PRACTICAL NO. 7

Web Security with SSL/TLS

Aim: Configure and implement secure web communication using SSL/TLS protocols, including certificate management and secure session establishment.

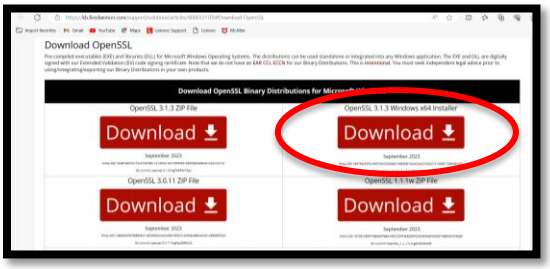
We implement this using a simple echo socket server in python.

Prerequisites:

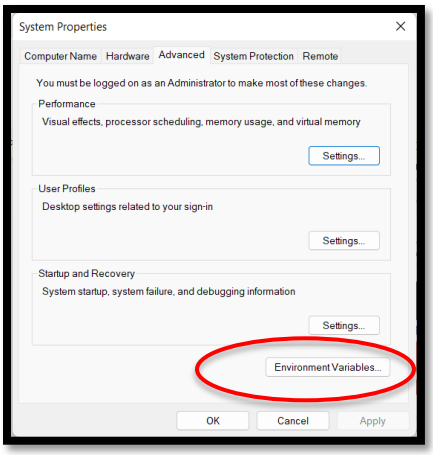
- Python 3
- OpenSSL 3.1: https://kb.firedaemon.com/support/solutions/articles/4000121705#Download-OpenSSL (Make sure to add the bin folder, usually **C:\Program Files\FireDaemon OpenSSL 3\bin** to your environment variable path)

Steps:

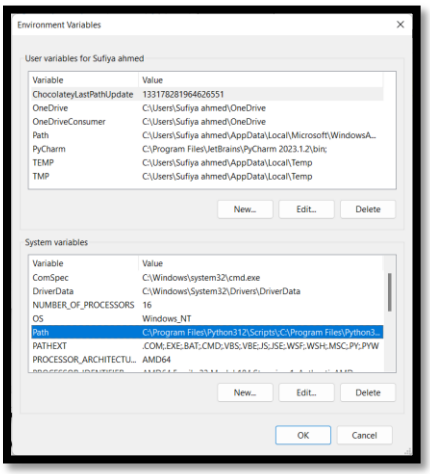
Step 1: download OpenSSL using the below link:
https://kb.firedaemon.com/support/solutions/articles/4000121705#Download-OpenSSL



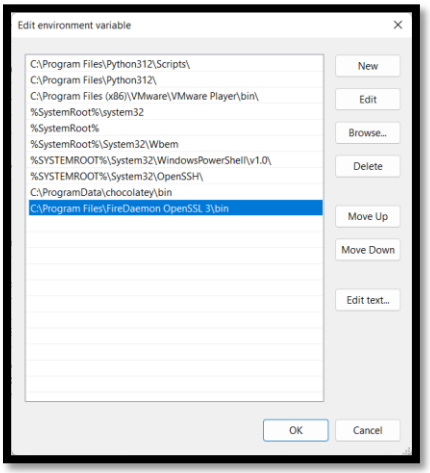
Step 2: In search bar type System Environment Variables > System Properties
Click on "Advanced system settings" on the left sidebar.
In the System Properties window, click the "Environment Variables" button.



In the Environment Variables window, under "System variables," scroll down to find "Path" and click "Edit."



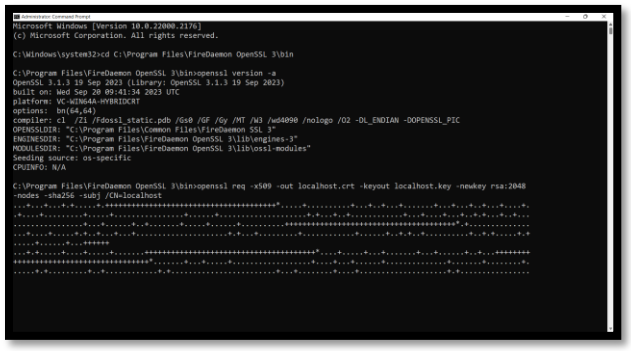
Click "New" and add the path to the OpenSSL binary directory.



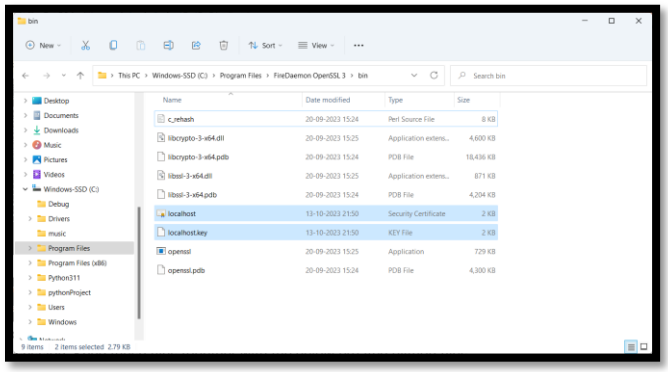
Click "OK" on all open windows to save your changes.

Step 3: Use the following command to generate a new self-signed SSL certificate and key for localhost:

openssl req -x509 -out localhost.crt -keyout localhost.key -newkey rsa:2048 -nodes -sha256 -subj /CN=localhost

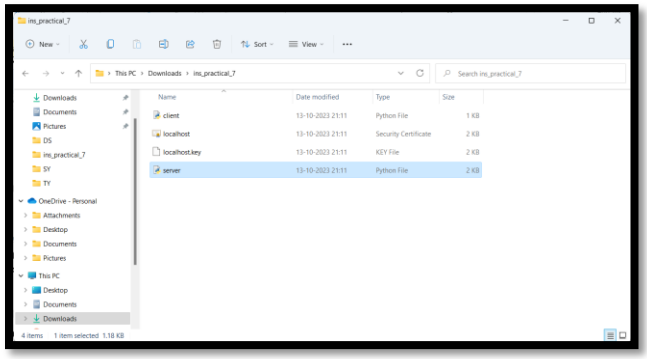


Now check certificate and key file generated.



This command creates a certificate (localhost.crt) and key (localhost.key) file to be used for the secure session.

Step 4: Create one folder in Desktop > Name it as INS Practical 7 > Keep these files (localhost.crt) and (localhost.key) together with the below-created python files in that folder.



Step 5: Create Server and Client

Create the SSL socket server (server.py) with the following code:

```
import socket
import ssl

context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
context.load_cert_chain(certfile="localhost.crt", keyfile="localhost.key")
```

```python

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind(("", 4434))
    server.listen(5)
    print("Server ready and listening for connections")

    # Wait for new connections in a loop
    while True:
        sock, address = server.accept()
        print("New connection from", f"{address[0]}:{address[1]}")

        # Wrap socket with ssl
        ssl_sock = context.wrap_socket(sock, server_side=True)

        while True:
            data = ssl_sock.recv(1024)
            # Decode byte array to utf-8 string
            decoded = data.decode('utf-8')

            # Close the socket if the sock sends empty bytes
            if decoded == "":
                break
            # Log what the sock sends
            print(f"[{address[0]}:{address[1]}] {decoded}")

            # Echo the data back to the sock
            ssl_sock.sendall(data)

        # Gracefully close the connection and wait for next one
        print("Closing connection with", f"{address[0]}:{address[1]}")
        ssl_sock.close()

```

Create the SSL socket client (client.py) with the following code:

```python
import socket
import ssl

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.settimeout(10)

    # Wrap socket with ssl
    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.load_verify_locations('localhost.crt')

```

```
ssl_sock = context.wrap_socket(sock, server_hostname="localhost")

# Connect to the server
ssl_sock.connect(("localhost", 4434))
print("Connected to server")

# Send input data to server and wait for response in a loop
while True:
    ssl_sock.sendall(bytes(input(">"), "utf-8"))
    data = ssl_sock.recv(1024)
    print("Server responded:", data.decode('utf-8'))

```

Run the server.py file

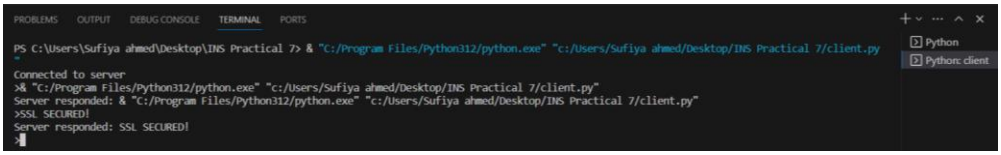
Output:

Keep the server running, and run the client.py file

Client output:

Server output:

Test the connection by sending a message from the client by typing in the console.



The server will echo the same content of the message back to the client.