Winter Semester 2024-2025

Course Name & Code: BCSE309L – Cryptography and Network Security Lab

Class Number: VL2024250505098

Slot: L45+L46

Lab Assignment-4 Questions

Design of a Network Security Models

1. **Secure Multiparty Key Exchange and Man-in-the-Middle (MITM) Attack Simulation Using Diffie-Hellman**

   A group of three users—**Alice, Bob, and Charlie**—want to securely establish a shared secret key using a **multiparty Diffie-Hellman key exchange protocol** over an insecure channel. However, an attacker **Mallory** attempts a **Man-in-the-Middle (MITM) attack** to intercept and manipulate their key exchange process.

   Implement a Multiparty Diffie-Hellman Key Exchange:
   - Choose a **large prime number (p)** and a **primitive root (g)**.
   - Each participant (**Alice, Bob, Charlie**) selects their **private key** and computes their **public key**.
   - They progressively exchange values to compute a **shared secret key** among all three parties.

   Simulate a Man-in-the-Middle Attack (MITM) by Mallory:
   - Mallory **intercepts and modifies** the public keys exchanged between participants.
   - Instead of allowing Alice, Bob, and Charlie to compute a common shared key, Mallory **establishes separate shared keys** with each party.
   - Verify that **Alice, Bob, and Charlie do not share the same key**, but instead, Mallory has established different keys with each of them.

2. **Secure Client-Server Communication Using SSL Sockets**

   A financial services company wants to securely transmit sensitive **customer data** between its **client application** (used by customers) and its **server** (handling transactions). To achieve this, they must implement a **secure client-server communication model** using **SSL/TLS sockets**, ensuring **encryption, authentication, and integrity** of transmitted data.

   Implement a Secure SSL Client-Server Communication Model

   - Create a **server application** that listens for incoming connections using **SSL sockets**.
   - The **server must authenticate itself** with an SSL/TLS certificate to establish trust.

- The **client application** should securely connect to the server over SSL and transmit a **confidential message (e.g., login credentials or banking details)**.
- The server should **decrypt and respond** with an acknowledgment message.

3. **Secure vs. Insecure Remote Access: Telnet vs. SSH Packet Analysis**

A **network administrator** needs to remotely manage a **server** from a **client machine**. Initially, they use **Telnet**, which transmits data in **plaintext** over the network. An attacker with packet-sniffing capabilities can easily **capture and extract sensitive data (e.g., username, password, and executed commands)** using **Wireshark**.

To enhance security, the administrator switches to **SSH (Secure Shell)**, which encrypts the communication, preventing unauthorized access. Your task is to implement both **Telnet and SSH communication**, capture network traffic, and compare security differences.

Implement an Insecure Client-Server Model Using Telnet

Setup a Telnet Server:

- Install a Telnet server (telnetd) on a machine and start the service.
- Ensure the server is accessible from a remote Telnet client.

Use a Telnet Client:

- Connect to the Telnet server from another machine
- Log in with a **username and password** and execute a few commands (e.g., whoami, ls, cat secret.txt).

Capture Telnet Traffic Using Wireshark

- Open **Wireshark** and start capturing packets on the **network interface**.
- Filter Telnet traffic
- Save the captured packets as a **pcap file** (e.g., telnet_capture.pcap).

Extract Transmitted Plaintext Data from pcap File

- Use a **packet-capturing library** (e.g., pyshark in Python) to analyze the **.pcap file** and extract **plaintext credentials and commands**.

Implement a Secure Client-Server Model Using SSH

- **Setup an SSH Server**:
  - Install and start an SSH server (sshd) on the same machine.
- **Use an SSH Client**:
  - Connect to the SSH server

Capture SSH Traffic Using Wireshark

- Start **Wireshark** and capture SSH packets.
- Filter SSH traffic
- Save the captured packets as **ssh_capture.pcap**

Compare SSH Packet Analysis with Telnet

- Try to extract plaintext data from **ssh_capture.pcap**.
- Observe that **SSH packets are encrypted**, and no readable credentials or commands can be extracted.

4. **Secure Web Authentication Using JSON Web Token (JWT)**

A company is developing a **secure web application** that requires **user authentication**. Instead of using **traditional session-based authentication**, the developers decide to implement **JSON Web Tokens (JWT)** for secure and stateless authentication. However, an attacker attempts to **steal the JWT token** and gain unauthorized access. Your task is to implement **JWT-based authentication**, analyze how the token is generated and validated, and understand the **security risks associated with token-based authentication**.

Develop a Web Application with JWT Authentication

- Create a **backend server** (using **Python/Flask, Node.js/Express, or Java/Spring Boot**) that:
  - Allows **user registration and login**.
  - Generates a **JWT token** upon successful authentication.
  - Uses the JWT token for **protected routes (e.g., user dashboard, account details, etc.)**.
- Create a **frontend (React/HTML+JavaScript)** that:
  - Sends login credentials to the server.
  - Stores and includes the JWT token in HTTP headers when accessing protected routes.

2. Implement JWT Token Verification

- The server should:
  - Verify **JWT tokens** before allowing access to protected routes.
  - Decode JWT payload and extract user data.
  - Reject expired or tampered tokens.

3. Capture JWT Token in HTTP Requests (Security Analysis)

- Use **Wireshark or browser developer tools (Network tab)** to:
  - Capture HTTP requests containing JWT tokens.
  - Observe how the token is sent in the Authorization header.

4. Simulate Security Attacks on JWT

- **Attempt Token Theft**:
  - Copy a valid JWT token and try using it on another machine/browser to test unauthorized access.
- **Tamper with JWT Token**:
  - Decode the token using an online tool (e.g., jwt.io).
  - Modify the payload and re-sign the token to attempt authentication.

5. Implement JWT Security Best Practices

- Use **secret keys** for signing tokens (HS256) or asymmetric encryption (RS256).
- Implement **token expiration** and **refresh tokens**.
- Secure JWT storage using **HTTP-only cookies instead of localStorage**.
- Use **HTTPS** to prevent token interception.