshark

```
C:\Users\Prakhar\Desktop\22BCI0127>python a.py
Capturing on 'eth0'
45 packets captured
^C

C:\Users\Prakhar\Desktop\22BCI0127>
```

Telnet Client Session

```
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.
Welcome to the simple telnet server!
Type a message and press enter. Type 'exit' to quit.
Hello Server
SERVER ECHO: Hello Server
This is a secret message
SERVER ECHO: This is a secret message
My password is password123
SERVER ECHO: My password is password123
exit
Goodbye!
Connection closed by foreign host.
```

Analysing the Telnet Capture

```
Analyzing telnet capture file: telnet_capture.pcap
Packet #3: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): Welcome to the simple telnet server!
Packet #4: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): Type a message and press enter. Type 'exit' to quit.
Packet #7: 192.168.1.5 -> 192.168.1.10
TCP Payload (ASCII): Hello Server
Packet #9: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): SERVER ECHO: Hello Server
Packet #11: 192.168.1.5 -> 192.168.1.10
TCP Payload (ASCII): This is a secret message
Packet #12: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): SERVER ECHO: This is a secret message
Packet #15: 192.168.1.5 -> 192.168.1.10
TCP Payload (ASCII): My password is password123
Packet #16: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): SERVER ECHO: My password is password123
Packet #19: 192.168.1.5 -> 192.168.1.10
TCP Payload (ASCII): exit
Packet #20: 192.168.1.10 -> 192.168.1.5
TCP Payload (ASCII): Goodbye!
```

Analyzing the SSH Capture

```
Analyzing SSH capture file: ssh_capture.pcap
Packet #1: 192.168.1.5 -> 192.168.1.10
SSH Protocol: SSH-2.0-OpenSSH_8.2p1 Ubuntu-4ubuntu0.5
Packet #2: 192.168.1.10 -> 192.168.1.5
SSH Protocol: SSH-2.0-paramiko_2.7.2
Packet #3: 192.168.1.5 -> 192.168.1.10
Encrypted data length: 756 bytes
Encrypted data: 000000f84a55aa1f7a9beccb851fcd6a96983d2bc574fe58...
Packet #4: 192.168.1.10 -> 192.168.1.5
Encrypted data length: 1024 bytes
Encrypted data: 000000c04d41435f414c474f5349474e4154555245532...
Packet #5: 192.168.1.5 -> 192.168.1.10
Encrypted data length: 84 bytes
Encrypted data: 0000001c0516cf85f634f05d24551ba66f3ef5f1f7ecdb28...
Packet #19: 192.168.1.5 -> 192.168.1.10
Encrypted data length: 52 bytes
Encrypted data: e53d11a12770bf4dcfe3b457daacf785f3a1f42...
Packet #20: 192.168.1.10 -> 192.168.1.5
Encrypted data length: 84 bytes
Encrypted data: 86c25df71a52e9f1ae6f5c3b438ad57f72c1e32f...
```

```python
import socketserver
import threading

class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):
        client_address = self.client_address[0]
        print(f"Connection from: {client_address}")
        try:
            # Send welcome message
            self.request.sendall(b"Welcome to the simple telnet server!\n")
            self.request.sendall(b"Type a message and press enter. Type 'exit' to quit.\n")

            while True:
                # Receive data from client
                data = self.request.recv(1024)
                if not data:
                    break

                message = data.strip().decode('utf-8')
                print(f"Received from {client_address}: {message}")

                # Check if client wants to exit
                if message.lower() == 'exit':
                    self.request.sendall(b"Goodbye!\n")
                    break
```

```
                # Echo the message back with a prefix
                response = f"SERVER ECHO: {message}\n".encode('utf-8')
                self.request.sendall(response)

        except Exception as e:
            print(f"Error handling client {client_address}: {e}")
        finally:
            print(f"Connection closed: {client_address}")

class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    allow_reuse_address = True

def start_server(host="0.0.0.0", port=8023):
    server = ThreadedTCPServer((host, port), ThreadedTCPRequestHandler)
    server_thread = threading.Thread(target=server.serve_forever)
    server_thread.daemon = True
    server_thread.start()
    print(f"Server started on {host}:{port}")

    try:
        # Keep the main thread alive
        while True:
            input("Press Enter to stop the server...\n")
            break
    except KeyboardInterrupt:
        pass
    finally:
        server.shutdown()
        server.server_close()
        print("Server stopped")

if __name__ == "__main__":
    start_server()
```

```
import pyshark
import sys

def analyze_telnet_capture(pcap_file):
    print(f"Analyzing telnet capture file: {pcap_file}")

    # Open the capture file
    capture = pyshark.FileCapture(pcap_file)
```

```
    # Process each packet
    for i, packet in enumerate(capture):
        try:
            # Check if the packet has TCP layer
            if 'TCP' in packet:
                # If the packet has TCP payload
                if hasattr(packet.tcp, 'payload'):
                    # Convert the hex payload to ASCII
                    hex_data = packet.tcp.payload.replace(':', '')
                    try:
                        ascii_data = bytes.fromhex(hex_data).decode('ascii', errors='replace')

                        # Print packet info
                        print(f"\nPacket #{i+1}: {packet.ip.src} -> {packet.ip.dst}")
                        print(f"TCP Payload (ASCII): {ascii_data}")
                    except Exception as e:
                        print(f"Error decoding payload: {e}")
        except AttributeError:
            continue

    capture.close()

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python analyze_telnet.py <pcap_file>")
        sys.exit(1)

    pcap_file = sys.argv[1]
    analyze_telnet_capture(pcap_file)
```

```
import socket
import sys
import threading
import paramiko
import os

# Setup logging
paramiko.util.log_to_file("ssh_server.log")

# Generate an SSH key pair for the server (or use existing)
HOST_KEY_PATH = "ssh_server_key"
if not os.path.exists(HOST_KEY_PATH):
    key = paramiko.RSAKey.generate(2048)
    key.write_private_key_file(HOST_KEY_PATH)
```

```
    print(f"Generated new server key: {HOST_KEY_PATH}")
else:
    key = paramiko.RSAKey(filename=HOST_KEY_PATH)
    print(f"Using existing server key: {HOST_KEY_PATH}")

class SimpleSSHServer(paramiko.ServerInterface):
    def __init__(self):
        self.event = threading.Event()

    def check_channel_request(self, kind, chanid):
        if kind == 'session':
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_auth_password(self, username, password):
        # For demo purposes, accept any username/password
        # In a real server, you would validate against a database
        print(f"Auth attempt - Username: {username}, Password: {password}")
        return paramiko.AUTH_SUCCESSFUL

    def get_allowed_auths(self, username):
        return 'password'

    def check_channel_shell_request(self, channel):
        self.event.set()
        return True

    def check_channel_pty_request(self, channel, term, width, height,
                                  pixelwidth, pixelheight, modes):
        return True

def handle_client(client_socket, client_address):
    print(f"Connection from: {client_address[0]}:{client_address[1]}")

    try:
        # Create a Transport object from the socket
        transport = paramiko.Transport(client_socket)
        transport.add_server_key(key)

        # Start the server
        server = SimpleSSHServer()
        transport.start_server(server=server)

        # Wait for authentication
        chan = transport.accept(20)
        if chan is None:
```

```
            print("No channel opened")
            return

        server.event.wait(10)
        if not server.event.is_set():
            print("Client never asked for a shell")
            return

        # Send a welcome message
        chan.send("Welcome to the simple SSH server!\r\n")
        chan.send("Type a message and press enter. Type 'exit' to quit.\r\n")

        # Handle the session
        while True:
            chan.send("> ")
            line = ""
            while not line.endswith("\r"):
                data = chan.recv(1024)
                if not data:
                    break
                line += data.decode('utf-8')

            line = line.strip()
            print(f"Received: {line}")

            if line == "exit":
                chan.send("Goodbye!\r\n")
                break

            chan.send(f"SERVER ECHO: {line}\r\n")

        chan.close()

    except Exception as e:
        print(f"Error: {e}")
    finally:
        client_socket.close()
        print(f"Connection closed: {client_address[0]}:{client_address[1]}")

def start_server(host="0.0.0.0", port=8022):
    # Create a socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    try:
        sock.bind((host, port))
```

```
        sock.listen(100)
        print(f"SSH server listening on {host}:{port}...")

        while True:
            client, addr = sock.accept()
            thread = threading.Thread(target=handle_client, args=(client, addr))
            thread.daemon = True
            thread.start()

    except KeyboardInterrupt:
        print("Server shutting down...")
    except Exception as e:
        print(f"Error: {e}")
    finally:
        sock.close()

if __name__ == "__main__":
    start_server()
```