COMPUTER NETWORK LAB 9

NAME : HARSHIT GARG
ROLL NO : CS22B024

Task 1: TCP Loopback Communication

1. Purpose of the Loopback Address (127.0.0.1):

The loopback address `127.0.0.1` is used for testing network functionalities within a local machine. It allows the system to simulate network communication between a client and server without requiring a physical network. This internal routing enables secure, efficient testing and debugging of applications.

---

2. Data Flow Between Client and Server:

Server Side:

- Initializes a socket and binds it to the loopback address and a designated port.
- Listens for incoming connections via `accept()`.
- Upon a successful connection, receives data from the client using `recv()`.
- Optionally sends a response using `send()`.
- Closes the connection when done.

Client Side:

- Creates a socket and connects to the server at `127.0.0.1` on the specified port.
- Send data using `send()`.
- Optionally receives data via `recv()`.
- Closes the connection afterward.

In summary, the client starts communication and the server responds, all via TCP on the loopback interface.

---

3. Advantages of Using Loopback for Testing:

No Network Dependency: Works without a physical or internet connection.

High Speed: Communication is fast as data remains within the system.

Security: Traffic is confined to the local machine, reducing risk of external exposure.

Isolation: Testing is unaffected by real-world network inconsistencies.

Consistency: Offers repeatable test conditions for accurate debugging.

Task 2: Simulated Remote Encryption

1. Explanation of the Encryption Method Used:

A simple XOR encryption technique was implemented. Each character in the message is XOR-ed with a constant key (`0x5A`). This symmetric method is lightweight and suitable for demonstration purposes. The same process is used for both encryption and decryption.

2. Integration of Encryption into Communication:

Client side :

```
char message[] = "Hello, Encrypted World!";
xor_encrypt_decrypt(message, strlen(message), XOR_KEY); // Encrypt before sending
send(sock, message, strlen(message), 0);


char buffer[BUFFER_SIZE];
int bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);
if (bytes_received < 0) {

} else {
    printf("Echoed (encrypted): %s\n", buffer);
    xor_encrypt_decrypt(buffer, bytes_received, XOR_KEY); // Decrypt response
    buffer[bytes_received] = '\0';
    printf("Echoed (decrypted): %s\n", buffer);
}
```

Server Side :

```
char buffer[BUFFER_SIZE];
int bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);
if (bytes_received < 0) {
    perror("recv failed");
} else {
    printf("Received (encrypted): %s\n", buffer);
    xor_encrypt_decrypt(buffer, bytes_received, XOR_KEY); // Decrypt
    printf("Received (decrypted): %s\n", buffer);


    xor_encrypt_decrypt(buffer, bytes_received, XOR_KEY); // Re-encrypt to echo
    send(client_socket, buffer, bytes_received, 0);
}
```

3. Limitations of the XOR Encryption Used:

Poor Security: Easily cracked due to static key usage.

No Key Management: Key is hardcoded, making it vulnerable.

Pattern Visibility: Identical messages yield the same output.

No Authentication: Cannot detect if a message was altered during transit.

---

4. Common Socket Errors and Their Handling:

Socket Creation Failure: Occurs when resources are insufficient. Handled by showing an error and terminating the program.

Bind Failure: Happens when the address is unavailable or already in use. The socket is closed and the error is logged.

Connection Failure: When the client can't reach the server. The client logs the error and exits.

Transmission Errors: Problems during send/receive due to buffer issues or broken links. Return values of the functions are checked to detect and report these errors.

---

5. Key Considerations for Real-World Key Management:

- **Confidentiality**: Keys must be kept secret and not exposed in code.
- **Secure Key Exchange**: Protocols like RSA or Diffie-Hellman should be used.
- **Key Rotation**: Periodically changing keys reduces compromise risks.
- **Secure Storage**: Store keys in encrypted files or secure hardware.
- **Access Control:** Only authorized entities should access the keys.
- **Integrity Checks**: Ensure keys haven’t been tampered with.
- **Minimized Exposure**: Avoid logging or transmitting keys unnecessarily.