Laboratory 5: Board discovery

Introduction

Working in pairs is encouraged for this laboratory. This laboratory aims to equip you with knowledge on how to:

- Find the IP (logical address) of other boards connected to the network.

Equipment and resources

Hardware:

- ESP8266 Aston IoT hardware platform.
- USB A to micro B cable.
- The shared class computer acting as 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: The Aston Discovery Protocol

To date, our IoT boards have been assigned an IP address by the access point via DCHP (Dynamic Host Configuration Protocol). However, this poses a problem, how does the monitoring system or other boards (for M2M – Machine-to-Machine communication) know if a mote is on the network and if so, obtain its IP address for communication?

Hence, a device discovery protocol is needed. This process will utilise the last address on the network for UDP **broadcast** messages such that a request can be made to all devices on the network if a particular device is present. The simple protocol is shown in Figure 1.

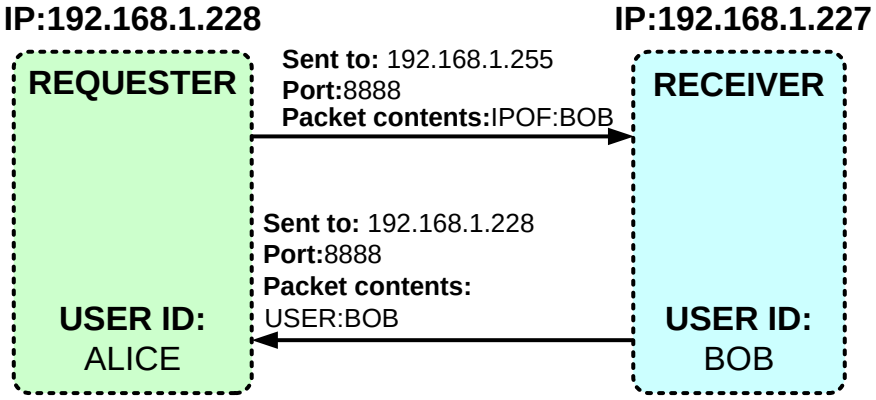

Figure 1: The Aston device discovery protocol

This protocol requires each board to have three processes for discovery:

1. A way of requesting if a known mote ID (Bob in this example) is present on the network and request its IP address. This involves sending out a packet on the **broadcast** address (which addresses all hosts on the network) which is typically the last host address. For a class C address (used in the laboratories), this is **X.Y.Z.255**.

   The format of the request/packet contents is **IPOF:usernameyouwishobtaintheIPof**.
2. If a mote receives a broadcast UDP packet requesting their IP (i.e. the request matches our username), send out a UDP packet containing the username of the mote to the IP address which sent the request.
3. Upon receiving a USER reply packet, if the username contained in the packet matches the device we wish to communicate with, store the IP of the intended recipient (source IP) for future communication.

Task 1: Discovering other devices

1. Create a copy of your sketch from laboratory 2 task 5 for this task. Utilise the file naming convention given above and used throughout the module.
2. Use a macro (**#define**) to define **your** username.
   - string constant literals are defined as "username"
3. Use a macro (**#define**) to define **your neighbours** username.
4. Add the function **int checkbutton (void)** from laboratory 1 task 4 to your sketch.
5. Modify the **setup()** function of your sketch such that the top line of the LCD display displays the IP address assigned to your board upon connection.
   - You performed this task in the last laboratory 4 task 1.
6. If the button is pressed, send a UDP packet containing **IPOF:partnerusername**, where **partnerusername** is the username of your lab partner.
   - This is a request from your board to get the IP address of your lab partner's board. The destination IP of this packet must be the **broadcast** address to ensure all possible device addresses are checked.
   - A class instance of **IPAddress** is typically utilised to hold IP addresses in a similar fashion to an array, `IPAddress tempip = WiFi.localIP();`.
   - The last octet of IP address can be modified in the same way as an array, i.e. `tempip[3] = 255;`.
7. Modify your code such that upon detecting a UDP containing the string **IPOF:yourusername**, your board responds to the IP address and port with a string containing **USER:yourusername**.
   - This is best actioned in the same manner as laboratory 2, task 5.
   - The **sprintf()** function can be used to form the contents of the **outBUFFER** array, utilising the username defined by the macro earlier. The string format specifier is %s and more information on sprintf() can be found at http://www.cplusplus.com/reference/cstdio/sprintf/
8. Modify your code such that upon detecting a UDP containing the string **USER:partnerusername**, you store the packet's source IP address in an instance of the **IPAddress** class called **destip**.
9. Display the destination IP of your partner's board on the second line of the LCD display.
10. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.
11. Verify that your sketch works by the LCD displaying your IP address and that of your neighbours on the 2nd line when the button is pressed.

SHOW A LECTURER/DEMONSTRATOR TO BE SIGNED OFF.

Task 2: Generating message authentication codes from messages

Note that this task simulates the use of MACs by calculating the MAC code on a message and a simulated received message on the same board. This allows for the message to be easily tampered with or different keys experimented with.

1. Download the crypto library from **Learning Resources > Support Material > Board Information.**
2. Download the sketch **hmacstart.ino** from the **Learning Resources > Laboratory sessions > Laboratory 5: Board discovery and Message Authentication Codes (MACs).**
3. Save the sketch in an appropriate folder on your network drive.
4. Modify the sketch in the **setup()** function such that it calculates a MAC code from the **outMESSAGE** array and stores the result in the **HMACfromoutput** array using the **generateHMAC()** function supplied.
5. After the HMAC has been calculated on the **outMESSAGE** array, print the HMAC function generated using the **printHMAC()** function.
6. Modify the setup function such that it calculates a MAC code from the **inMESSAGE** array and stores the result in the **HMACfrominput** array using the **generateHMAC()** function supplied.
7. After the HMAC has been calculated on the **inMESSAGE** array, print the HMAC function generated using the **printHMAC()** function.
8. Use the **checkHMACS()** function to compare the two HMAC codes. If the codes match, print a message over serial indicating so. If the codes do not match, print a message indicating that the message has been tampered with.
9. Compile your sketch, debug any errors and upload the sketch to the Aston IOT hardware platform.
10. Verify that your sketch works by changing the incoming message, **inMESSAGE**.

SHOW A LECTURER/DEMONSTRATOR TO BE SIGNED OFF

Optional Task 3: Integrating HMACs into the UDP sketch

1. Modify the sketch from task 1 such that it sends a message to your lab partner with a MAC code appended at the end of a message. **This should occur when the button on the board has been pressed**.
2. Upon receiving the message, your lab partner's board should calculate the MAC from the message contained in the UDP packet and compare it against the one contained at the end of packet. If the MACs match, print the message over serial and tell the user the message is authentic and not tampered with. If not, inform the user that the message has been tampered with.
3. Experiment with tampering with the outgoing message, the array can be changed after the MAC has been computed.

Hints:

- Using a fixed length message will make this task significantly easier as the MAC can be copied to a fixed point in the buffer.
- Copying one array from another is best achieved using the **memcpy()** function. It can be used in the following manner:

```
memcpy(destinationarray,sourcearray,numberofbytestocopy);
```

Note that if you wish to copy 16 bytes to the middle of the 256 byte buffer, this could be achieved by:

```
memcpy(&outBUFFER[128],outputHMAC, SHA256HMAC_SIZE);
```

Note that we pass the **address** of outBUFFER element 128, as memcpy requires pointers or memory addresses to copy from and to.

- You will need some way of indicating that packets sent are a message. Messages should begin with **MSG:**
- You can copy the first 4 characters into a temporary char array of 5 characters long to check if a message is present in the input buffer. This can be achieved via Figure 2. Note that a null terminator is required (\0) as that is how C signifies the end of a string!

```
// check for message and MAC
for ( i = 0 ; i < 4 ; i++ )
  temp[i] = inBUFFER[i];
temp[4] = '\0';
if ( strcmp(temp,"MSG:") == 0)
```

Figure 2: Checking the first 4 characters