Implementing SSL/TLS on a Node.js Server

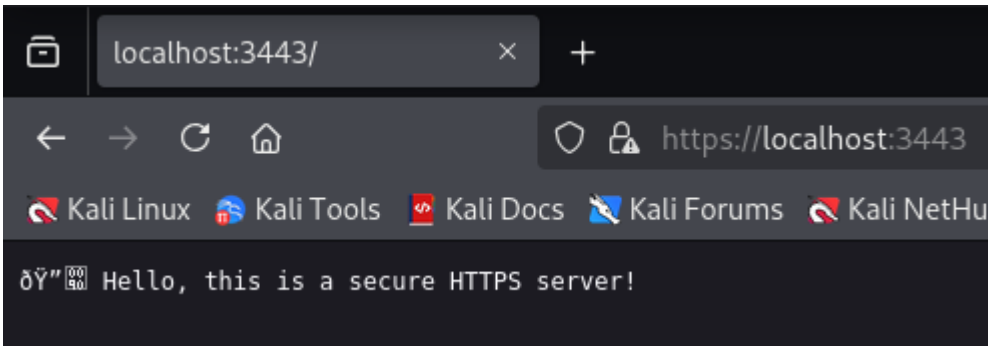
Student Name: Jerome Arsany Mansour Farah
Id: 2305093
Course: Secure Software Development

1. Overview:

In this lab, I successfully transitioned a basic Node.js HTTP server to a secure HTTPS server. This process involved generating cryptographic keys, creating a self-signed certificate using OpenSSL, and configuring the Node.js server to handle encrypted traffic using the TLS/SSL protocol.

2. Certificate Generation:

I used OpenSSL to generate a private key (server.key) and a self-signed certificate (server.cert). The certificate was configured with the Common Name (CN) set to "localhost" to match the local testing environment.



3. Secure Browser Access:

After modifying the server code to use the https module and the generated certificates, I launched the server on port 3443. When accessing https://localhost:3443, the server successfully loaded the secure content.

4. Explanation of Browser Warning:

When accessing the secure server, the browser displayed a "Potential Security Risk" warning.

Reason:
The browser warns the user because the certificate is **self-signed**. Web browsers rely on a "Chain of Trust" where certificates must be validated by a trusted third-party **Certificate Authority (CA)** (such as Verisign or Let's Encrypt). Since I generated this certificate locally on my own machine, the browser cannot verify the identity of the server against its list of trusted authorities, resulting in the warning. However, the connection itself is still encrypted.

5. Command Line Verification ("Curl"):

I verified the server's response using the curl command-line tool. I used the -k (or --insecure) flag to explicitly tell curl to trust the self-signed certificate for testing purposes.

```
┌──(kali㉿kali)-[~/SecureLab/certs]
└─$ openssl req -nodes -new -x509 -keyout server.key -out server.cert -days 365
.....+.........+.............+.....+....+..............+ ... +....+++++++++++++++
.++++++++++++++++++++++++++++++++++++++*....+.+......+ ... +................+.+
... +.............+........+.......+ .. +......+ ... +.......+ .. +......+.+ ... + .. +...
.......+..........+ .. + ... +.......+........+.......+......+ .. +....+ ... + ... + .. +..
.+.....+.......+ ... +...........++++++
.+.........+.+.....+ ... +.++++++++++++++++++++++++++++++++++++++* ... +....+.....
+.....+.++++++++++++++++++++++++++++++++++++++++*....+ ... + ... +....+ ... + .. +.+....
.+.+............+ ... +..............+...........+ ... +.+............+ .. + ... + ... +
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:EG
State or Province Name (full name) [Some-State]:Alex
Locality Name (eg, city) []:Sporting
Organization Name (eg, company) [Internet Widgits Pty Ltd]:University
Organizational Unit Name (eg, section) []:Alex University
Common Name (e.g. server FQDN or YOUR name) []:localhost
Email Address []:jerome.arsany12@gmail.com
```

6. Comparison: HTTP vs. HTTPS:

The following table highlights the key differences observed between the original insecure server and the modified secure server.

Feature	HTTP Server (Insecure)	HTTPS Server (Secure)
Protocol	HTTP (HyperText Transfer Protocol)	HTTPS (HTTP Secure)
Encryption	None (Data sent in Plain Text)	Encrypted (SSL/TLS)
Port Used	3000	3443
Data Privacy	Vulnerable to eavesdropping	Protected from eavesdropping
Certificate	Not required	Requires SSL/TLS Certificate

7. Conclusion:

This lab demonstrated the practical implementation of SSL/TLS. I successfully generated keys, configured a Node.js server for HTTPS, and verified the encryption using both a web browser and command-line tools. I also analyzed why self-signed certificates trigger trust warnings in browsers.