INFORMATION SYSTEMS SECURITY

PROJECT - 1

GROUP MEMBERS: KARADE ASHISH (B00979064)

ANINDA MANDAL (B00980010)

Introduction:

The above program is a basic client-server application that utilizes Secure Sockets Layer/Transport Layer Security (SSL/TLS) encryption and message digest for data integrity verification. The program consists of two scripts, alice.py and bob.py, which are implemented in Python.

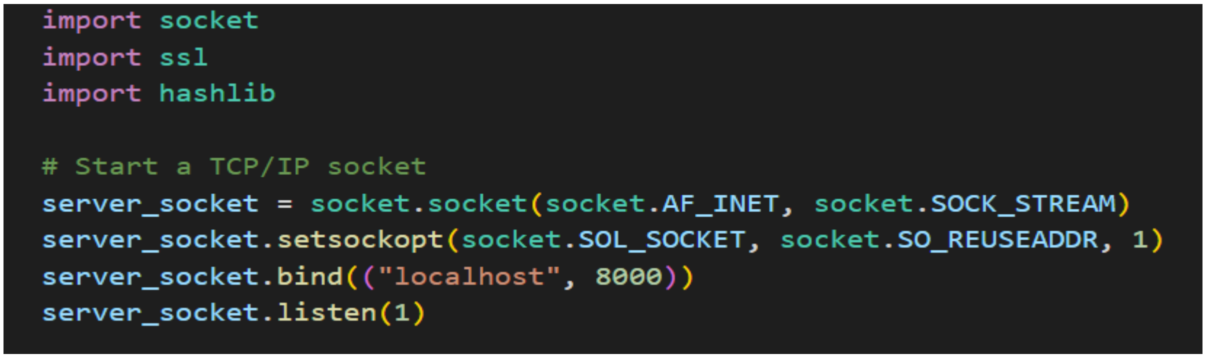
Implementation:

This program is a basic implementation of a Alice-Bob as known as client-server application with SSL/TLS encryption and message digest for data integrity verification. The server.py script creates a TCP/IP socket and listens for client connections. Once a client connects, the server wraps the client socket with SSL/TLS encryption and receives data from the client. The server then computes the message digest of the received data using the SHA-256 algorithm and sends the digest back to the client. The client.py script creates a TCP/IP socket and connects to the server. It then wraps the client socket with SSL/TLS encryption and sends data to the server. The client also computes the message digest of the sent data using the SHA-256 algorithm and compares it with the digest received from the server to verify the integrity of the data.

Here is the step by step implementation for server and client,

Server.py:

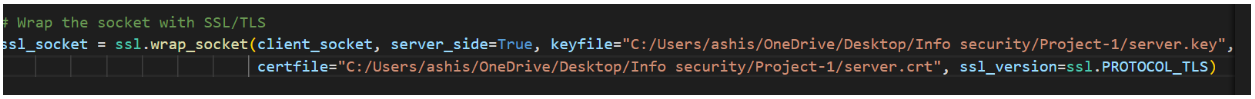
1. Import necessary modules: socket, ssl and hashlib.
2. Create a TCP/IP socket using socket.socket() method and assign it to a variable called server_socket.
3. Set the socket options by using setsockopt() method. Set socket option level as SOL_SOCKET and option name as SO_REUSEADDR. This allows the socket to bind to a port which was previously in use.



```python
import socket
import ssl
import hashlib

# Start a TCP/IP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind(("localhost", 8000))
server_socket.listen(1)
```

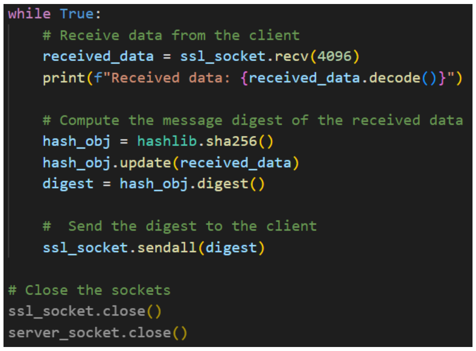
4. Bind the socket to the local host and port number using bind() method.
5. Listen for a client connection using listen() method.
6. Wait for a client to connect by using accept() method which will return client socket object and client address.

7. Wrap the client socket with SSL/TLS using wrap_socket() method. Set server_side to True since the server is wrapping the client socket. Specify the key and certificate file locations for SSL/TLS. (IMPORTANT: For this program to run we have to add the file path of generated keyfile and certificate i.e., “server.key” and “server.crt”. Please refer readme.txt for more info)



```
# Wrap the socket with SSL/TLS
ssl_socket = ssl.wrap_socket(client_socket, server_side=True, keyfile="C:/Users/ashis/OneDrive/Desktop/Info security/Project-1/server.key",
                             certfile="C:/Users/ashis/OneDrive/Desktop/Info security/Project-1/server.crt", ssl_version=ssl.PROTOCOL_TLS)
```

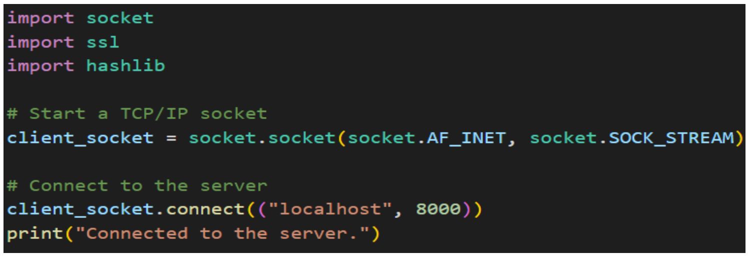
8. Create a loop that runs indefinitely.
9. Receive data from the client using recv() method.
10. Compute the message digest of the received data using sha256() method from hashlib module.
11. Send the digest to the client using sendall() method.
12. Close the SSL/TLS and server sockets using close() method



```
while True:
    # Receive data from the client
    received_data = ssl_socket.recv(4096)
    print(f"Received data: {received_data.decode()}")

    # Compute the message digest of the received data
    hash_obj = hashlib.sha256()
    hash_obj.update(received_data)
    digest = hash_obj.digest()

    #  Send the digest to the client
    ssl_socket.sendall(digest)

# Close the sockets
ssl_socket.close()
server_socket.close()
```

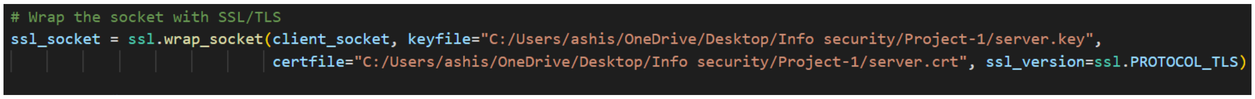
Client.py:

1. Import necessary modules: socket, ssl and hashlib.
2. Create a TCP/IP socket using socket.socket() method and assign it to a variable called client_socket.
3. Connect to the server using connect() method and pass the server host and port number as arguments.



```
import socket
import ssl
import hashlib

# Start a TCP/IP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
client_socket.connect(("localhost", 8000))
print("Connected to the server.")
```

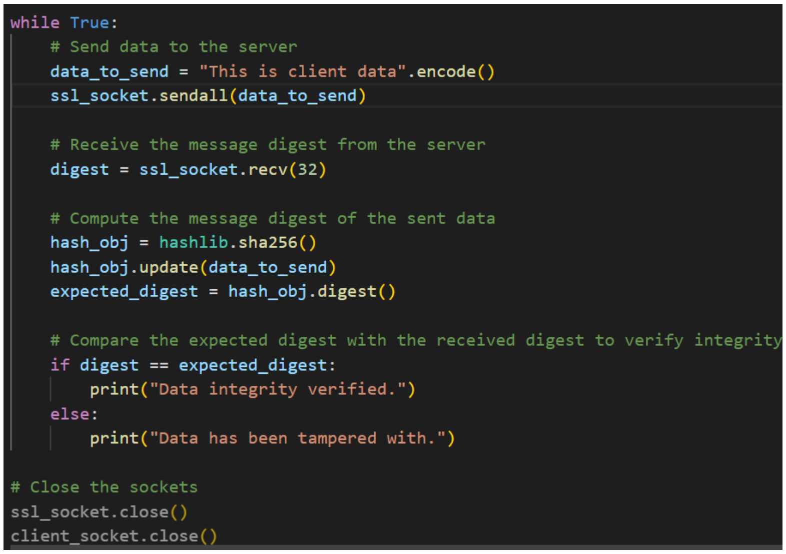
4. Wrap the client socket with SSL/TLS using wrap_socket() method. Specify the key and certificate file locations for SSL/TLS.



```
# Wrap the socket with SSL/TLS
ssl_socket = ssl.wrap_socket(client_socket, keyfile="C:/Users/ashis/OneDrive/Desktop/Info security/Project-1/server.key",
                             certfile="C:/Users/ashis/OneDrive/Desktop/Info security/Project-1/server.crt", ssl_version=ssl.PROTOCOL_TLS)
```

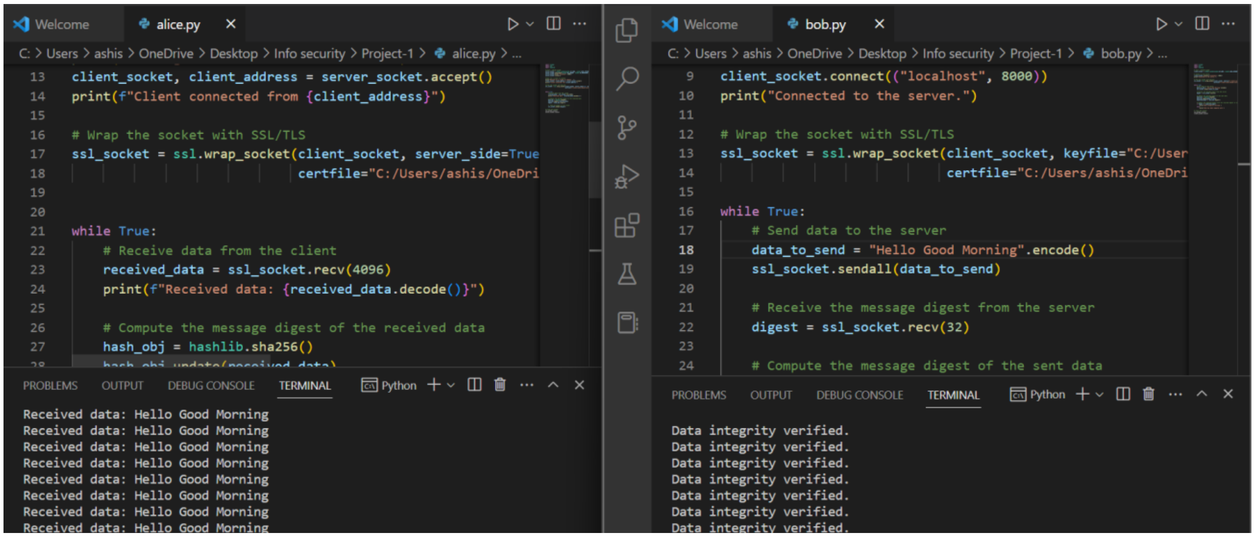
5. Create a loop that runs indefinitely.
6. Send data to the server using sendall() method.

7. Receive the message digest from the server using recv() method.
8. Compute the message digest of the sent data using sha256() method from hashlib module.
9. Compare the expected digest with the received digest to verify integrity of data.
10. Close the SSL/TLS and client sockets using close() method.



```
while True:
    # Send data to the server
    data_to_send = "This is client data".encode()
    ssl_socket.sendall(data_to_send)

    # Receive the message digest from the server
    digest = ssl_socket.recv(32)

    # Compute the message digest of the sent data
    hash_obj = hashlib.sha256()
    hash_obj.update(data_to_send)
    expected_digest = hash_obj.digest()

    # Compare the expected digest with the received digest to verify integrity
    if digest == expected_digest:
        print("Data integrity verified.")
    else:
        print("Data has been tampered with.")

# Close the sockets
ssl_socket.close()
client_socket.close()
```

Output Generated:



Welcome
alice.py
C: > Users > ashis > OneDrive > Desktop > Info security > Project-1 > alice.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python

```
client_socket, client_address = server_socket.accept()
print(f"Client connected from {client_address}")

# Wrap the socket with SSL/TLS
ssl_socket = ssl.wrap_socket(client_socket, server_side=True
                             certfile="C:/Users/ashis/OneDri


while True:
    # Receive data from the client
    received_data = ssl_socket.recv(4096)
    print(f"Received data: {received_data.decode()}")

    # Compute the message digest of the received data
    hash_obj = hashlib.sha256()
    hash_obj.update(received_data)
```

```
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
Received data: Hello Good Morning
```

Welcome
bob.py
C: > Users > ashis > OneDrive > Desktop > Info security > Project-1 > bob.py > ...
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python

```
client_socket.connect(("localhost", 8000))
print("Connected to the server.")

# Wrap the socket with SSL/TLS
ssl_socket = ssl.wrap_socket(client_socket, keyfile="C:/User
                             certfile="C:/Users/ashis/OneDri

while True:
    # Send data to the server
    data_to_send = "Hello Good Morning".encode()
    ssl_socket.sendall(data_to_send)

    # Receive the message digest from the server
    digest = ssl_socket.recv(32)

    # Compute the message digest of the sent data
```

```
Data integrity verified.
Data integrity verified.
Data integrity verified.
Data integrity verified.
Data integrity verified.
Data integrity verified.
Data integrity verified.
```

Security features:

The use of SSL/TLS encryption provides a secure channel for data transmission between the client and server. The message digest calculated using the SHA-256 hashing algorithm provides a way to verify the integrity of the data, ensuring that the data has not been tampered with during transmission.

Limitations:

The program does not handle errors or exceptions, which can lead to unexpected behavior and system crashes.

Conclusion:

In conclusion, the above program provides a basic implementation of a client-server application that utilizes SSL/TLS encryption and message digest for data integrity verification. However, further improvements can be made to enhance the security and robustness of the system, such as adding authentication, access control, and error handling features.