FAST National University of

Computer and Emerging Sciences

Information Security

Assignment 2 – Report

- Name: Muneel Haider
- Roll No: 21I-0640

Introduction

This report explains the testing and analysis of a Python-based client-server application developed for secure communication.
The system uses Diffie-Hellman key exchange to generate secure keys and AES encryption to protect data transmitted over
the network. The main objective is to ensure that sensitive information such as usernames, passwords, and messages is kept private.

The testing process included checking the functionality of features like registration, login, and messaging, as well as ensuring the security
of data in transit. Wireshark, a network traffic analyzer, was used to monitor and verify that no readable (plaintext) data was transmitted.
This document provides a step-by-step explanation of the testing process, the code implementation, the results, and a final conclusion.

Objectives

The objectives of this testing process were:

- Functionality Testing: Ensure the application's core features (registration, login, and chat) work as expected.
- Security Testing: Confirm the secure transmission of sensitive information using Diffie-Hellman and AES encryption.
- Traffic Analysis with Wireshark: Verify that all network traffic is encrypted and no sensitive data is visible in plaintext.

Steps

Setting Up the Environment

1. Install Required Libraries:

- Ensure the following Python libraries are installed:
    - socket: For network communication between client and server.
    - getpass: To securely input passwords without displaying them on the screen.
    - hashlib: For hashing (SHA-256).
    - cryptography.hazmat.primitives.ciphers: For AES encryption and decryption.
    - cryptography.hazmat.primitives.padding: For PKCS7 padding.
    - cryptography.hazmat.backends: Backend support for cryptographic functions.
    - threading: To manage multiple client connections concurrently (server-side).
    - random: For generating private keys.
    - os: For creating salts and initialization vectors (IVs).

2. Run the Server and Client Scripts:

- Launch the server using `server.py`, which listens on a specific port (e.g., 8080).
- Run `client.py` to establish a connection with the server.

3. Set Up Wireshark for Traffic Analysis:

- Open Wireshark and select the appropriate network interface (e.g., Loopback or Ethernet).
- Use the display filter `tcp.port == 8080` to capture only relevant traffic.
- Start the capture before initiating any communication from the client to record all packets.

Functional Testing

Registration:

- Run the client and choose the "register" option. Enter an email, username, and password.
- Registration should fail if the username already exists in the system.
- Verify that the server responds with "Successfully Registered" for new users or "Registration failed" for duplicate usernames.

Login:

- Attempt to log in using correct and incorrect credentials.
- Verify that valid credentials allow login and incorrect credentials return a "Login failed" message.

Chat:

- After successful login, send messages between the client and server.
- Verify that messages are encrypted and both parties can communicate securely.
- Type "exit" to terminate the chat session.

Security Testing with Wireshark

1. Monitor Traffic:

- Observe the Wireshark capture during registration, login, and chat activities.
- Ensure no readable (plaintext) data, such as usernames or passwords, is visible.
- Verify that all communication between the client and server is encrypted.

Code Explanation

Server Code

- **`diffieHell()`**: Implements Diffie-Hellman key exchange to generate a public key from a private key.
- **`createSKey()`**: Creates a shared secret key from the Diffie-Hellman exchange and hashes it using SHA-256 for AES encryption.
- **`aesEncry()` and `aesDecry()`**: Perform AES encryption and decryption using CBC mode.
- **`padMessage()` and `unPadMessage()`**: Handle message padding to match AES block requirements.

Client Code

- Implements Diffie-Hellman for secure key exchange, AES encryption, and message padding.
- Sends user credentials securely encrypted with AES.
- Uses Diffie-Hellman to establish a shared key with the server.

Results

The testing results are as follows:

- **Register New User:** Registration successful.
- **Register Existing User:** Registration failed.
- **Login with Valid Credentials:** Login successful.
- **Login with Invalid Credentials:** Login failed.
- **Chat Communication:** Encrypted message exchange.

Wireshark Verification

All monitored traffic showed encrypted data only, confirming the secure transmission of sensitive information.

Conclusion

The Python-based client-server application effectively secures communication using Diffie-Hellman for key exchange and AES for encryption. Functional testing confirmed the application's capabilities in registration, login, and chat. Security testing verified that all transmitted data was encrypted, with no plaintext data visible in Wireshark captures.

The application demonstrates a strong foundation for secure communication. To maintain this security, regular reviews and updates to encryption methods are recommended to guard against emerging threats.