

Assignment III: End-point Authentication

Members: Beatriz Glaser
 Thuy Määttä

1. Goals of the experiment

One of the goals of this experiment is to develop a sketch that can scan for 802.11b/g/n networks and then display the information of the available networks and their authentication type on the Arduino IDE serial monitor. The experiment also aims to explore using Arduino boards in wireless communication and networking for innovative solutions. Lastly, it is important to understand different authentication methods used in 802.11 networks and how they work.

2. Experimental Setup

- a. For this experiment we used:
 - i. Arduino MKR WiFi 1010 board
 - ii. Micro USB cable
 - iii. Arduino IDE
 - iv. Library: WiFiNINA (<https://github.com/arduino-libraries/WiFiNINA>)
- b. Step-by-step setup:
 - i. Have a working Arduino with SAMD installed
 - ii. Connect the Arduino board and port to the IDE
 - iii. Install the above-mentioned library
 - iv. Create a new sketch
 1. Import the library
 2. Write a “setup” function to:
 - a. Establish serial communication between the Arduino and the computer via cable
 - b. Check if the WiFi module exists
 - c. Read and print the MAC Address of the board (can also be done on a separate function that is called on the “setup”)
 3. Implement code in a “loop” function where:
 - a. The program scans available WiFi networks and prints their SSID, BSSID, signal, and encryption type (every 10 seconds).
 - b. After scanning and printing three networks, it prompts the user to input a network name and password for connection.

- c. If the name and/or password are incorrect, it asks for new inputs.
- d. If name and password are correct, the connection is established and the program prints out the SSID, BSSID and signal strength of the network the arduino is connected to.
- v. Compile and upload code to the board
- vi. Test with the Serial Monitor

3. Results

- a. After uploading the code and running the Serial Monitor, the program prints the MAC address of the board and begins to scan networks.

```

Message (Enter to send message to 'Arduino MKR WiFi 1010' on 'COM7')
MAC Address: 64:8C:4C:D6:EB:4C
Scanning networks...

```

- b. The program then (every 10 seconds) starts printing available WiFi networks and their information.

```

Message (Enter to send message to 'Arduino MKR WiFi 1010' on 'COM7')
MAC Address: 64:8C:4C:D6:EB:4C
Scanning networks...

Network channel: dont even try it
BSSID: CE:89:B6:96:DC:5B
Signal: -35dBm
Encryption type: WPA2

Network channel: pb_network
BSSID: C0:C9:E3:38:97:E
Signal: -50dBm
Encryption type: WPA2

Network channel: TP-Link_C64E
BSSID: C0:6:C3:25:C6:4E
Signal: -59dBm
Encryption type: WPA2

```

- c. After 3 networks, it prompts the user for a network name and password. If either is incorrect, it asks for new inputs:

```
wrong
MAC Address: 64:8C:4C:D6:EB:4C

Scanning networks...

Network channel: dont even try it
BSSID: CE:89:B6:96:DC:5B
Signal: -41dBm
Encryption type: WPA2

Network channel: pb_network
BSSID: C0:C9:E3:38:97:E
Signal: -56dBm
Encryption type: WPA2

Network channel: AA-kerho
BSSID: 7C:10:C9:75:41:80
Signal: -56dBm
Encryption type: WPA2

Type the name of the network you would like to connect: dont even try it
Type the password:
```

- d. If either the name of password are incorrect, the program informs the user it was unable to connect and asks for new inputs:

```
Type the name of the network you would like to connect: dont even try it
Type the password: wrong

Network name or password incorrect! Try again.

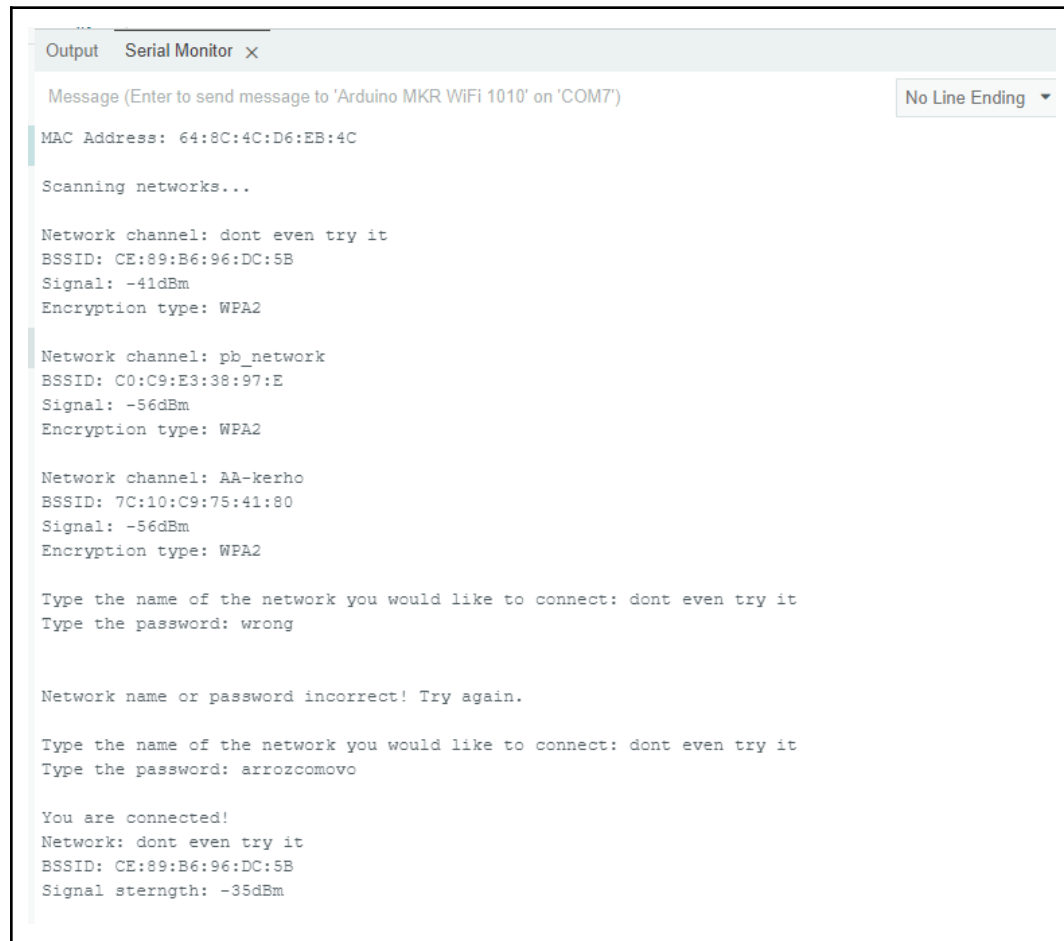
Type the name of the network you would like to connect:
```

- e. If the connection is successful, it informs the user and prints out information about the connected network.

```
Type the name of the network you would like to connect: dont even try it
Type the password: arrozcomovo

You are connected!
Network: dont even try it
BSSID: CE:89:B6:96:DC:5B
Signal strength: -35dBm
```

- f. The complete program:



```
Output Serial Monitor x
Message (Enter to send message to 'Arduino MKR WiFi 1010' on 'COM7') No Line Ending ▼
MAC Address: 64:8C:4C:D6:EB:4C
Scanning networks...
Network channel: dont even try it
BSSID: CE:89:B6:96:DC:5B
Signal: -41dBm
Encryption type: WPA2

Network channel: pb_network
BSSID: C0:C9:E3:38:97:E
Signal: -56dBm
Encryption type: WPA2

Network channel: AA-kerho
BSSID: 7C:10:C9:75:41:80
Signal: -56dBm
Encryption type: WPA2

Type the name of the network you would like to connect: dont even try it
Type the password: wrong

Network name or password incorrect! Try again.

Type the name of the network you would like to connect: dont even try it
Type the password: arrozcomovo

You are connected!
Network: dont even try it
BSSID: CE:89:B6:96:DC:5B
Signal strength: -35dBm
```

4. Answer to questions

- a. Which authentication methods did you find for 802.11?
 - i. Open System Authentication
 - ii. Shared Key Authentication
 - iii. Wired Equivalent Privacy (WEP)
 - iv. Wi-Fi Protected Access (WPA)
 - v. Wi-Fi Protected Access 2 (WPA2)
 - vi. Wi-Fi Protected Access 3 (WPA3)
- b. Describe three authentication methods in detail.
 - i. Open System Authentication

This is the most basic form of authentication in 802.11. It does not provide any security, as any client can join without providing any credentials. Open system authentication consists of 2 communications. First, an

authentication request is sent from a client to the AP. Then, an authentication response is sent back from the AP.

ii. Wi-Fi Protected Access (WPA)

WPA is a security enhancement over WEP, using TKIP (Temporal Key Integrity Protocol) encryption and supporting two methods: WPA-PSK and WPA-Enterprise.

WPA-PSK uses a pre-shared key (256-bit) and four-way handshake for client authentication, while WPA-Enterprise uses an authentication server, like RADIUS, to verify client credentials.

WPA-PSK is used in small/home networks, while WPA-Enterprise is used in larger networks with a centralized authentication infrastructure.

iii. Wi-Fi Protected Access 2 (WPA2)

WPA2 is an improvement from WPA, which offers stronger security and uses AES encryption with two authentication methods: WPA2-PSK and WPA2-Enterprise.

WPA2-PSK uses a robust key exchange (with the use of AES algorithm) and a four-way handshake for client authentication (exchange and verify the pre-shared key before granting the client access to the network).

WPA2-Enterprise uses an authentication server (for example, RADIUS server) with EAP (Extensible Authentication Protocol) to support various authentication methods, such as EAP-TLS, EAP-TTLS, EAP-PEAP, etc.

c. Describe briefly applications scenarios for these methods.

i. Open System Authentication (OSA)

Public Wi-Fi Hotspots, for example in airports, libraries, etc. often use OSA. The main goal is that the guests can access the internet easily without passwords. However, it's important to note that OSA provides no security or encryption, making it vulnerable to attacks.

ii. Wi-Fi Protected Access (WPA)

Even though WPA is mostly replaced by WPA2 and WPA3, here is an example of its applications in the past.

Home network: WPA-PSK (Pre-Shared Key) was used in home networks to provide a reasonable level of security and privacy for data transmission. Users connected their devices to the Wi-Fi network by entering a shared password. WPA-PSK was easier to set up and manage compared to WEP, while still offering improved security.

Additionally, WPA with TKIP (WPA-TKIP) is the default choice for old routers that don't support WPA2.

iii. Wi-Fi Protected Access 2 (WPA2)

Public Wi-Fi Hotspots: In some public spaces (such as malls, coffee shops, etc.), WPA2 can be used to secure Wi-Fi networks while still providing accessibility to the public. For example, a coffee shop may use a WPA2-PSK network with a password displayed to customers. This ensures some level of security while allowing only the shop's customers to connect easily.

5. Bonus (brute forcing passwords)

a. Structure of code (4-number password):

- i. "setup" function, that establishes serial communication between the Arduino and the computer via cable
- ii. "loop" function, which establishes password length and calls the brute force function
- iii. "brute force" function, which is a recursive function responsible for creating possible password strings and attempt to connect to the networking using them

b. Practicalities

- i. The WiFi hotspot on the phone used in this experiment has a minimum 8 digits requirement, so a template "aaaa" is used in the beginning of the password and trials. So the last 4-digits are the ones being trialed/guessed.
- ii. The four digits used were numbers (0-9), in order to facilitate the cracking and diminish time constraints.
- iii. Each possible 4-number sequence is concatenated with the default template "aaaa". Then the program attempts to connect to the network "dont even try it" with possible password. The serial monitor prints "Connected!" if the password is correct and connection is successful, and "Wrong password" otherwise.

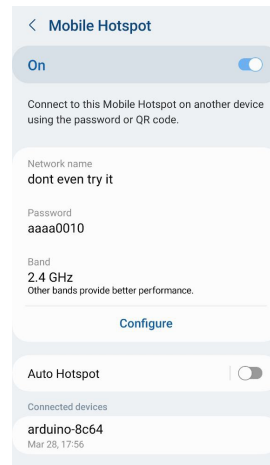
c. Results

- i. After uploading and running the code, the serial monitor begins to try the combinations and attempts to connect to the network

```
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
```

- ii. Once the correct password is tried, the connection is successful! The serial monitor prints it, and the connection can also be confirmed on the phone.

```
wrong password:
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Wrong password!
Connected!
```



- iii. I had initially tried to create this program with a password with a different number sequence such as “aaaa6728”. However, the code tries different passwords in order according to the character sequence (0-9), therefore it would have taken hours to reach the correct one. This proves how inefficient and impractical this method and resources are when it comes to decoding passwords.

d. Questions

- i. *How long does a single guess take?*

It takes 5 seconds for each guess to be generated and tested.

- ii. *How long does it take for it to guess the password?*

The time taken for the password to be guessed depends on the password itself. Guessing the password “0010” took 55 seconds. However, testing

all possible 4-number combinations would take a total of 10^4 possible combinations times the time taken for each guess (5). Therefore a total of 50.000 seconds (or approximately 14 hours).

If we were considering letters, there are 26 characters. So the time to check every possible combination would be $(5 * 26^4 =) 2.284.880$ seconds (or 26,4 days).

iii. What about if the password was 5, 6, 8, 12 letters?

- 5 digits, then $5 * 26^5 = 59.406.880$ seconds = 687,6 days
- 6 digits, then $5 * 26^6 = 1.544.578.880$ s = 17.877 days
- 8 digits, then $5 * 26^8 = 1.044.135.322.880$ s = 12.084.899,6 days
- 12 digits, then $5 * 26^{12} = 4.77e17$ s = 362.875.338.809 years

Testing more numbers might also increase the testing time for each guess. However, most of the time taken comes from the connection attempt (not the formation of a new sequence). Therefore the millisecond second difference would not be significant given the magnitude of the possible combinations/time taken.

iv. How about when adding numbers and symbols?

Adding numbers and symbols would increase the possible character combinations. There are 10 possible numbers (0-9) and 33 special characters (!"#\$%&'()*+,-./:;<=>?@[\\]^_`{|}~). If we also add capital letters, there is a total of $26 + 26 + 10 + 33 = 95$ character sequence.

For a 4 digit password, it would take $5 * 95^4$ seconds. For a 5 digits one, $5 * 95^5$ seconds. And so on.

v. How to reduce the time to break the password?

How long it takes to crack a password using brute force algorithm depends on its length, character types, and letter case. Additionally, using CPU and GPU with higher speed and capability can reduce the time to crack the password. Other techniques/algorithms are also used to hack passwords.

vi. How to prevent this method/was it prevented by your router/phone?

There are several ways to prevent a brute force attack, on both routers and personal devices (such as phones):

- Using strong passwords, creating passwords that are not easy to guess, a strong password should be a combination of small letters, capital letters, numbers, special symbols and not a common phrase or word.
- Limit login attempts: many routers and devices have a built-in security feature that limits the number of login attempts.
- Using two-factor authentication: this method adds an extra layer of security by requiring a second factor such as a code being sent to your phone
- Using firewall: firewalls can eliminate unauthorized access

6. Conclusion

In conclusion, we learned how to use an Arduino board to scan for available WiFi networks, check their information and connect to one. We also learned about different authentication methods and their importance in order to maintain security. Overall, it is useful, on both personal and professional levels, to understand about authentication methods and their applications.

7. Annex

a. Source code implemented (network scanning and connecting):

```

authentication.ino
1 #include <SPI.h>
2 #include <WIFI_NINA.h>
3
4 void setup() {
5   Serial.begin(9600);
6
7   while (!Serial);
8
9   // check for the presence of the WiFi module
10  if (WiFi.status() != WL_NO_MODULE) {
11    Serial.println("Communication with WiFi module failed!");
12    while (true);
13  }
14
15  // print the MAC address of the board
16  printMacAddress();
17
18
19  void loop() {
20
21    Serial.println("\nScanning networks...");
22
23    int numSsid = WiFi.scanNetworks();
24    if (numSsid == -1) {
25      Serial.println("Couldn't get a wifi connection");
26      while (true);
27    }
28
29    // print info about three networks
30    int num_net = 0;
31    while (num_net < 3) {
32      Serial.println();
33      Serial.print("Network channel: ");
34      Serial.println(WiFi.SSID(num_net));
35
36      byte bssid[6];
37      WiFi.BSSID(num_net, bssid);
38      printBSSID(bssid);
39
40      Serial.print("Signal: ");
41      Serial.print(WiFi.RSSI(num_net));
42      Serial.println("dBm");
43      Serial.print("Encryption type: ");
44      Serial.println(WiFi.encryptionType(num_net));

```

```

authentication.ino
45
46 // make sure printing happens 10 seconds appart
47 delay(10000);
48 num_net++;
49 }
50
51 int status = WL_IDLE_STATUS;
52 int loop_count = 0;
53 while (status != WL_CONNECTED) {
54     // If user already tried to connect once
55     if (loop_count != 0) {
56         Serial.println("\nNetwork name or password incorrect! Try again.");
57     }
58
59     // prompt user for a network and password
60     Serial.print("\nType the name of the network you would like to connect: ");
61     while (Serial.available() == 0) {}
62     String ssid_s = Serial.readString();
63     int ssid_len = ssid_s.length();
64
65     Serial.println(ssid_s);
66
67     Serial.print("Type the password: ");
68     while (Serial.available() == 0) {}
69     String pass_s = Serial.readString();
70     int pass_len = pass_s.length();
71
72     Serial.println(pass_s);
73
74     const char* ssid = ssid_s.c_str();
75     const char* pass = pass_s.c_str();
76
77     // attempt connection
78     status = WiFi.begin(ssid, pass);
79
80     Serial.println();
81     loop_count++;
82 }
83
84 Serial.println("You are connected!");
85 Serial.print("Network: ");
86 Serial.println(WiFi.SSID());
87

```

```

authentication.ino
88 byte con_bssid[6];
89 WiFi.BSSID(con_bssid);
90 printBSSID(con_bssid);
91
92 Serial.print("Signal strength: ");
93 Serial.print(WiFi.RSSI());
94 Serial.println("dBm");
95
96
97 }
98
99 void printMacAddress() {
100     byte mac[6];
101
102     // Get the MAC address
103     WiFi.macAddress(mac);
104
105     Serial.print("MAC Address: ");
106
107     // Print the MAC address in hexadecimal format
108     for (int i = 0; i < 6; i++) {
109         Serial.print(mac[i], HEX);
110
111         // Add colons between the MAC address bytes
112         if (i < 5) {
113             Serial.print(":");
114         }
115     }
116
117     Serial.println();
118 }
119
120 void printBSSID(unsigned char* bssid) {
121     Serial.print("BSSID: ");
122
123     // print bssid in hexadecimal format
124     Serial.print(bssid[5], HEX);
125     Serial.print(":");
126     Serial.print(bssid[4], HEX);
127     Serial.print(":");
128     Serial.print(bssid[3], HEX);
129     Serial.print(":");
130     Serial.print(bssid[2], HEX);
131     Serial.print(":");

```

```

132     Serial.print(bssid[1],HEX);
133     Serial.print(":");
134     Serial.println(bssid[0],HEX);
135 }
136
137 void encryptionType(int type) {
138     switch (type) {
139         case ENC_TYPE_WEP:
140             Serial.println("WEP");
141             break;
142         case ENC_TYPE_TKIP:
143             Serial.println("WPA");
144             break;
145         case ENC_TYPE_CCMP:
146             Serial.println("WPA2");
147             break;
148         case ENC_TYPE_NONE:
149             Serial.println("None");
150             break;
151         case ENC_TYPE_AUTO:
152             Serial.println("Auto");
153             break;
154     }
155 }
156

```

b. Code to brute force a 4 number pincode.

```

brute_force.ino
1  #include <SPI.h>
2  #include <WiFiNINA.h>
3
4  char charset_main[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
5
6  int len = sizeof(charset_main)/sizeof(char);
7
8  char network[] = "dont even try it";
9  // password = "aaaa0010" (for reference)
10
11 void setup() {
12
13     Serial.begin(9600);
14
15     while (!Serial);
16
17 }
18
19 void loop() {
20
21     //n is the length of the password
22     int n = 4;
23     bruteForce(charset_main, "", len, n);
24 }
25

```

brute_force.ino

```
26 void bruteforce(char charset[], String prev, int len, int n){
27     // check if n == 0
28     if (n == 0){
29         //Serial.println(prev);
30         char trial[sizeof(password)+1];
31         strcpy(trial, "aaaa");
32
33         const char* p = prev.c_str();
34         strcat(trial, p);
35
36         //Serial.println(trial);
37         int status = WL_IDLE_STATUS;
38
39         if (status == WL_CONNECTED) {return;}
40
41         status = WiFi.begin(network, trial);
42         if (status != WL_CONNECTED) { Serial.println("Wrong password!");}
43         else {Serial.println("Connected!");}
44
45         return;
46     }
```

```
47     //for each character in charset
48     int i;
49     for (i = 0; i < len; i++){
50         //convert previous charset to String
51         String newprev = prev + charset[i];
52         // decrement n and recursively call bruteforce
53         bruteforce(charset, newprev, len, n - 1);
54     }
55 }
56
```