Laboratory 4: The Secure HyperText Transfer Protocol (HTTPS)

Introduction

This laboratory aims to equip you with knowledge on how to:

- Reply to HTTPS get requests with **HTML** webpages using the **HTTPS** protocol.
- Modify the webpage content to convey temperature and humidity information.

Equipment and resources

Hardware:

- ESP8266 Aston IoT hardware platform.
- USB A to micro B cable.
- Access to an access point.

Software & documentation:

- Arduino Integrated Development Environment (IDE).

A reminder on saving and file name convention

You should keep a copy of every task that you complete on your **student network drive (H:), in the folder H:\EE3IOT\labs.** Sketches should be saved in the format of **surnameinitialLTX**, where **L** is the lab number and **X** is the task number.

For example, if Fred Smith has completed task 1 of Lab 1, the sketch should be saved as **smithfL1T1**.

Task 0: A very brief overview of how HTTPS works

HTTP/HTTPS utilises a response and request model using TCP. UDP is not utilised as it does not guarantee delivery of packets or that packets will arrive in the correct order. Referring to Figure 1, the client (typically a web browser application) makes a request of the server (the Aston IOT board).

The Aston IOT application trainer must decide upon what action it should take from this request, and respond appropriately. It is important to note that:

a) HTTP/HTTPS is an application protocol that is responsible for requests and delivery of content.
b) HTML is a mark-up language, i.e. it contains webpage content and how this content should be presented in a web browser. HTML does not handle how web content gets delivered to your browser.

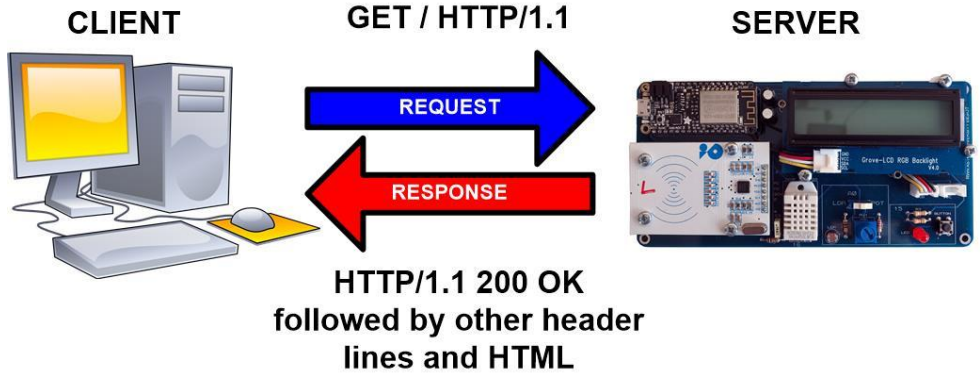

Figure 1: Simplified block diagram of the HTTP protocol

Hence, our sketch needs to implement the following functionality:

1. Check for a client requests periodically.
2. Store the request in memory.
3. Dependent upon the request decide upon an action to be taken. Additional arguments can be passed to the server after the IP address and this is represented in the request as text after the first forward slash of the GET request or after a ?.
4. Respond appropriately with a HTTP/HTTPS response, which consists of a header followed by a HTML webpage, the content our server is to deliver.

Thankfully, most of this is done for us by the library. However, HTTPS also handles:

1. Encryption. All data is encrypted, therefore if we look at data on the network, it will look like random characters. This is not true of HTTP, where data would be visible as plain text. This is achieved via public key cryptography where two keys are used. One is used for encryption (the public key) and another one is used for decryption (the private key). Anyone can encrypt data using the public key issued by the server, but it is almost impossible to decrypt data without the private key.
2. It creates trust, i.e. we can ensure that we are actually communicating with the server and are not visiting a site pretending to be the site we wish to visit.

Task 1: Connecting to the access point and setting up a server

1. Create a copy of your sketch from laboratory 2 task 1 for this task. Utilise the file naming convention given above and used throughout the module.
2. Include the following additional libraries to your sketch:

```
#include <ESP8266mDNS.h>
#include <ESP8266WebServerSecure.h>
```

   - The **mDNS** library allows for our board to be represented by a domain name (for example https://nockr.local/ rather than an IP address. Imagine how difficult browsing the web would be if you were required to remember IP addresses for sites you often visit! mDNS (multicast Domain Name Server) is a way of resolving domain names to IP addresses, without a dedicated DNS server and is primarily used in small networks.
   - The **WebServerSecure** library offers secure webserver functionality. I.e. it uses encryption to ensure data is transferred across the network securely. We will cover how this works a little more later on in the course.
3. Change the SSID you connect to "**EE4IOT_IoT**".
4. Create a global instance of the ESP8266WebServerSecure class called **SERVER**.

```
BearSSL::ESP8266WebServerSecure SERVER(443);
```

   - This should be placed **above your setup()** function.
   - The number 443 is a parameter for the class constructor and configures the server to operate on port 443, which is a port typically used by **HTTPS**. Port 80 is typically used for HTTP, which offers no encryption or authentication.
5. We need to define the domain name of which our board will use. Below your statements declaring the **ssid** and **password**, add the following, replacing **nockr** with your own Aston username. You must change this, such mDNS will resolve your board's IP address when you type it into a browser.

```
const char domainname[] = "nockr";
```

6. Add a server certificate (public key) and a private encryption key to your sketch. One is provided for you on blackboard as **certandkey.txt**.

- Copy the contents of this file to your sketch. These arrays should be placed globally, i.e. before your **setup** function.
- This is a self-generated certificate. This will cause issues later on as they are not from a trusted authority and most web browsers will rightfully complain about it.

7. Create a function `void respond ( void )` and create a function prototype (not strictly required in Arduino programming, but it is still good practice). Note that nothing is required in this function at the moment, but this function will contain our response to an HTTPS request.
8. At the end of your **setup()** function, add the code shown in Figure 2 to start a server.

```
if (MDNS.begin(domainname))
{
   Serial.print("Access your server at https://");
   Serial.print(domainname);
   Serial.println(".local");
}

SERVER.getServer().setRSACert(new BearSSL::X509List(cert), new BearSSL::PrivateKey(key));
SERVER.on("/", respond);
SERVER.begin();

Serial.println("Server is now running");
```

Figure 2: Code to start an HTTPS server

- We start mDNS processes with the MDNS.begin method call, and pass it the value of our desired domain name. Upon successfully starting the mDNS processes (returning 1), we can print a message letting us know the full address to access the board's HTTPS server.
- The following lines are a little more involved. The line highlighted in red configures the server with our public key (cert) and private key (key).
- The line in blue defines how the server should respond to requests. In our case, it will call the function **respond**, of which you just created the stub for. Remember that a function name is actually a pointer to start of the functions code.
- Finally, the line in the purple box starts the actual server.

9. Referring to the previous laboratory, add code to the end of your sketch such that the local address is displayed on top line of the LCD and the IP address is shown on the lower line of the Grove RGB LCD display.
   - This can be done by using the **LCD.print()**. Hint: you can use **LCD.print()** same way as you do with **Serial.print**!
10. Upload the sketch to the IOT hardware platform and verify it works as expected via the Serial Monitor and the LCD.

Task 2: Serving a webpage

1. Create a new sketch and copy the code from task 1 to use as a starting point.
2. In the **loop()** function of your sketch, add the code shown below. This will periodically update all processes required for our **mDNS** and **HTTPS** webserver.

```
SERVER.handleClient();
MDNS.update();
```

3. Copy the function **servepage()** from **servepage.c** from the **Laboratory sessions** section of blackboard into the end of your sketch.
   - Remember to create a function prototype after your #include directives. This is merely a case of copying the function header and adding a semicolon at the end.
4. Modify the **servepage()** function such that it includes **your** username and information. It should not end up displaying my name!
5. Call the **servepage()** function from the **respond()** function.
6. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.
7. Verify that your sketch works and displays your webpage on the access point PC or via your mobile phone.
   - Try accessing your webpage server via typing the address in as follows: https://yourusername.local
   - The browser will probably require that you accept the risks that the certificate is not from a trusted authority.
   - On a iPhone, when the "This connection is not private", click show details. When the new text appears, click on "visit this website". Finally, click "Visit Website".
   - When using Chrome, when the warning is generated (NET::ERR_CERT_AUTHORITY_INVALID"), click on "Advanced", then click on "Proceed to yourusername.local (unsafe)"

Task 3: Using the DHT22 temperature and humidity sensor to populate the page

1. If you have not already done so, download the DHT sensor library from **Blackboard EE3IOT | Learning resources | Support Material | Board information.** Then add the library in the Arduino IDE via **Sketch | Include Library | Add .ZIP Library…**
   - Add the library you have just downloaded.
2. Include the SimpleDHT.h library in your sketch.
3. Use a macro to define DHTPIN as 16.
4. Instantiate a public instance of the SimpleDHT22 class as of:
   - `SimpleDHT22 dht22;`
   - This should be placed above your setup() function.
5. Add the code shown in Figure 3 to your sketch. This function will measure the temperature and humidity via a one wire communication interface to the DHT22 sensor.
   - Place the function at the end of your sketch and do not forget to include a prototype.
6. Create a copy of the **servepage()** function called **servepagewithtemp()**.
7. Modify the function **servepagewithtemp()** such that the webpage conveys both temperature and humidity information to the user.
   - The C language only allows one return value. Hence, the **gettemphumid()** function uses pointers to circumvent this limitation.
   - You must declare two floating point variables in your **servepagewithtemp()** function to store the result of this function.
   - Instead of passing copies of these variables to the **gettemphumid()** function, we can pass the memory address of the local variables in the **servepagewithtemp()** function.
   - This is achieved via the address of operator, &. For example:
     i. `a = temp;` // will copy the value of temp into A.
     ii. `a = &temp;` // will copy the memory address of where variable is stored to a.
   - **Detailed knowledge of HTML is not needed. You can merely add extra lines as shown in the servepage() function.**
   - **A new line is represented by
 in HTML.**
8. Instead of calling **servpage()** in the **respond()** function, call **servepagewithtemp()** instead.
9. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.

10. Verify that your sketch works and displays your webpage with temperature and humidity information on the access point PC.

```
int gettemphumid ( float *temperature, float *humidity )
{

  if ( dht22.read2(DHTPIN, temperature, humidity, NULL) != SimpleDHTErrSuccess )
  {
    Serial.print("Failed to read DHT sensor");
    delay(2000);
    *temperature = -999;
    *humidity = -999;
    return 0; // fail
  }

  Serial.print("Humidity: ");
  Serial.print(*humidity);
  Serial.println(" RH(%)");
  Serial.print("Temperature: ");
  Serial.print(*temperature);
  Serial.println(" *C");
  return 1; // success
}
```

Figure 3: Code to be measure the temperature and humidity

Task 4: Using arguments after the domain

This task will modify your sketch from **task 3** such that your sketch can implement more functionality. In particular, we want to be able to control the state of our program running on the Aston IOT board.

1. In the **respond()** function, we can check to see if arguments have been passed after the domain name. For example, if we were to type https://username.local?LEDON, we could check to see if the argument "LEDON" is present.

   Modify the respond function (and your **setup()** function, remember you must configure the pin as an output) such that you can control the state of the LED as of:
   - **?LEDON** turns the LED on.
   - **?LEDOFF** turns the LED off.
   - To achieve this, use the SERVER.hasArg() method. Note that it accepts the argument you wish to check for as its argument, without the ? mark and returns true upon the argument being found.
2. Modify the **servepagewithtemp()** function such that it also reports the state of the LED. For example, the website should state "The LED is on", when the LED is on and "The LED is off" when the LED is off.
3. Modify the **servepagewithtemp()** function such that it includes hyperlinks to control the state of the LED.
   - Refer to this link on how to achieve this: https://www.w3schools.com/html/html_links.asp
   - Note, that you can use the local link style of hyperlink to achieve this.
4. Debug your sketch and program the Aston IOT board. Verify your code's functionality.