Rogue Wi-Fi AP detection with Snappy

Detry Pierrick

CYBER DEFENCE LAB

November 24, 2024

Rogue Wi-Fi AP detection with Snappy
Detry Pierrick
November 24, 2024

Cyber Defence Lab
https://cylab.be

BibTeX citation:

```
@techreport{citekey,
title = {Rogue Wi-Fi AP detection with Snappy},
author = {Detry Pierrick},
institution = {Cyber Defence Lab},
year = {2024},
month = {11},
}
```

Acknowledgements

I would like to thank the Cyber Defence Lab for offering me this internship opportunity and for welcoming me. Thanks also to Mr Debatty for being my academic supervisor and for guiding me.

Abstract

Open Wi-Fi networks are widely used but pose significant risks as malicious actors can create fake access points to capture user traffic. Snappy, a Python script developed by SpiderLabs, aims to fingerprint and detect rogue Wi-Fi access points. This report evaluates its effectiveness in identifying and mitigating these threats.

Keywords: Snappy, Wi-Fi, Open Wi-Fi, rogue access point

Contents

1. Background

1.1. Wi-Fi connection process

The IEEE 802.11 standard defines three types of frames that enable wireless communication:

- **Management frames**: Used to establish and maintain the Wi-Fi network.
- **Control frames**: Facilitate access to the wireless medium, such as coordinating transmissions through acknowledgments.
- **Data frames**: Carry payload data, encapsulating higher-layer protocol information, such as IP packets or application data.

The process of connecting to a Wi-Fi network relies heavily on the **management frames**. This connection process consists of three key steps: (1) device discovery (or scanning), (2) authentication (where compatibility and security requirements are verified), and (3) connection establishment (association). A graph of the message exchange involved in this connection process is shown below[1].

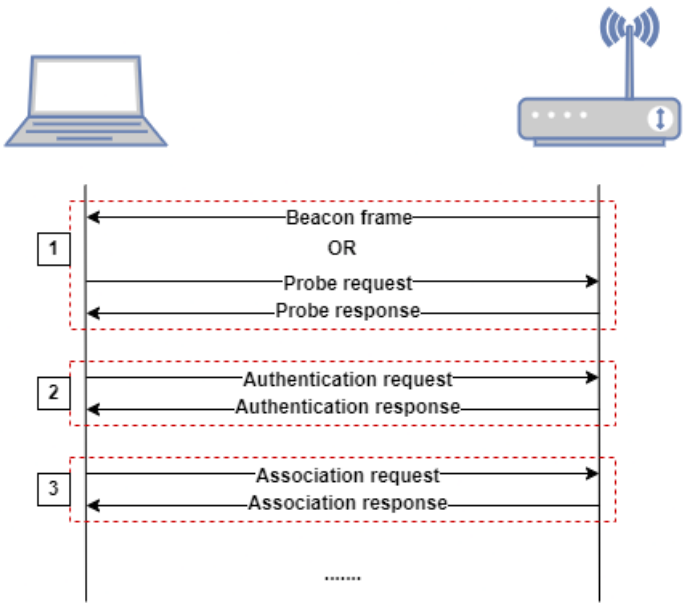

Figure 1.1. Wi-Fi connection process

1. **Scanning**: To determine if compatible Wi-Fi networks are available around a client, the client initiates a scanning phase, which can be either passive or active.

   - *Passive*: In passive scanning, the client switches between channels, listening for beacon frames. These beacon frames are sent by Access Points (APs) to announce their presence and include values like SSID, BSSID, supported rates, and more, as shown in the figure below:

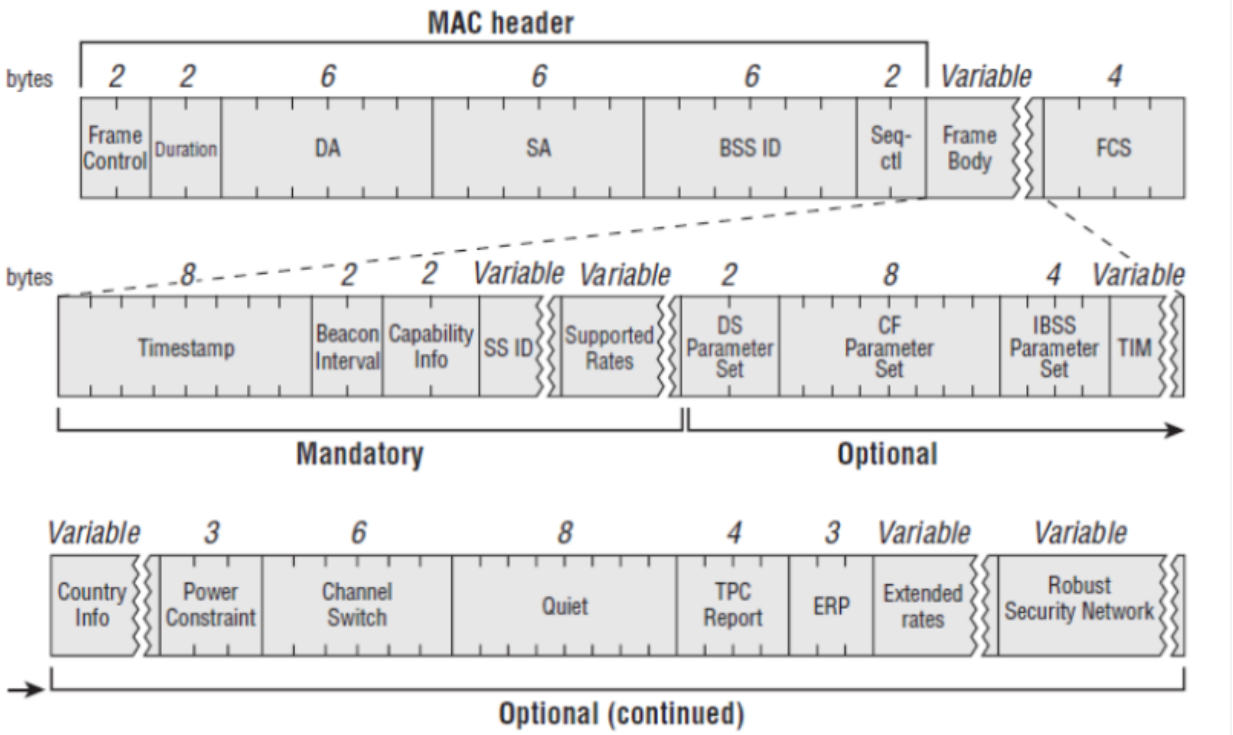

Figure 1.2. Beacon frame structure[2]

We will now explain the beacon frame structure in detail[3], as it is a fundamental part of the topic discussed in this report.

- Mandatory headers
  (a) Timestamp: The number of microseconds the AP has been active. The timer resets after reaching its maximum value (2^{64}).
  (b) Beacon Interval: Time interval between two consecutive beacon transmissions. It is measured in time units (TU), where 1 TU equals 1024 microseconds.
  (c) Capability Information: This field contains the number of subfields that are used to indicate requested or advertised optional capabilities.
  (d) SSID: The Service Set Identifier is the network name, which can be up to 32 characters long.

(e) Supported Rates: The data rates supported by the AP. Each data rate is represented in units of 500 kbps. The last bit of each value indicates whether the rate is mandatory (basic) or optional.

- Optional headers
  (a) DS Parameter: Specifies parameters for networks utilizing Direct Sequence Spread Spectrum (DSSS) technology, such as the channel number.
  (b) Country: This specifies the regulatory domain (country) where the AP operates, along with the allowed channels and maximum power levels.
  (c) Extended Supported Rates: If the AP supports more than 8 data rates, this field lists the additional ones not included in the mandatory "Supported Rates" field.
  (d) Robust Secure Network: This field provides details about the security settings of the AP, such as encryption protocols (e.g., WPA2, AES) and authentication methods.

- *Active*: Passive scanning can sometimes be inefficient since it requires waiting on each channel. In active scanning, the client still switches channels but sends a probe request on each channel. This request can be either broadcasted to all APs or directed to a specific SSID. The client waits briefly for a probe response, allowing for faster scanning compared to the passive method.

2. **Authentication**: The next step for the client is the authentication process. The client sends an authentication request to verify compatibility with the AP's capabilities. If the AP accepts the request, it responds with an authentication frame indicating "success".

3. **Association**: The final phase of the connection process is association. In this step, the client joins the network and is assigned an Association ID (AID). The client sends an association request, and the AP checks the parameters to confirm compatibility. If they match, the AP replies with an association response, completing the connection process.

1.2. 802.11 Wireless adapter modes

Wireless network adapters can operate in several different modes, each serving a unique purpose in managing and interacting with network traffic. This section describes some of the most common modes:

- **Managed Mode**: This is the default mode for most wireless adapters and is used for typical client connections. In managed mode, the adapter connects to a specific AP and only communicates with that AP.
- **Ad-Hoc Mode**: In ad-hoc mode, devices communicate directly with each other without the need for an AP.
- **Access Point Mode**: In AP mode, the wireless adapter acts as an AP itself, allowing other devices to connect to it.
- **Monitor Mode**: Monitor mode is essential for wireless network analysis and is frequently used in security testing and research. In this mode, the wireless adapter can capture all traffic within range on a specific channel, regardless of whether it is addressed to the adapter.
  - There is often confusion between *monitor mode* and *promiscuous mode*. As we explained, monitor mode captures raw wireless traffic on all nearby networks, whereas promiscuous mode is used for capturing traffic within an associated wired or wireless network, including packets not directly addressed to the device.
- **Promiscuous Mode**: Unlike the operational modes described above, promiscuous mode is not a separate mode but rather a feature or configuration that can be enabled within certain modes. It allows a wired or wireless network interface controller to capture and process all traffic it receives, regardless of whether the frames are addressed to the device.

1.3. Examples of Wi-Fi attacks

One of the most common Wi-Fi attacks is the **Evil Twin** attack. In this attack, an attacker sets up a rogue AP that impersonates a legitimate network by cloning the SSID and other identifiers. Victims connect to the rogue AP, assuming it is the trusted network. Once connected, the attacker can intercept and manipulate the victim's network traffic or scanning the victim's device for vulnerabilities. This attack is highly effective in public places, where users often connect to open networks.

The **KARMA** attack leverages clients' tendency to automatically connect to previously known networks. When a device sends out probe requests to find known networks, a rogue AP can respond to these requests, masquerading as one of the networks on the device's preferred list. The victim unknowingly connects to the attacker's AP, making it possible for the attacker to intercept sensitive data or launch further attacks.

2. Snappy

Snappy[4] is a python script designed to detect rogue wireless access points (APs) by taking snapshots of the environment and comparing them over time. It achieves this by generating SHA256 hashes of various attributes of wireless access points, allowing it to detect whether any changes have occurred since the last snapshot. This is particularly useful for identifying rogue APs, which may attempt to impersonate legitimate networks by mimicking their SSID or other network attributes.

In addition, Snappy can be used to directly detect the use of airbase-ng which is often used to create rogue access points.

2.1. How Snappy works

Snappy works by capturing specific attributes from the beacon frame of each visible AP and hashing them using the SHA256 algorithm. By storing these hashes, it can compare the current environment with previous scans to detect changes. If a hash mismatch is found, this indicates that some attribute of the AP has changed, which could potentially indicate a rogue AP or network manipulation.

The attributes used to generate the hash include:

- **BSSID:** The Basic Service Set Identifier is the MAC address of the access point.
- **SSID:** The Service Set Identifier is the name of the wireless network.
- **Channel:** The channel on which the wireless network operates.
- **Country:** The country code determines the regulatory domain in which the AP is operating, which can affect factors like allowed frequencies and power levels.
- **Rates:** The supported data rates of the access point.
- **Extended Rates:** Additional supported data rates.
- **Max Transmit Power:** The maximal power given the Country.
- **Capabilities:** The capabilities of the device/network.

- **Max AMSDU (Aggregated MAC Service Data Unit):** This refers to the maximum frame aggregation size supported by the AP.
- **Vendor-Specific Information:** Some APs may broadcast vendor-specific information in their beacons, which could provide additional insights into the device's manufacturer.

2.2. Rogue access point detection

By creating a hash of these attributes, Snappy should detect any changes in the environment. If a rogue AP is introduced, even if it attempts to mimic the SSID and some capabilities of the legitimate AP, the hash will not match because certain details like the BSSID, channel, or supported rates are likely to differ.

Additionally, Snappy should detect attacks where an attacker uses the tool airbase-ng to create a rogue AP because the tool has hard-coded values for attributes such as rates, extended rates, country code, and vendor information, which can make it easier to identify if the access point was generated using airbase-ng.

2.3. Snappy script improvement

During the analysis and testing of the Snappy script, a few potential improvements were identified.

- **Correcting channel detection**: At line 20 of the Snappy code, the following line can lead to occasional errors during the parsing of the Wi-Fi frames:

```
channel = int(ord(frame[Dot11Elt:3].info))
```

  This line can be replaced by the following code to directly retrieve the channel from the frame object, avoiding potential parsing issues:

```
channel = frame.channel
```

  This avoids potential errors and improves code readability.

- **Duplicate Vendor Value**: Another small correction, at line 55, the vendor's value is repeated twice.

```
# line with duplicate value:

all=frame.addr3+str(frame.info)+str(channel)+str(country)+str(vendor)+
    str(frame.rates)+str(erates)+str(power)+str(cap)+str(htmax)+str(
    vendor)
```

As we can see, one "str(vendor)" can be removed.

- **Extended Rates**: In the original code, the method used to retrieve the Extended Rates incorrectly captures only the normal rates in a different format. The correct approach to retrieve the Extended Rates is:

```
frame.getlayer(Dot11Elt, ID=50).rates
```

A pull request of these modifications have been submitted and can be found here: https://github.com/SpiderLabs/snappy/pull/2

3. Lab

In this chapter, we conduct a lab experiment to evaluate whether Snappy can accurately detect rogue access points. The experiment involves setting up a legitimate AP and then creating rogue APs that attempt to mimic it by spoofing various parameters. We will examine if the generated hashes for the rogue APs match the hash of the legitimate AP.

3.1. Setup

The equipment used for this lab includes:

- 2 TP-Link EAP650
- 1 TP-Link TL-WN725N Wi-Fi adapter (2 if you want to try Airgeddon)
- A virtual machine running Kali Linux on VMware Workstation Pro

3.1.1. Legitimate access point

In this section, we configure the legitimate access point, which we will later attempt to spoof. We use a TP-Link EAP650, as shown in the figure bellow:

Figure 3.1. TP-Link EAP650

To set up the AP, plug in power and reset it by pressing the reset button until the light flashes. A Wi-Fi network with a name like *"TP-Link_2.4GHz_XXXXXX"* will appear; connect to it and access the configuration page via http://tplinkeap.net/.

Listing 3.1. Potential issue

```
Sometimes, when connecting to the Wi-Fi network, the computer does not
   receive an IP address from DHCP. If this happens, set an IP manually:
IP : 192.168.0.200
Netmask : 255.255.255.0
Gateway : 192.168.0.254
```

Use the default username and password (*admin/admin*) to log in, then create an SSID. In this case, we use an open network named *test_snappy*.

3.1.2. Tools

To complete this lab, we need some tools. In this section, we will describe the tools and how to install them. If you use a Kali Linux virtual machine, all the tools are already pre-installed.

- **Aircrack-ng**: Aircrack-ng is a suite of tools used to assess the security of Wi-Fi networks. It allows users to monitor, capture, and analyze wireless traffic, as well as perform various attacks on Wi-Fi encryption (WEP/WPA/WPA2). The most commonly used tools within the suite include:
  - aircrack-ng: Used to crack WEP and WPA/WPA2-PSK keys once sufficient data has been captured.
  - airodump-ng: Used to capture network traffic and display information about nearby access points and clients.
  - aireplay-ng: Used to inject packets into a network, useful for deauthentication or capturing handshakes.
  - airmon-ng: Used to enable monitor mode on a wireless interface.
  - airbase-ng: Used to set up rogue access points for testing or penetration purposes.

  You can download andinstall Aircrack-ng by using the following command:

```
sudo apt-get install aircrack-ng
```

- **Wireshark**: Wireshark is a popular network protocol analyzer used to capture and interactively browse traffic running on a computer network. It can dissect a wide variety of protocols, including 802.11 frames, making it useful for analyzing Wi-Fi traffic in detail. Wireshark provides a graphical interface that allows users to apply filters, view packet details, and troubleshoot network issues.

  You can download Wireshark by using the following command:

```
sudo apt-get install wireshark
```

3.1.3. Wi-Fi adapter

We use the TP-Link TL-WN725N Wi-Fi adapter. If the adapter does not appear after plugging in, you may need to install the correct driver. Check if it is detected with:

```
lsusb
```

You should see a line like *"Realtek Semiconductor Corp. RTL8188EUS 802.11n Wireless Network Adapter"*.If detected but not working, install the appropriate driver[5]:

```
git clone https://github.com/aircrack-ng/rtl8188eus.git
cd rtl8188eus
make && sudo make install
echo 'blacklist r8188eu' | sudo tee -a '/etc/modprobe.d/realtek.conf'
echo 'blacklist rtl8xxxu' | sudo tee -a '/etc/modprobe.d/realtek.conf'
```

After rebooting, the adapter should work.

If you encounter a "missing dependencies" error when running "make && sudo make install," use the following commands to install the required tools and headers:

```
sudo apt update
sudo apt install bc build-essential dkms linux-headers-$(uname -r)
```

3.1.4. Monitor mode

As explained before, to capture 802.11 frames, we need to put our network interface into a specific mode called *monitor* mode. Not all network interfaces support this mode, but you can check if yours does by running the following command:

```
iw list | grep "Supported interface modes" -A 8
```

Using the TP-Link TL-WN725N Wi-Fi adapter, which supports monitor mode. To enable this mode, run the following commands:

```
sudo airmon-ng check kill # To remove potential conflicts
sudo ip link set %INTERFACE% down
sudo iw dev %INTERFACE% set type monitor
sudo ip link set %INTERFACE% up
```

You can verify if monitor mode is working by running the following command:

```
sudo aireplay-ng --test %INTERFACE%
```

Optionally, you can also specify the channel on which you want your interface to operate. For example, to set it to channel 11, use this command:

```
sudo iw dev %INTERFACE% set channel 11
```

3.1.5. Intercepting 802.11 frames

Once the network interface is set to monitor mode, we can intercept 802.11 frames. Using tools such as Wireshark, we can capture and analyze these frames.

Management frames are typically transmitted at default data rates, which are compatible with most network adapters. Therefore, they can generally be captured by any network card in monitor mode. However, capturing data frames depends on the compatibility between your network card and the 802.11 standard used by the access point and the communicating device. If your network card supports an older 802.11 standard than the AP and device (for example, 802.11g instead of 802.11n), it may not capture the data frames, as traffic is destined to be transmitted at the highest modulation possible.

For open networks, captured communications are unencrypted at the 802.11 layer, allowing immediate analysis.

In WPA2-encrypted networks, you can decrypt captured communications if you know the network's passphrase. To achieve this, Wireshark can be configured to decrypt traffic, but you must capture the initial connection process (also known as the four-way handshake) between the client and AP to obtain the necessary encryption keys. This can be done by sending a disassociation frame, which force the client to reconnect, allowing you to capture the four-way handshake for decryption.

In WPA3 networks, intercepting and decrypting traffic is considerably more challenging than with WPA2 due to WPA3's enhanced security protocols. WPA3 uses Simultaneous Authentication of Equals (SAE) instead of the traditional four-way handshake, making it resistant to offline attacks and preventing decryption through simple capture methods. Additionally, WPA3 provides forward secrecy, meaning each session uses a unique encryption key. Consequently, even if an attacker gains the passphrase, previously captured traffic cannot be decrypted.

For open networks, WPA3 employs Opportunistic Wireless Encryption (OWE), which encrypts data without a password, unlike the cleartext transmission in WPA2 open

networks. This feature prevents casual interception, enhancing privacy on public networks. But since the AP and the client are not authenticated a Man-In-The-Middle attack could occur.

3.2. Creating rogue access points

In this section, we will create rogue access points to spoof the identity of the legitimate access point we set up previously. Using different software, we will verify if snappy can detect any differences in the beacon frame of the rogue AP.

Below is the output generated by snappy when analyzing the beacon frame of the legitimate access point.

Listing 3.2. Result of snappy on the legitimate AP

```
kali@kali:~$ sudo python3 snap.py legitimate.pcap
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: dd38a84a8d68dd1d2952ace861f9f7491f6b7f0ad38bf1cd72d742070322dd46
```

3.2.1. Using the same hardware

The first test involves determining whether it is possible to replicate the legitimate network exactly using the same hardware. To do this, we use a second TP-Link EAP650 and proceed identically to the configuration described in Section 3.1.1.

Once a beacon frame has been intercepted, we can analyze it using Snappy:

Listing 3.3. Result of Snappy on the Same Hardware

```
kali@kali:~$ sudo python3 snap.py ../pcap/hardware.pcap
BSSID: 28:87:ba:c0:42:fe
SSID: test_snappy
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: f7d5deaac59f46c82f274d3f47505b84a60c9fd70305251bfbbcbf27758ccfd9
```

As we can see, only the BSSID differs from the legitimate AP. In order to match the hash of the legitimate AP, we need to find a way to change the BSSID. We explored the following methods:

- **Web interface**: Navigating the default web interface (http://tplinkeap.net/) did not reveal any option to modify the BSSID.
- **Command-line interface (SSH)**: We enabled an SSH connection to the AP through the web interface. However, upon reviewing the available commands, we found that modifying the BSSID was not possible due to the limited privileges we had.
- **Vulnerabilities**: We searched for known vulnerabilities that might allow root access or modification of the MAC address but found none.
- **Backup/Restore system**: We attempted to modify the backup file to adjust the BSSID. However, the file is encrypted, authenticated, and compressed, making modification impossible. Furthermore, replacing the rogue AP's backup with that of the legitimate AP did not change the BSSID, suggesting it is not stored in the backup file.

3.2.2. Airbase-ng

Airbase-ng, part of the Aircrack-ng suite, enables the creation of a rogue AP. Before using it, we must set our network interface to monitor mode. We can verify the interface mode with *iwconfig*:

```
kali@kali:~$ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     unassociated  ESSID:""  Nickname:"<WIFI@REALTEK>"
          Mode:Managed  Frequency=2.412 GHz  Access Point: Not-Associated
          Sensitivity:0/0
          Retry:off   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality:0  Signal level:0  Noise level:0
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

As we can see, *wlan0* is in managed mode. We can change it to monitor mode using:

```
kali@kali:$ sudo airmon-ng check kill
kali@kali:$ sudo ip link set wlan0 down
kali@kali:$ sudo iw dev wlan0 set type monitor
kali@kali:$ sudo ip link set wlan0 up
```

Once in monitor mode, we can create an AP using airbase-ng. Using information from snappy about the legitimate AP's SSID, channel, and BSSID, we run:

```
kali@kali:~$ sudo airbase-ng -e test_snappy wlan0 -c 11 -a 28:87:BA:C0:43:38
08:37:35  Created tap interface at0
08:37:35  Trying to set MTU on at0 to 1500
08:37:35  Access Point with BSSID 28:87:BA:C0:43:38 started.
```

After capturing a beacon frame sent by the AP, we analyze it with snappy:

Listing 3.4. Result of Snappy on airbase-ng

```
kali@kali:~$ sudo python3 snap.py airbase-ng.pcap
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: 0
Supported Rates: [2, 4, 11, 22]
Extended Rates: b'\x02\x04\x0b\x16'
Max Transmit Power: 0
Capabilities: short-slot+ESS
Max_A_MSDU: 0
Vendor: 0
SHA256: 36d359740808f87bbaa078652bdfdc0adfd996c025ec97480408c72e38d14823
******** AIRBASE-NG DETECTED AT THIS ACCESS POINT ********
```

By comparing the hash from the legitimate AP and the AP generated by airbase-ng, we see differences due to airbase-ng's limitations in setting specific values for country, supported rates, extended rates, maximum transmit power, capabilities, Max_A_MSDU, and vendor. Additionally, snappy detected airbase-ng from these defaults values. Modifying any of these values in the source code can potentially evade the airbase-ng's detection.

Currently, clients connecting to this AP will not have internet access. To provide connectivity, additional steps are required[6]. When we ran airbase-ng, a new tap interface, "at0", was created. We need to bring up the "at0" interface and assign it an IP address:

```
kali@kali:~$ sudo ifconfig at0 up
kali@kali:~$ sudo ifconfig at0 10.0.0.1 netmask 255.255.255.0
```

We then add the necessary IP forwarding and iptables rules:

```
kali@kali:~$ sudo sysctl -w net.ipv4.ip_forward=1
kali@kali:~$ sudo route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.0.1
kali@kali:~$
kali@kali:~$ sudo iptables -F sudo iptables -t nat -F
kali@kali:~$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
kali@kali:~$
kali@kali:~$ sudo iptables -A FORWARD -i at0 -o eth0 -j ACCEPT
kali@kali:~$ sudo iptables -A FORWARD -i eth0 -o at0 -m state --state
   RELATED,ESTABLISHED -j ACCEPT
```

Finally, we need to give an IP address to the clients that connect to our AP. To do that, we run a dnsmasq server. It is a lightweight software providing DNS, DHCP, etc. You can install it using:

```
sudo apt-get install dnsmasq
```

We then create a configuration file named *dnsmasq.conf*:

Listing 3.5. dnsmasq.conf

```
interface=at0
dhcp-range=10.0.0.100,10.0.0.250,255.255.255.0,12h
dhcp-option=3,10.0.0.1
dhcp-option=6,10.0.0.1
server=8.8.8.8
log-queries
log-dhcp
listen-address=127.0.0.1
```

Start the dnsmasq server using:

```
sudo dnsmasq -C dnsmasq.conf -d
```

At this point, clients should have internet access.

Future research could focus on trying to modify airbase-ng's source code to allow custom values for country, supported rates, extended rates, maximum transmit power, capabilities, Max_A_MSDU, and vendor.

3.2.3. Airgeddon

Another tool that can be used to create a rogue access point is Airgeddon (https://github.com/v1s1t0r1sh3r3/airgeddon). Airgeddon is a multi-purpose security tool that supports creating rogue access points, among other Wi-Fi auditing features. This tool is not included by default in Kali Linux, but you can install it with the following commands:

```
git clone https://github.com/v1s1t0r1sh3r3/airgeddon.git
cd airgeddon
sudo bash airgeddon.sh
```

Once installed, Airgeddon allows you to easily set up a rogue access point for testing purposes. We chose the "Evil Twin attack" option from the menu and captured a beacon frame from the AP we just generated:

Listing 3.6. Result of Snappy on Airgeddon

```
kali@kali:~$ sudo python3 snap.py ../pcap/airgeddon.pcap
[sudo] password for kali:
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: 0
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 0
Capabilities: short-slot+ESS
Max_A_MSDU: 0
Vendor: 0
SHA256: afd02664587c0f26f85689329dad034f77c59bd239592165b14c63dc84b45496
```

In this output, several attributes match the desired configuration. However, some values, such as Country, Max Transmit Power, Capabilities, Max_A_MSDU, and Vendor, may not fully align with the legitimate AP's settings due to Airgeddon's limitations in setting specific values.

3.2.4. Lnxrouter

Lnxrouter (https://github.com/garywill/linux-router) enables the creation of an AP. Although it isn't specifically designed for creating rogue access points, it can be used for that purpose. Unlike other tools that require multiple steps to set up an AP, lnxrouter simplifies the process by handling the configuration in one command, making it easy to use.

To use lnxrouter, first clone the repository and navigate into the directory with the following commands:

```
git clone https://github.com/garywill/linux-router.git
cd linux-router
```

Next, configure lnxrouter to match the legitimate router's settings as closely as possible. The command below sets the SSID, channel, BSSID, and country code to

align with the characteristics of the target AP:

```
sudo ./lnxrouter --ap wlan0 test_snappy -c 11 --mac "28:87:BA:C0:43:38" --
    country DE --wifi4 --ht-capab [MAX-AMSDU-7935]
```

After running the command, lnxrouter configures the AP with the specified settings. The resulting beacon frames can be captured and analyzed with snappy to confirm their similarity to the legitimate AP, as shown below:

Listing 3.7. Result of snappy on the lnxrouter

```
kali@kali:~$ sudo python3 snap.py ../pcap/lnxrouter.pcap
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS
Max_A_MSDU: 1
Vendor: 0
SHA256: 050ecfb901318fc4c0541c42f888307a4f03de3dd41e5d655bb849b5e29688cd
```

As seen from the result, only the capabilities and vendor fields differ from the legitimate AP. This outcome illustrates that while lnxrouter can closely mimic the target AP, it cannot fully replicate all fields.

3.2.5. Hostapd

Hostapd is a powerful utility for creating software-based access points on Linux systems. While it is primarily intended for legitimate access points, it can also be configured to imitate a specific access point, which may be useful for testing and research purposes. Unlike lnxrouter, hostapd provides a higher level of customization over various AP settings, allowing for closer emulation of a legitimate access point's characteristics.

To get started with hostapd, you will need to install it (if it isn't already) and create a configuration file specifying the desired AP parameters. Begin by installing hostapd and then navigating to a suitable directory for creating the configuration file:

```
sudo apt install hostapd
```

Next, create a configuration file (e.g., 'hostapd.conf') to define the SSID, BSSID, channel, and other details to match the legitimate AP. Below is an example configuration that specifies the SSID, channel, country code, and MAC address:

Listing 3.8. hostapd.conf

```
interface=wlan0
driver=nl80211
ssid=test_snappy
channel=11
hw_mode=g
bssid=28:87:ba:c0:43:38
supported_rates=10 20 55 110 60 90 120 180
preamble=1
country_code=DE
ieee80211d=1
ieee80211n=1
ht_capab=[MAX-AMSDU-7935]
vendor_elements=dd180050f2020101800003a4000027a4000042435e0062322f00
```

- **interface**: The network interface to be used for the access point (e.g., wlan0).
- **driver**: The driver used for the wireless interface, typically set to nl80211 for modern Linux systems.
- **ssid**: The name of the Wi-Fi network (SSID) that will be broadcast.
- **channel**: The radio channel on which the access point will operate.
- **hw_mode**: The operating mode of the hardware (g for 2.4 GHz).
- **bssid**: The MAC address of the access point, which can be set to mimic a legitimate AP.
- **supported_rates**: List of rates that can be used. (Needs to be supported by the hardware)
- **preamble**: Determines the preamble type; 1 enables short preambles.
- **country_code**: Defines the regulatory domain for the access point, affecting allowed channels and power levels.
- **ieee80211d**: Enables 802.11d for country code compliance.

- **ieee80211n**: Enables 802.11n support for improved performance.
- **ht_capab**: Used to set the maximum A-MSDU length.
- **vendor_elements**: Used to specify additional vendor-specific elements for Beacon frames.

After creating the configuration file, you can start the access point using the following command:

```
sudo hostapd /etc/hostapd/hostapd.conf
```

We can now run snappy on a beacon frame of the AP:

Listing 3.9. Result of snappy on hostapd

```
kali@kali:~$ sudo python3 snap.py ../pcap/hostapd.pcap
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: dd38a84a8d68dd1d2952ace861f9f7491f6b7f0ad38bf1cd72d742070322dd46
```

As we can see, all the values match those of the legitimate AP, and therefore the hash is also identical.

Like airbase-ng, if a client connects to the AP now, it won't have internet access. To enable a connection, we need to do a few additional steps. First, we need to create a configuration file for *dnsmasq*:

Listing 3.10. dnsmasq.conf

```
interface=wlan0
dhcp-range=10.0.0.100,10.0.0.250,255.255.255.0,12h
dhcp-option=3,10.0.0.1
dhcp-option=6,10.0.0.1
server=8.8.8.8
log-queries
log-dhcp
listen-address=127.0.0.1
```

Once done, to start dnsmasq we can use the following command:

```
sudo dnsmasq -C dnsmasq.conf -d
```

We also need to enable IP forwarding, assign an IP address to the wireless interface we are using, and create NAT on our interface connected to the internet.

```
sudo ifconfig wlan0 10.0.0.1/24
sudo sysctl -w net.ipv4.ip_forward=1

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Since we have finally managed to create an AP that bypasses Snappy's detection, we can now attempt to create an access point using WPA2 passphrase authentication. In many public spaces, such as fast-food restaurants, there is no longer an open network but rather a WPA2 network that uses a publicly known passphrase (e.g., the passphrase may be displayed on the wall). We will test whether it is possible to spoof the identity of these types of APs when we know the passphrase.

To do this, we first need to create a legitimate WPA2 network. As described in Section 3.1.1, we set up another network using WPA2 with the name "test_snappy_wpa2." Below is the result of a beacon frame from that new AP, where we can see that "privacy" has been added in the capabilities information:

Listing 3.11. Result of snappy on the legitimate WPA2 AP

```
kali@kali:~$ sudo python3 snap.py ../pcap/legitimate_wpa2.pcap
BSSID: 2e:87:ba:c0:43:38
SSID: test_snappy_wpa2
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+privacy+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: ef698256278f054ac7984e5f6da4671b82b700ff82f8cfc9d4c29c745949a4d5
```

Now we will try to spoof the identity of this new access point. To do that, we only need to add a few lines to the hostapd configuration file (i.e., the last ones):

Listing 3.12. hostapd.conf

```
interface=wlan0
driver=nl80211
ssid=test_snappy_wpa2
channel=11
hw_mode=g
bssid=2e:87:ba:c0:43:38
supported_rates=10 20 55 110 60 90 120 180
preamble=1
country_code=DE
ieee80211d=1
ieee80211n=1
ht_capab=[MAX-AMSDU-7935]
vendor_elements=dd180050f2020101800003a4000027a4000042435e0062322f00
wpa=1
wpa_passphrase=test_snappy_wpa2
wpa_key_mgmt=WPA-PSK
wpa_pairwise=CCMP
rsn_pairwise=CCMP
```

After this, we can capture a beacon frame and analyze it:

Listing 3.13. Result of snappy on the hostapd WPA2 AP

```
kali@kali:~$ sudo python3 snap.py ../pcap/hostapd_wpa2.pcap
BSSID: 2e:87:ba:c0:43:38
SSID: test_snappy_wpa2
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+privacy+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: ef698256278f054ac7984e5f6da4671b82b700ff82f8cfc9d4c29c745949a4d5
```

We can see that the hashes match as well, which means that Snappy could not detect the rogue AP.

We created a small script called **hostapdCreation.py** that takes a pcap file containing beacon frames as a parameter. This script then generates a hostapd configuration file and starts it, effectively bypassing Snappy's detection:

https://github.com/Fufuches/RogueAP/blob/main/hostapdCreation.py

3.2.6. Scapy

We saw previously that we can use hostapd to create an AP that, according to Snappy, looks like the legitimate AP. However, when we use Wireshark to capture the beacon frames of both the legitimate AP and the hostapd AP, we notice some differences:

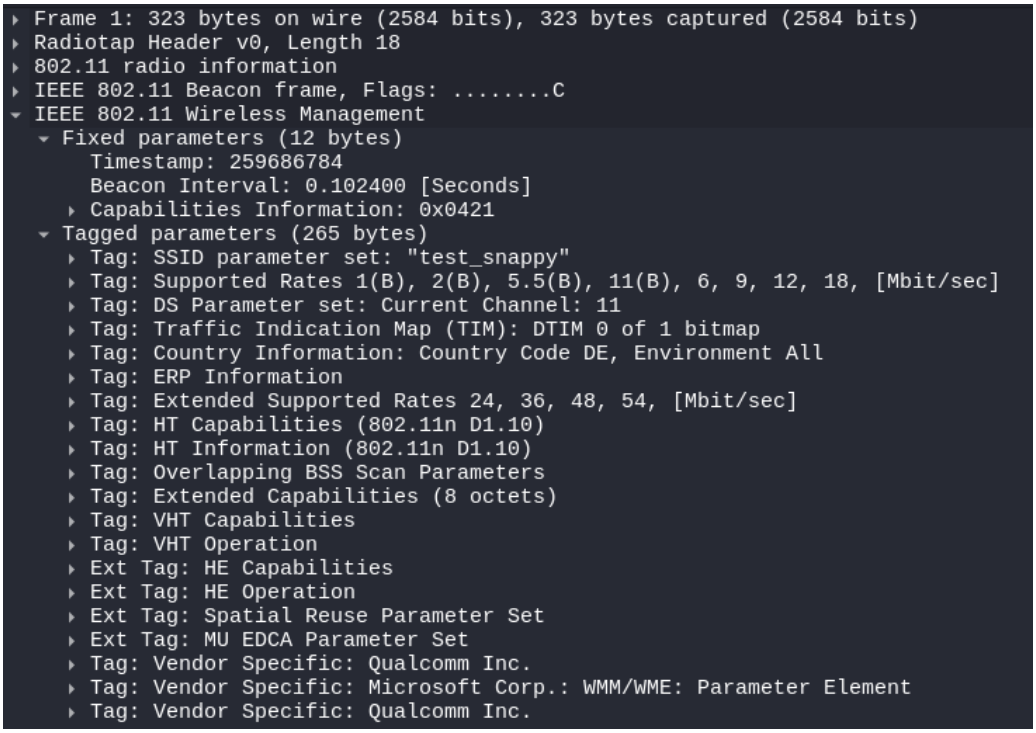

Figure 3.2. Beacon frame of legitimate AP in Wireshark

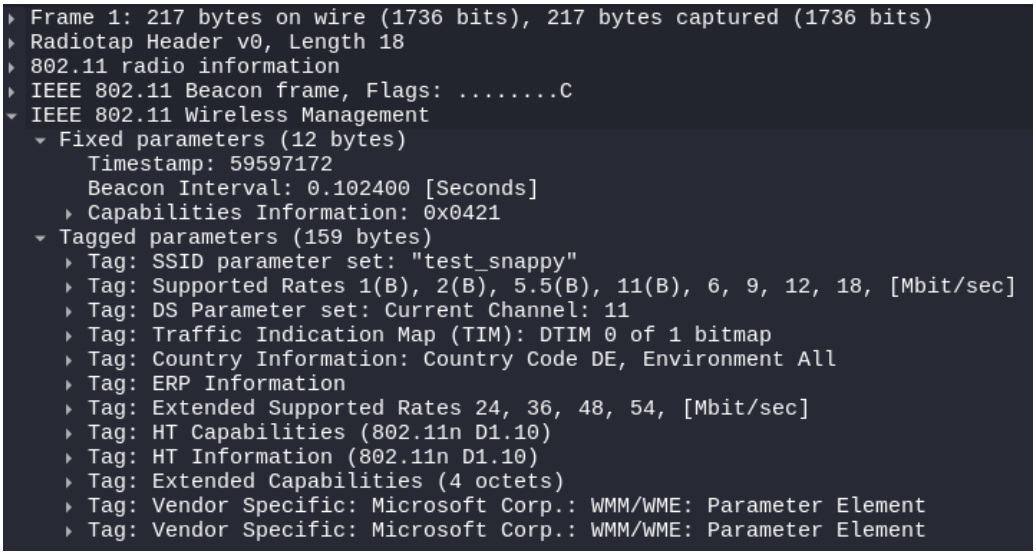

Figure 3.3. Beacon frame of hostapd AP in Wireshark

We came up with the idea of creating an AP using Scapy (https://scapy.net/), a Python package for network manipulation, because it enables us to create frames from scratch. This would give us more flexibility than using an existing program and might allow us to generate a beacon frame that matches the legitimate AP exactly.

By using a wireless adapter in monitor mode, the Python program with Scapy is able to listen to all 802.11 communications and send packets on a specific chan-

nel. A thread is used to send beacon frames every 0.01 seconds. At the same time, the program listens for probe requests, authentication requests, and association requests, responding to them as needed. Once a client connects, the program creates a TAP interface to manage data frames, similar to how airbase-ng works. A TAP device is a virtual network interface that resembles an Ethernet network card. When a data packet arrives, its 802.11 envelope is removed, and the frame is wrapped in an Ethernet frame and forwarded to the TAP interface. On this interface, a dnsmasq server runs, offering DHCP and DNS services.

You can find the code and the readme on this github:

https://github.com/Fufuches/RogueAP/tree/main

Clients can connect to the AP we just created and obtain an IP address, but the internet connection will be very slow, making it unusable for practical purposes.

We tried running the code directly on the computer's OS and on a Raspberry Pi instead of a virtual machine, but the speed issue persisted.

Nevertheless, we still captured a beacon frame from the AP, ran Snappy on it, and, as you can see, it successfully bypassed Snappy's detection, obtaining the same hash as the legitimate AP:

Listing 3.14. Result of snappy on scapy script

```
kali@kali:~$ sudo python3 snap.py ../pcap/scapy.pcap
BSSID: 28:87:ba:c0:43:38
SSID: test_snappy
Channel: 11
Country: DE
Supported Rates: [130, 132, 139, 150, 12, 18, 24, 36]
Extended Rates: b'\x82\x84\x8b\x96\x0c\x12\x18$'
Max Transmit Power: 20
Capabilities: short-slot+ESS+short-preamble
Max_A_MSDU: 1
Vendor: 20722
SHA256: dd38a84a8d68dd1d2952ace861f9f7491f6b7f0ad38bf1cd72d742070322dd46
```

In theory, the creation of an AP using Scapy should bypass Snappy's verification, but in practice, we were not able to make it work.

3.3. Results

The table below provides a summary of the lab tests. In the first row, each column represents a different method or software used to create an AP. In the first column, each row shows the values that Snappy uses to generate its hash. The final row indicates whether the hash matches the legitimate AP's hash: green check mark indicates a match, red X mark signifies no match, and blue check mark means it should theoretically work, though practical implementation was not successful.

	Same Hardware	Airbase-ng	Airgeddon	Inxrouter	hostapd	Scapy
BSSID	✗	✓	✓	✓	✓	✓
SSID	✓	✓	✓	✓	✓	✓
Channel	✓	✓	✓	✓	✓	✓
Country	✓	✗	✗	✓	✓	✓
Rates	✓	✗	✓	✓	✓	✓
Extended Rates	✓	✗	✓	✓	✓	✓
Max Transmit Power	✓	✗	✗	✓	✓	✓
Capabilities	✓	✗	✗	✗	✓	✓
Max_ A_MSDU	✓	✗	✗	✓	✓	✓
Vendor	✓	✗	✗	✗	✓	✓
Hash	✗	✗	✗	✗	✓	✓

As shown in the table, a correctly configured hostapd can bypass Snappy's verification by generating the same hash as the legitimate AP. This highlights some inherent limitations of Snappy in detecting rogue APs.

Several limitations of utilizing Snappy were observed:

- **Networks with Multiple APs:** Many legitimate networks deploy multiple APs to provide wider coverage under the same ESSID. These APs often have unique BSSIDs and may operate on different channels. Snappy would treat each AP as a distinct entity, potentially flagging them as changes or rogue APs.

- **Dynamic Channel Switching:** Some access points dynamically adjust their operating channel to reduce interference or comply with regulatory requirements. These legitimate changes might be incorrectly flagged by Snappy as rogue behavior, leading to false positives.
- **Attacks Mimicking Hash Attributes:** While Snappy is effective against many rogue APs, as demonstrated, an attacker using tools like hostapd can mimic the attributes of a legitimate AP to produce identical hashes. This makes detection challenging when an attacker carefully replicates the key attributes Snappy uses for verification.

4. Other findings

In this chapter, we include smaller findings that were discovered during the internship.

4.1. Information about the model of the access point

It is sometimes possible to determine the manufacturer of an Access Point (AP) using the OUI (Organizationally Unique Identifier) database. The OUI is the first 24 bits of a MAC address and is assigned to specific manufacturers. For example, using a website like https://macvendors.com/, we can look up the MAC address of the legitimate AP "28:87:ba:c0:43:38" and determine that the manufacturer is *TP-LINK*.

However, identifying the exact model of the AP is not always straightforward. This is generally possible only if the AP uses Wi-Fi Protected Setup (WPS). When WPS is enabled, it allows for querying the specific model of the device. However, WPS has been deprecated by the Wi-Fi Alliance, and as a result, fewer APs support it today.

For example, in the figure below, we can observe a probe response from an AP where the model, a *BBOX3* from Sagemcom, is visible. (https://miloserdov.org/?p=306)

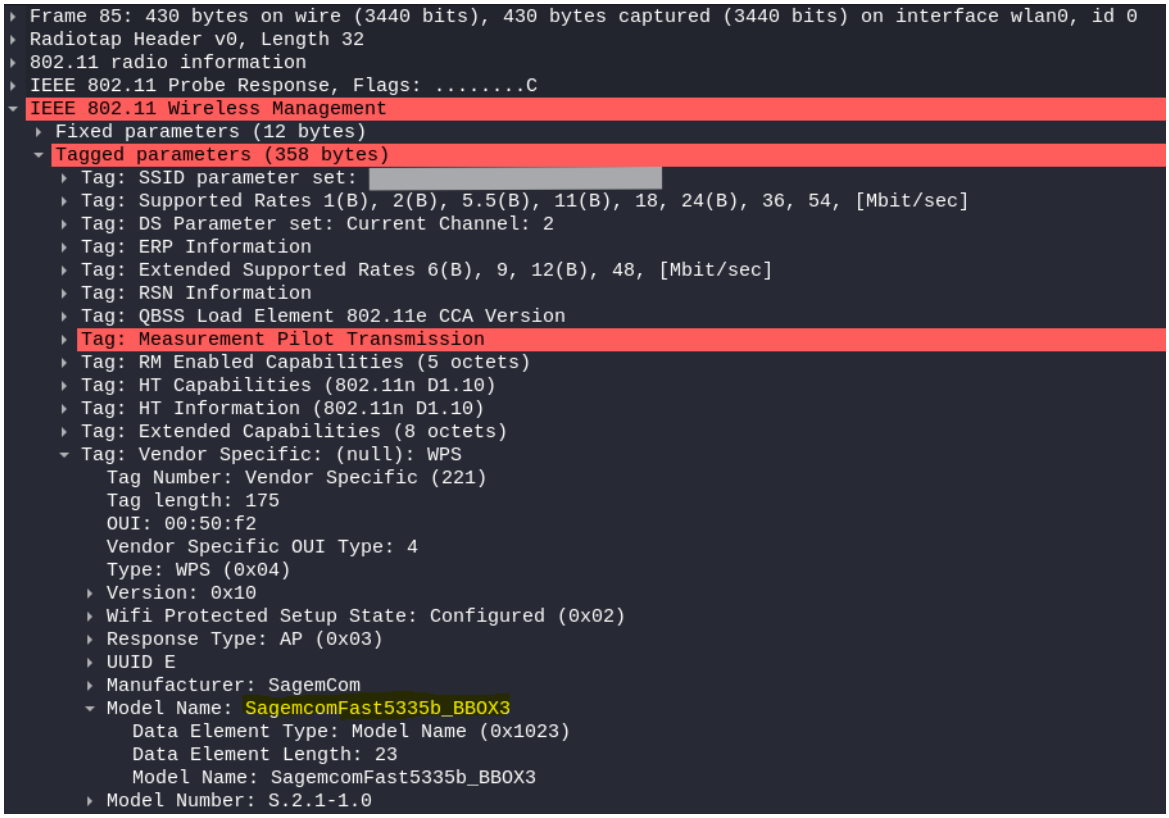

Figure 4.1. Probe response showing AP model: BBOX3 from Sagemcom.

4.2. Identifying the channel on which a WiFi network operates

As previously discussed, the 802.11 standard operates on different channels. To determine the channel on which a Wi-Fi network is operating, two techniques can be used:

1. **Wireshark**: Using Wireshark with an interface in monitor mode, we can capture 802.11 frames, including beacon frames from an AP. To switch between channels manually, you can use the following command, where "XX" is the desired channel:

```
sudo iw dev %INTERFACE% set channel XX
```

   By doing this, you will eventually capture the beacon frame from the AP you are investigating. In the example below, after switching channels, we captured the beacon frame from "test_snappy" and determined that it operates on channel 11:

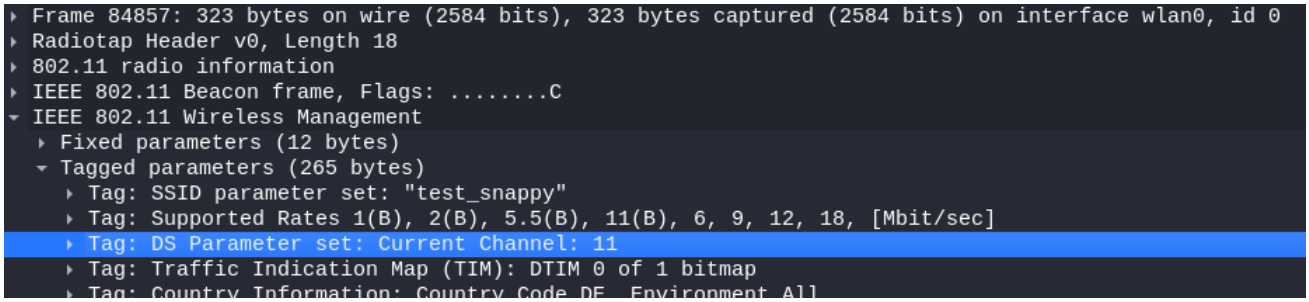

Figure 4.2. Capture of a beacon frame from "test_snappy".

2. **Airodump-ng**: This tool from the Aircrack-ng suite allows us to automatically scan all available channels and list nearby access points. Using it allow us to determine by referring to the "CH" column that the channel :

Listing 4.1. Result of airodump-ng.

```
kali@kali~$ sudo airodump-ng wlan0
[CH  7 ][ Elapsed: 6 s ][ 2024-10-24 09:46]
BSSID              PWR Beacons #Data, #/s CH  MB ENC CIPHER AUTH ESSID
28:87:BA:C0:43:38 -39       16      0    0 11 360 OPN              test_snappy
```

5. Conclusion

In conclusion, our tests demonstrated that Snappy successfully detected most rogue access points attempting to impersonate a legitimate AP. However, the ability to generate a rogue AP using Hostapd that produces an identical hash to the legitimate AP calls into question the reliability of Snappy as a tool to consistently identify rogue access points. This finding underscores the need for caution when connecting to open Wi-Fi networks, as the security of such networks can be compromised.

Bibliography

[1] J. Pacheco. "[802.11] wi-fi connection/disconnection process." (2020), [Online]. Available: https://community.nxp.com/t5/Wireless-Connectivity-Knowledge/802-11-Wi-Fi-Connection-Disconnection-process/ta-p/1121148 (visited on 09/09/2024).

[2] Jeremymsharp. "802.11 frame types and formats." (2020), [Online]. Available: https://howiwifi.com/2020/07/13/802-11-frame-types-and-formats/ (visited on 11/15/2024).

[3] R. Nayanajith. "802.11 mgmt : Beacon frame." (2014), [Online]. Available: https://mrncciew.com/2014/10/08/802-11-mgmt-beacon-frame/ (visited on 11/15/2024).

[4] SpiderLabs. "Snappy." (2023), [Online]. Available: https://github.com/SpiderLabs/snappy/blob/main/snap.py (visited on 09/02/2024).

[5] "Realtek rtl8188eus rtl8188eu rtl8188etv wifi drivers." (), [Online]. Available: https://github.com/aircrack-ng/rtl8188eus (visited on 11/16/2024).

[6] cyb3r_dan. "Creating an evil twin or fake access point on your home network using aircrack-ng and dnsmasq." (2018), [Online]. Available: https://thecybersecurityman.com/2018/08/11/creating-an-evil-twin-or-fake-access-point-using-aircrack-ng-and-dnsmasq-part-2-the-attack/ (visited on 09/16/2024).