ESP32 Wifi Tests

Introduction

ESP32 is a powerful microcontroller with built-in WiFi and Bluetooth. It can be programmed using the Arduino framework, which is on top of the ESP-IDF framework. The latter is the official framework for ESP32 development. Code is usually written in C, but Arduino defaults to C++ and it a bit higher level.

Development Setup

For this project, I have chosen to use Arduino. I will still use functions from the ESP-IDF framework, though!

For editing, I use Visual Studio Code with the Arduino extension. I also use the PlatformIO extension, which is a great tool for managing libraries and boards.

Getting started

Wifi modes

ESP32 can be in one of the following modes:

Mode	Description	Description
`WIFI_STA`	station mode	ESP32 can scan and connect
`WIFI_AP`	access point mode	stations can connect to the ESP32
`WIFI_AP_STA`	access point and station mode	access point and a station connected to another access point

Station mode

I configured ESP to be in station mode and scanned all networks. This was the result:

```
3 networks found:
1: A2 (-69) WPA2_PSK
2: Casa Anca  (-78) WPA_WPA2_PSK
3: Casa  ANCA (-82) WPA_WPA2_PSK
```

I was interested in the following information:

```
<ID> <SSID> <MAC / BSSID> <channel> <encryption>
```

I scanned again, with the above information in mind:

```
4 networks found:
A2 74:4D:28:36:01:60 1 WPA2_PSK
Casa Anca  44:00:4D:51:FD:AC 9 WPA_WPA2_PSK
Casa  ANCA 68:FF:7B:F0:23:5D 5 WPA_WPA2_PSK
DIGI-3J9P D0:C6:5B:08:79:20 11 WPA_WPA2_PSK
```

Awesome! I can get all the necessary information to clone an **open** network (more on that later...). Let's beautify the output a bit:

```
Scan done! ✅
22 networks found:
ID  SSID                              MAC                CH  ENC
-------------------------------------------------------------------------
 1  A2-c                              B8:DD:71:E5:C2:63   9  WPA_WPA2_PSK
 2  DIGI-5tyt                         04:20:84:12:B7:1F   1  WPA2_PSK
 3  ASUSHN                            F0:79:59:EB:A2:10   4  WPA2_PSK
 4  AndreiNinaFlorin                  F8:1A:67:DE:E5:08   1  WPA2_PSK
 5  nero2.4ghz                        F0:79:59:EB:A2:13   4  WPA2_PSK
 6  BodyPriority                      80:8C:97:73:D2:B0   1  WPA2_PSK
 7  ASUSHN2                           F0:79:59:EB:A2:11   4  WPA_WPA2_PSK
 8  ASUS_Guest                        F0:79:59:EB:A2:12   4  WPA2_PSK
 9  TP-Link_94FA                      60:32:B1:FE:94:FA   9  WPA2_PSK
10  Kasem2                            80:8C:97:74:29:1D   1  WPA2_PSK
11  DIGI_4312d0                       6C:A4:D1:43:12:D0   1  WPA2_PSK
12  Tenda_8CD120                      58:D9:D5:8C:D1:21   4  WPA_WPA2_PSK
13  Raul                              04:95:E6:8B:5C:C1   1  WPA_WPA2_PSK
14  DIGI-Q635                         40:7D:0F:95:E8:40   2  WPA_WPA2_PSK
15  DIGI-93sx                         DC:21:E2:8E:A0:F8   4  WPA_WPA2_PSK
16                                    66:32:B1:FE:94:FA   9  WPA2_PSK
17  Sitaru                            C8:3A:35:04:BE:E0  10  WPA_PSK
18  NSA Surveillance Hub Zorilor      C8:B3:73:07:90:9A   1  WPA_WPA2_PSK
19  ROS                               D8:47:32:51:BE:BA   1  WPA2_PSK
20                                    DE:07:B6:8A:4B:6B  12  WPA2_PSK
21  DIGI_f4f4f4                       F8:64:B8:F4:F4:F4   9  WPA_WPA2_PSK
22  TP-Link_4B6B                      D8:07:B6:8A:4B:6B  12  WPA_WPA2_PSK
```

Okay, the scanning part is done! Moving on...

Access point mode

> TODO: Write stuff here

Station + Access point mode

This mode is a combination of the previous two. The ESP32 acts as an access point and a station at the same time. This means that it can connect to another access point and other stations can connect to it.

Attacks

Deauthentication attack

One of the most annoying attacks is the **deauthentication attack**. It is a DoS (Denial of Service) attack. What does it do? It disconnects a device from a network. It is also known as a **deauth attack** or **deauth flood**.

Usages:

- **Evil Twin attack**: disconnect a device from a network and create a fake network with the same name. This way, one can force the client to connect to a rogue access point.
- to get the **hash of the password** of a network. When a device connects to a network, it sends the password in the form of a hash. When the client will try to reconnect, one can capture the handshake containing the hash of the password. Later, the password can be cracked using, for instance, a dictionary attack or a rainbow table.

To implement this, I have written a code which builds the necessary packet and sends it to the target. It looks like this:

```
uint8_t deauthPacket[26] = {
    /*  0 - 1  */ 0xC0, 0x00,                         // type, subtype c0: deauth (a0: disassociate)
    /*  2 - 3  */ 0x00, 0x00,                         // duration (SDK takes care of that)
    /*  4 - 9  */ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // reciever (target)
    /* 10 - 15 */ 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, // source (ap)
    /* 16 - 21 */ 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, // BSSID (ap)
    /* 22 - 23 */ 0x00, 0x00,                         // fragment & squence number
    /* 24 - 25 */ 0x01, 0x00                          // reason code (1 = unspecified reason)
};
```

The packet is sent to the target using the `esp_wifi_80211_tx` function.

Now, theoretically, if I upload this code, it *should* work, right? Well, yes, but no.

The ESP32 does a **sanity check** when sending packets. This check examines the frame's structure, header information, payload, and other essential components to confirm that they adhere to the expected format and content.

One more thing is does is protect against malicious packets. It's an important aspect of such checks primarily because it helps in preventing the processing of malicious or malformed frames that could be used for attacks like packet injection, deauthentication, or other network-based attacks. This is a good thing, but it also means that I cannot send the packet I have built. The ESP32 will not allow it. Let's find a way to bypass this check!

Bypassing the Sanity Check

To bypass the sanity check and successfully send the deauth packet, I took the following steps:

1. **Redeclaring the Function**: I redeclared the `ieee80211_raw_frame_sanity_check` function, which is responsible for performing the sanity check on raw IEEE 802.11 frames. This allowed me to `return 0;` whenever the function was called, effectively bypassing the sanity check.
2. **Linker Flags (-Wl,-zmuldefs)**: In my `platformio.ini` configuration file, I added the linker flags `-Wl,-zmuldefs`. These linker flags help in resolving multiple definitions of functions or symbols during the linking process, allowing the redefined `ieee80211_raw_frame_sanity_check` to take precedence over the original definition. Otherwise, the compiler would throw an error.

Now it works! It successfully sends the deauth packet to the target and disconnects it from the network! Awesome! 🎉

A bit of theory

SSID

SSID (Service Set Identifier) is the name of the access point. This is what you see when you scan for networks on your computer or phone. It is a string of up to 32 characters. It is *cAsE sEnSiTiVe*.

MAC address

Generally, a **MAC address** is a assigned to a device by the manufacturer. It is a unique identifier for that device. Similarly, in networking, a MAC address (aka **physical address** or **hardware address**) is a unique identifier assigned to network interfaces for communications on the physical network segment. It is a 48-bit number (12 hexadecimal characters). The first 6 characters are the vendor ID. The last 6 characters are the serial number of the device.

BSSID

BSSID stands for Basic Service Set Identifier. It is the MAC address of the access point.

Channel

The **channel** is the frequency at which the access point is broadcasting. It is a number between 1 and 14.

Each channel is 5 MHz apart, except for channel 14, which is 12 MHz apart from channel 13. The 2.4 GHz band is divided into 14 channels. However, in most countries, only channels 1-11 are available for use. Channels 12 and 13 are available in some countries, but channel 14 is not available for use.

Resources

Theory

- List of WLAN Channels

ESP32

- ESP32 Wroom-32 Datasheet
- ESP32 API - WiFi Driver (Espressif docs)
- ESP32 AP Configuration (Read the Docs)
- ESP32 WiFi Functions (randomnerdtutorials)
- ESP32 WiFi MAC configuration
- risinek's Master Thesis

4-Way handshake

- 4-Way Handshake (NordVPN's glossary)

Deauth

- Wikipedia: Deauth attack
- WiFi Deauth Frame
- ESP8266 Deauther

Sanity check bypass

- Reddit post
- ESP32 Marauder Arduino IDE Setup
- PIO build_flags