



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Characterizing MAC randomiza- tion

Wireless Internet Project

Author: **Gianluca Vigo [XXXXXXXX]**

Academic Year: 2023-2024

Contents

| | |
|---|-----------|
| Contents | i |
| 1 Abstract | 1 |
| 2 Introduction | 3 |
| 2.1 Active scanning and Probe requests | 3 |
| 2.1.1 Passive mode | 4 |
| 2.1.2 Active mode | 4 |
| 2.2 Privacy dilemma and MAC randomization | 5 |
| 3 Project implementation | 7 |
| 3.1 Set-up | 7 |
| 3.1.1 Operating System | 7 |
| 3.1.2 Monitor mode | 8 |
| 3.2 Wireshark | 13 |
| 3.3 Colab Notebook | 15 |
| 3.3.1 Import & Installation | 15 |
| 3.3.2 Capture file selection and reading | 15 |
| 3.3.3 Frame Validity check | 15 |
| 3.3.4 Probes filtering according to RSSI | 16 |
| 3.3.5 Vendors representation | 16 |
| 3.3.6 MAC Randomization Check | 17 |
| 3.3.7 Final metrics: Average RSSI, Burst size and Average IAT | 18 |
| 4 Conclusions | 19 |
| 4.1 Android | 19 |
| 4.1.1 Samsung A53 | 21 |
| 4.1.2 Huawei Mediapad T5 | 21 |
| 4.2 Ios-iPad | 21 |

| | |
|------------------------|-----------|
| Bibliography | 23 |
| List of Figures | 25 |

1 | Abstract

The main goal of this project is producing a set of .pcap files, containing probe requests emitted by one or more devices **under different conditions** as screen on/off, Wi-Fi on/off, Power saving on/off in order to characterize the behaviour of probe request generation and MAC randomization.

Together with this final report and the set of capture files, a python Colab page designed for filtering and analyse these files is added.

2 | Introduction

In this section, I'm presenting some introductory concepts that are useful for a better understanding of the project.

2.1. Active scanning and Probe requests

In 802.11 standards, the connection procedure includes three major steps that shall be performed to make the device part of the Wi-Fi network and communicate in the network. Those three steps are device discovery (scanning), device authentication (checking compatibility-capability etc. before connection) and then finally establishment of connection (Association).

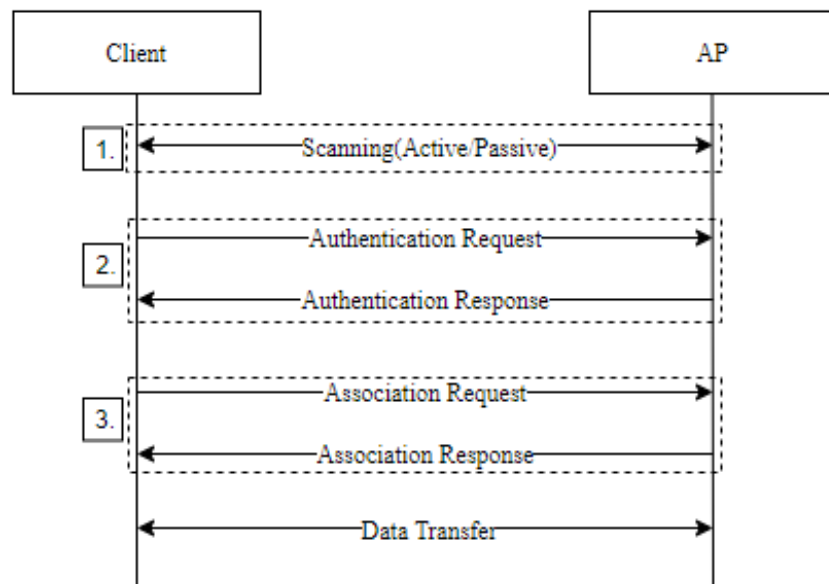


Figure 2.1: Connection process

The project is more focused on the very first step: Scanning. [1]

In order to join any network, first client or station needs to find it. In the wired networks, just plugging the cable or jack will be enough. In the wireless world, this requires iden-

tification of the compatible network before joining process can begin. This identification process of the network is referred as scanning and its duty is to discover all the available Basic Service Sets (BSS).

The scanning procedure is performed by the stations (STA).

There are two possible options:

- Passive mode
- Active mode

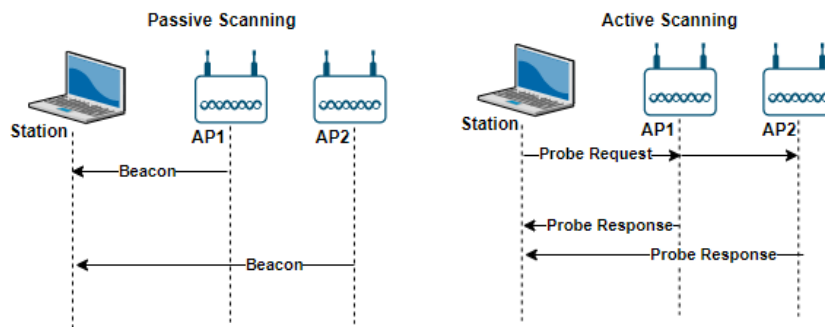


Figure 2.2: Scanning methods

2.1.1. Passive mode

In Passive Scanning, STA listens to the available channels in sequence and it stores all beacon frames received. Beacon frames are used by the access points (AP) to communicate or to announce themselves.

2.1.2. Active mode

In Active Scanning, the STA uses Probe Request frames for soliciting the transmission of beacons. If any AP is presenting that frequency, they should answer with the probe response frame.

Active mode is usually preferred cause it's more efficient, but it also carries risks connected to the proactive behaviour of STAs and the multitude of non-encrypted information incorporated in probe requests.

2.2. Privacy dilemma and MAC randomization

802.11 probe requests contain the MAC address of the sending device and, without any countermeasure, these leak relevant sensible information: listeners (or traffic sniffers) are able to build an history of devices activity and, in this way, they are able to "track" them.

The most adopted solution is MAC randomization, invented by Apple Inc. and then also extended to other electronic devices and software producers.

MAC randomization is a process where software changes the hardware MAC address to random and unpredictable strings.

It must be clear that not all MAC randomization protocols are equal: they depends on the device manufacturer or even on the device model.

3 | Project implementation

3.1. Set-up

Many devices of different vendors with different characteristics were considered in the project in order to examine their MAC randomization protocols.

| Device ID | Device OS | Device OS Version | Device Vendor | Device Model | Random MAC |
|-----------|-----------|-------------------|---------------|-------------------------------|------------|
| A | Android | 14 | Samsung | Galaxy A53-5G, SM-A536B/DS | Yes |
| B | Ios | 17.4.1 | Apple | iPad 9a generation, MK2K3TY/A | Yes |
| C | Android | 8 | Huawei | Huawei MediaPad T5, AGS2-W09 | Yes |

I also needed additional equipment:

- Lenovo Ideapad 310-15ABR (*laptop*) [Kali GNU]
- Alfa Netowrk AWUS036NHA (*Long-Range USB Adapter*) [Qualcomm Atheros AR9271]

The first one was used to capture the frames and I bought the other one in order to have a more reliable wireless interface to work with, instead of using the Lenovo laptop built-in one.

3.1.1. Operating System

Since the Lenovo laptop was running Windows and this particular OS is very limited concerning the activation of the "monitor mode" which is absolutely necessary to implement the project, I decided to install Kali Linux on it.

In particular the specific distribution I installed is:

Linux LAPTOP-2THDN3J6 6.8.11-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.8.11-1kali2 (2024-05-30) x86_64 GNU/Linux

3.1.2. Monitor mode

Monitor mode allows a computer with a Wireless Network Card Interface (WNIC) to monitor all traffic received on a wireless channel. Unlike promiscuous mode, monitor mode allows packets to be captured without having to associate with an AP.

Through this mode, I was able to capture some messages sent in-the-clear as Beacons, Probe Requests, Probe Responses, etc. coming from APs and STAs.

How to enable/disable Monitor Mode

In order to work with Monitor Mode in Kali Linux [6], I used the *airmon-ng* [2] packet and this script can be used to enable monitor mode on wireless interfaces. It may also be used to kill network managers, or go back from monitor mode to managed mode.

Here's the list of commands I used:

- **ip addr**

To see ip address interfaces

```

(gianluca@LAPTOP-2THDN3J6)-[~]
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
   link/ether c8:5b:76:68:2f:97 brd ff:ff:ff:ff:ff:ff
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
   link/ether 78:45:63:c9:d7:b9 brd ff:ff:ff:ff:ff:ff
   inet 192.168.0.123/24 brd 192.168.0.255 scope global dynamic noprefixroute wlan0
       valid_lft 603780sec preferred_lft 603780sec
   inet6 fe80::b207:2536:7bcc:78de/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: wlan1: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default qlen 1000
   link/ether 82:3b:f4:57:d9:64 brd ff:ff:ff:ff:ff:ff permaddr 00:c0:ca:b4:f8:8c
(gianluca@LAPTOP-2THDN3J6)-[~]
$

```

Figure 3.1: ip addr result

In my case I have two wlan interfaces: **wlan0** is the built-in one and it's already associated to my Wi-Fi home network (Casa Vigo) so that's why it's **UP**, instead **wlan1** is the interface offered by the Alfa Network adapter and it will be this one on which I'm relying on for this project. At the moment it's not associated to any AP and this explains the **DOWN** state. We can also see that both interfaces are in **Managed** mode.

- **iwconfig**

An alternative to the *ip addr* command

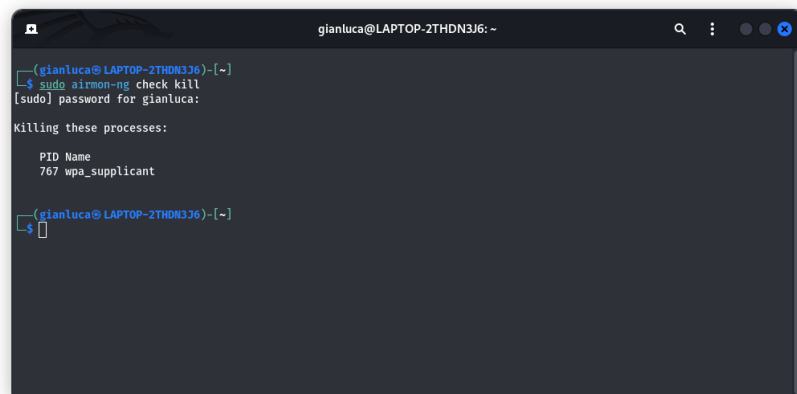


```
gianluca@LAPTOP-2THDN3J6: ~  
$ iwconfig  
lo        no wireless extensions.  
  
eth0      no wireless extensions.  
  
wlan0     IEEE 802.11  ESSID:"Casa Vigo"  
          Mode:Managed  Frequency:5.18 GHz  Access Point: 78:98:E8:D7:BB:C8  
          Bit Rate=43.3 Mb/s   Tx-Power=30 dBm  
          Retry short limit:7   RTS thr=2347 B   Fragment thr:off  
          Power Management:on  
          Link Quality=56/70  Signal level=-54 dBm  
          Rx invalid mwid:0  Rx invalid crypt:0  Rx invalid frag:0  
          Tx excessive retries:0  Invalid misc:3293  Missed beacon:0  
  
wlan1     IEEE 802.11  ESSID:off/any  
          Mode:Managed  Access Point: Not-Associated  Tx-Power=30 dBm  
          Retry short limit:7   RTS thr:off   Fragment thr:off  
          Power Management:off  
  
$
```

Figure 3.2: iwconfig result

- **sudo airmon-ng check kill**

Once obtained root privileges through *sudo*, I'm going to use *airmon-ng check kill* to find any conflicting processes and so kill them.




```
gianluca@LAPTOP-2THDN3J6: ~  
$ sudo airmon-ng check kill  
[sudo] password for gianluca:  
  
Killing these processes:  
  
PID Name  
767 wpa_supplicant  
  
$
```

Figure 3.3: check kill result

- `sudo airmon-ng start wlan1`

To change the mode on the wlan1 interface: from *Managed* to *Monitor*.



```
(gianluca@LAPTOP-2THDN3J6)-[~]
$ sudo airmon-ng start wlan1

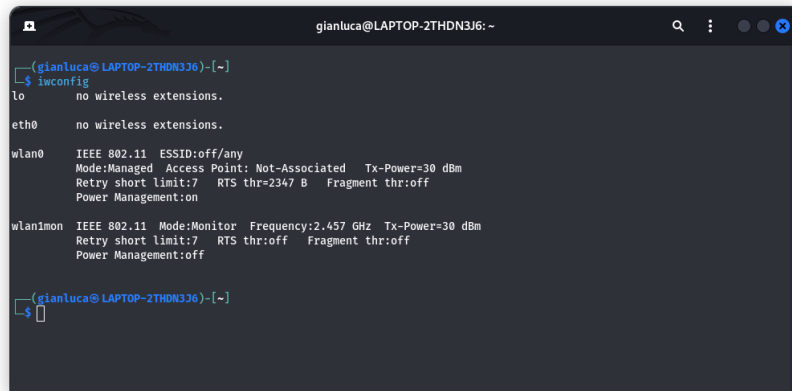
PHY   Interface  Driver      Chipset
phy0   wlan0       rtl8821ae   Realtek Semiconductor Co., Ltd. RTL8821AE 802.11ac PCIe Wireless Network Adapter
phy1   wlan1       ath9k_htc   Qualcomm Atheros Communications AR9271 802.11n
        (mac80211 monitor mode vif enabled for [phy1]wlan1 on [phy1]wlan1mon)
        (mac80211 station mode vif disabled for [phy1]wlan1)

(gianluca@LAPTOP-2THDN3J6)-[~]
$
```

Figure 3.4: start result

Now the monitor mode is enabled for *wlan1mon* interface: this will be the interface that I'll use in Wireshark.

By using again the `iwconfig` command it's more evident the change.



```
(gianluca@LAPTOP-2THDN3J6)-[~]
$ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated  Tx-Power=30 dBm
          Retry short limit:7   RTS thr=2347 B   Fragment thr:off
          Power Management:on

wlan1mon  IEEE 802.11  Mode:Monitor  Frequency:2.457 GHz  Tx-Power=30 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

(gianluca@LAPTOP-2THDN3J6)-[~]
$
```

Figure 3.5: checking start

- `sudo airodump-ng wlan1mon`

To see all the networks that are available together with stations.

```

gianluca@LAPTOP-2THDN3J6: -
Mode:Managed Access Point: Not-Associated Tx-Power=30 dBm
CH 4 [ Elapsed: 6 s ] [ 2024-08-28 16:25

BSSID PWR Beacons #Data, #s CH MB ENC CIPHER AUTH ESSID
9E:A2:F4:0C:EA:F0 -88 3 0 0 11 270 WPA2 COMP PSK <length: 0>
DC:F8:B9:A6:A7:07 -83 4 0 0 11 270 WPA2 COMP PSK FASTWEB-Casa
AC:1D:0F:01:76:20 -84 4 0 0 11 235 WPA2 COMP PSK <length: 0>
B0:02:9C:0A:08:04 -88 6 1 0 4 270 WPA2 COMP PSK Linken_0A8803
82:AA:F4:BB:2F:E1 -87 5 1 0 9 720 WPA2 COMP PSK PosteMobile-38504642
98:0E:00:F8:0A:10 -84 6 0 0 3 195 WPA2 COMP PSK TP-Link_CK0
A6:00:6A:03:32:AA -89 6 3 0 3 720 WPA2 COMP PSK TIM-01173104
7A:98:E8:D7:BB:C7 -58 5 0 0 1 130 WPA2 COMP PSK <length: 19>
04:59:F8:02:E7:F8 -84 17 1 0 8 130 WPA2 COMP PSK Vodafone-33472125
B4:0F:3B:88:96:A9 -84 5 0 0 6 130 WPA2 COMP PSK Tiscali casa pet
B0:0A:D5:51:40:72 -75 11 2 0 6 360 WPA3 COMP SAE VF_IT_FWA_4072
08:0D:17:0F:08:84 -77 22 0 0 2 270 WPA2 COMP PSK TP-Link_DB84
78:98:E8:D7:BB:C7 -57 8 0 0 1 130 WPA2 COMP PSK Casa Vigo
7A:98:E8:D7:BB:C7 -53 13 0 0 1 130 WPA2 COMP PSK <length: 19>
06:10:3B:20:0F:BE -85 5 0 0 1 130 WPA2 COMP PSK Infinity 5g
78:98:E8:D7:BB:C7 -60 13 4 1 1 130 WPA2 COMP PSK Casa Vigo
00:19:3B:20:0F:BE -85 9 0 0 1 130 WPA2 COMP PSK Infinity.pet
5E:B1:01:13:AB:18 -87 7 0 0 1 720 WPA2 COMP PSK TIM-Anna-Simo_2_4g
9C:A2:F4:7C:EA:F0 -86 5 0 0 11 270 WPA2 COMP PSK TP-Link_EAR6

BSSID STATION PWR Rate Lost Frames Notes Probes
(not associated) B8:4D:43:9C:B1:63 -79 0 - 1 2 5 Casa Vigo
B4:0F:3B:88:96:A9 04:E6:76:EE:3E:80 -1 1e- 0 0 5
78:98:E8:D7:BB:C7 8C:CE:AE:F0:8F:86 -77 0 - 6 0 1
78:98:E8:D7:BB:C7 A2:06:01:0F:21:C1 -83 0 - 1 0 4
78:98:E8:D7:BB:C7 20:FE:00:9B:24:7E -47 0 - 1e 0 5
78:98:E8:D7:BB:C7 E4:34:93:0A:C3:81 -34 0 - 6 0 1

```

Figure 3.6: airodump-ng

- `sudo airmon-ng stop wlan1mon`

To put the network interface back to *Managed* mode.

```

gianluca@LAPTOP-2THDN3J6: ~
(gianluca@LAPTOP-2THDN3J6)~$ sudo airmon-ng stop wlan1mon

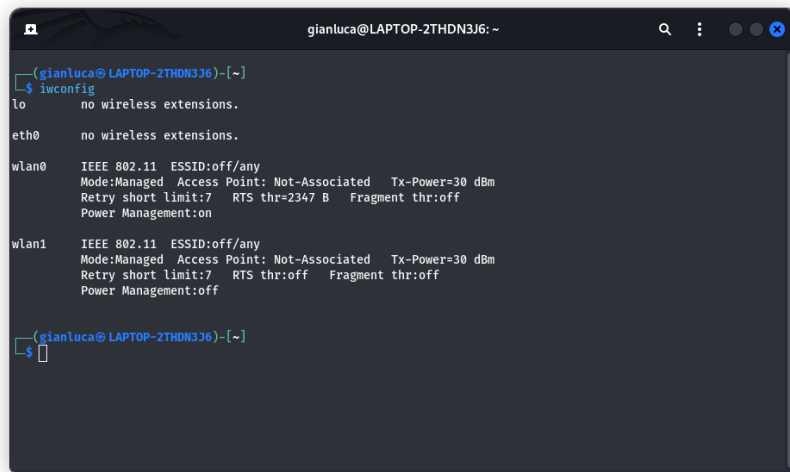
PHY Interface Driver Chipset
phy0 wlan0 rtl8821ae Realtek Semiconductor Co., Ltd. RTL8821AE 802.11ac PCIe Wireless Network Adapter
phy1 wlan1mon ath9k_htc Qualcomm Atheros Communications AR9271 802.11n
      (mac80211 station mode vif enabled on [phy1]wlan1)
      (mac80211 monitor mode vif disabled for [phy1]wlan1mon)

(gianluca@LAPTOP-2THDN3J6)~$

```

Figure 3.7: stop result

By using again the `iwconfig` command it's more clear the change.

A terminal window titled 'gianluca@LAPTOP-2THDN3J6: ~' showing the output of the 'iwconfig' command. The output lists network interfaces: 'lo' (no wireless extensions), 'eth0' (no wireless extensions), 'wlan0' (IEEE 802.11, ESSID:off/any, Mode:Managed, Access Point: Not-Associated, Tx-Power=30 dBm, Retry short limit:7, RTS thr=2347 B, Fragment thr:off, Power Management:on), and 'wlan1' (IEEE 802.11, ESSID:off/any, Mode:Managed, Access Point: Not-Associated, Tx-Power=30 dBm, Retry short limit:7, RTS thr:off, Fragment thr:off, Power Management:off). The prompt is '(gianluca@LAPTOP-2THDN3J6)-[~]' and the cursor is on a new line.

```
(gianluca@LAPTOP-2THDN3J6)-[~]
$ iwconfig
lo          no wireless extensions.

eth0       no wireless extensions.

wlan0      IEEE 802.11  ESSID:off/any
Mode:Managed  Access Point: Not-Associated   Tx-Power=30 dBm
Retry short limit:7  RTS thr=2347 B  Fragment thr:off
Power Management:on

wlan1      IEEE 802.11  ESSID:off/any
Mode:Managed  Access Point: Not-Associated   Tx-Power=30 dBm
Retry short limit:7  RTS thr:off   Fragment thr:off
Power Management:off

(gianluca@LAPTOP-2THDN3J6)-[~]
$
```

Figure 3.8: checking stop

NB

When I got back to *Managed* mode from the *Monitor* one, in order to be able to connect again to Wi-Fi networks (so to restart the Network Manager), the simplest way I found is to restart the system.

3.2. Wireshark

The sniffing operations couldn't be performed in an anechoic chamber, so, in order to highlight as much as possible the Wi-Fi activities of the four subjects, I placed them next to the Alfa Adapter Antenna so that the Received Signal Strength Indicator (RSSI) could be the best possible.

"The collection of probe requests from multiple devices that implement the randomization of MAC addresses is complex because we cannot discriminate against more than one device at a time, so I collect them separately for each device." [4]

For all the subjects, I considered the following conditions:

| Mode Nr | Wi-Fi ON | Save Battery ON | Active Screen | Plane Mode ON |
|---------|----------|-----------------|---------------|---------------|
| 1 | Y | N | Y | N |
| 2 | Y | Y | Y | N |
| 3 | Y | N | Y | Y |
| 4 | Y | Y | Y | Y |
| 5 | Y | N | N | N |
| 6 | Y | Y | N | N |
| 7 | Y | N | N | Y |
| 8 | Y | Y | N | Y |
| 9 | N | - | - | - |

Where:

- *Mode Nr* identifies the particular combination of concurrent conditions on a device.
- *Wi-Fi ON* specify whether the Wi-Fi toggle is ON (Y) or not (N). When ON, all the devices were not associated to any AP.
- *Save Battery ON* specify whether the Save Battery mode is active (Y) or not (N).
- *Active Screen* specify whether the screen is active (Y) or in stand-by mode (N). For all the captures, when active, the screen has shown the Wi-Fi settings.
- *Plane Mode ON* specify whether the Plane Mode is active (Y) or not (N).

From mode number 9 (since the Wi-Fi toggle was OFF), no probe requests were generated and thus the related captures are not considered in the final project folder.

Each capture was executed in a 15 minute time slot so that the data volume generated

could be enough for the analysis.

In order to display in Wireshark [9] only the *Probe Request* messages, I applied the following filter:

wlan.fc.type_subtype==0x0004

Each capture has been exported into two formats: pcap and pcapng.

For each device, there are available two sets of capture files:

- *Not filtered*

set of non filtered/unaltered captures: these files contain all kind of packets that can be captured in monitor mode (beacons, probe requests, probe replies, ...)

- *Filtered*

set of filtered captures: these files contain only probe requests (obtained through the *File > Export Specified Packets...* Wireshark option)

3.3. Colab Notebook

The Colab page was elaborated over the examples provided by Prof. Alessandro E.C. Redondi.

The code is commented for a deeper explanation instead here I'm just providing an high level view of it.

The notebook code is organised in the following sections.

3.3.1. Import & Installation

I installed and imported libraries for different purposes: plotting, working with REST APIs and Wireshark, ...

3.3.2. Capture file selection and reading

In this section, it's possible to select the capture file of interest once the following data are provided:

- *Device*: choose one of the four devices, by commenting conveniently the rows.
- *Filtered/Not filtered version*: of course the non filtered files require more time to be parsed/read.
- *Mode number*: from 01 to 08.
- *File version*: if more than one file per mode is available.

3.3.3. Frame Validity check

In order to consider a frame *VALID*, I must be able to recover all the information I'm interested in, which are:

- Type and Subtype
- Timestamp
- Source Address
- RSSI (Received Signal Strength Indication)
- SSID (Service Set Identifier)

Among the *VALID* frames, I'm interested only in *Probe Requests* so I can discard all the

other ones.

3.3.4. Probes filtering according to RSSI

At the moment, the dataset is made of all the captured valid probe requests but, since the sniffing operation was performed in a noisy environment, it's plausible that the dataset contains also probe requests coming from devices which do not belong to the subject set of this project, so these must be identified and discarded.

This objective can be achieved by leveraging on the RSSI concept:

"In an IEEE 802.11 system, RSSI is the relative received signal strength in a wireless environment, in arbitrary units. RSSI is an indication of the power level being received by the receiving radio after the antenna and possible cable loss. Therefore, the greater the RSSI value, the stronger the signal. Thus, when an RSSI value is represented in a negative form (e.g. -100), the closer the value is to 0, the stronger the received signal has been." [8]

I verified that Android devices emit a burst of Probe Requests characterized by different emitted powers and so received with different quality levels (= RSSI) instead Ios devices are characterized by bursts of more stable power.

Considered the Android devices behaviour, the filter is able to keep all the frames that belong to a burst in which at least one packet has a RSSI value greater than -40 [dbm].

This threshold has been suggested in the paper "A dataset of labelled device Wi-Fi probe requests for MAC address de-randomization" [4]

NB: not always this threshold is enough to discard noisy signals so I increased it, for instance to -30 [dbm], or I performed a different version of some dirty captures.

In order to highlight the effect of this filter it's possible to choose in the notebook whether using it or not.

In conclusion some interesting plots are reported.

3.3.5. Vendors representation

I wanted to verify whether the probe requests leaked some interesting information about the vendor/manufacturer of a device.

Starting from a MAC address, it is possible to extract the vendor/manufacturer by looking at the first half of it which is also named OUI (= Organizationally Unique Identifier).

"An organizationally unique identifier (OUI) is a 24-bit number that uniquely identifies a vendor, manufacturer, or other organization. OUIs are purchased from the Institute of Electrical and Electronics Engineers (IEEE) Registration Authority by the assignee (IEEE term for the vendor, manufacturer, or other organization)." [7]

In order to obtain the vendor name starting from the vendor OUI, I have found an online application called "MACLookup" [5] which also provide a free REST API.

It has the following rate limits:

- Requests per seconds: 50
- Requests day 1.000.000 The day is reset after midnight GMT+0.
- Max API Keys 10

In conclusion, there's an interesting pie plot.

3.3.6. MAC Randomization Check

In this section I want to highlight the existence of random and true MAC addresses in the dataset.

A MAC address consists of 6 [Bytes] organized as follow:

- 3 B [OUI: Organizationally Unique Identifier -> Manufacturer info]
- 3 B [NIC: Unique indentifier of the network card]

Take the first Byte, then take the second least significant bit:

- if 0 -> REAL MAC addr
- if 1 -> FAKE MAC addr

Take the first Byte -> take the least significant bit:

- if 0 -> unicast
- if 1 -> multicast

Assuming only Unicast communications (last bit = 0), you have a FAKE MAC address only in these scenarios:

- 2 (0010)
- 6 (0110)

- A (1010)
- E (1110)

3.3.7. Final metrics: Average RSSI, Burst size and Average IAT

In this last phase, I was able to compute interesting metrics as:

- *Average RSSI* of all the probe requests belonging to a particular burst which is identified by a source address.
- *Burst size* so the number of probe requests that are part of a burst.
- *Average IAT* (= Inter Arrival Time) of all the probe requests belonging to a particular burst.

The IAT is

"the time between the occurrences of two events. In queuing theory, it is the time between the arrivals of two consecutive customers to a queue." [3]

4 | Conclusions

Among the concurrent conditions applied to different devices, the one that is affecting the most the behaviour and the volume of bursts of probe requests is the *Active Screen* one: so the Saving Battery and Plane Mode options are not determinant.

I'm presenting now some peculiarities for each device.

4.1. Android

This class of devices is characterized by burst of probe requests of extremely varying emitted power and so power received (RSSI).

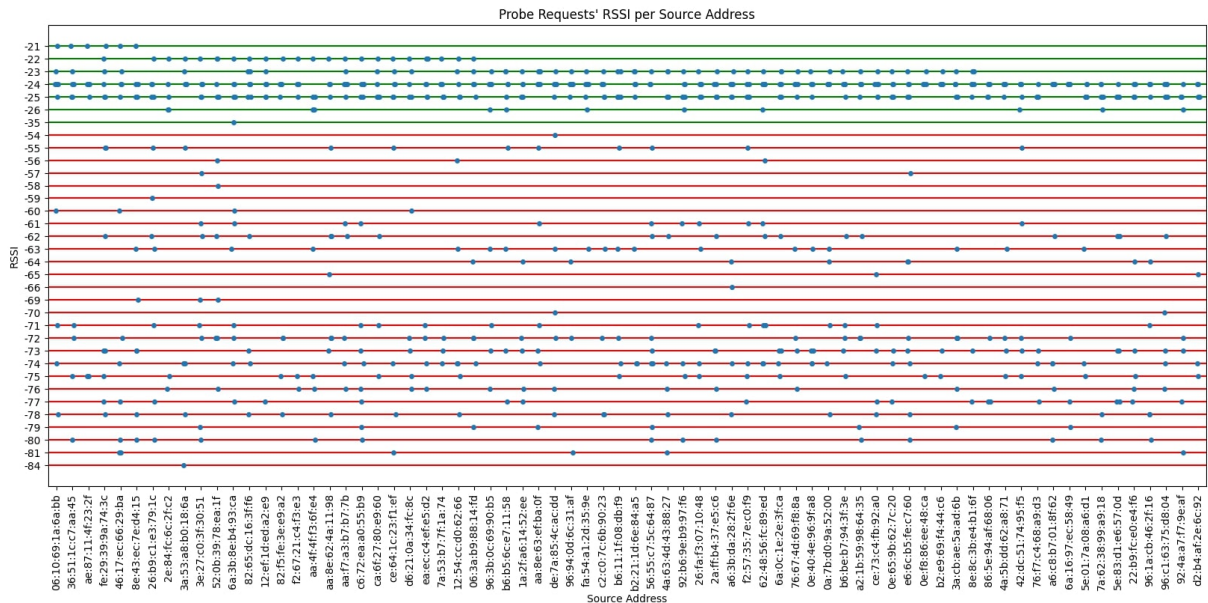


Figure 4.1: Android-SamsungA53-Mode01, Variable emitted power

When the screen is active the burst frequency is much higher than the stand-by mode.

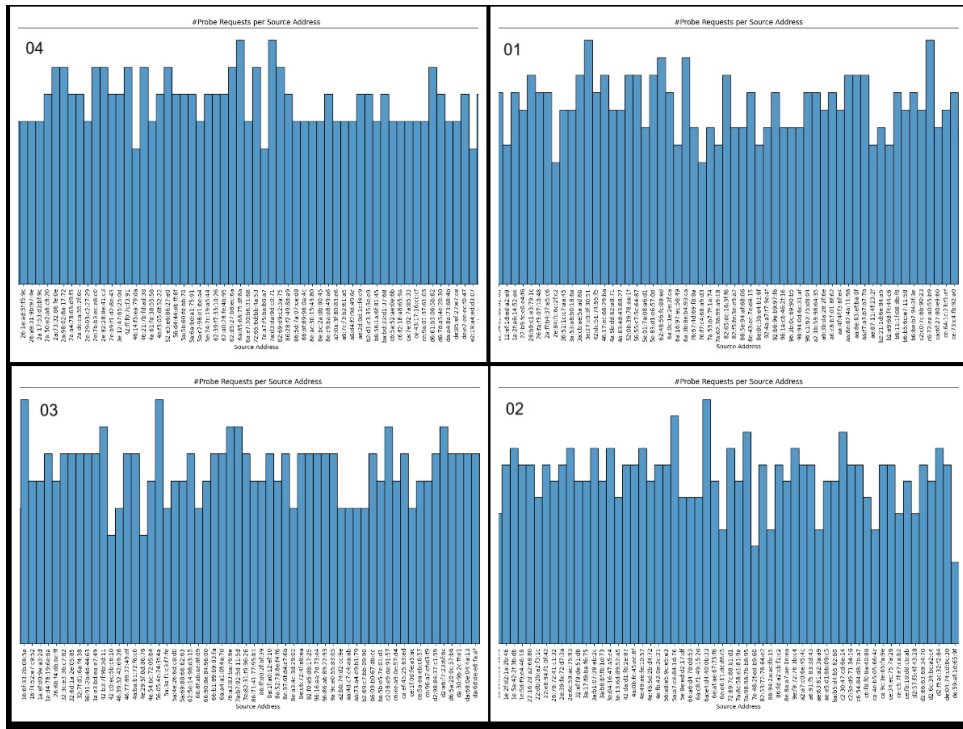


Figure 4.2: SamsungA53-Active Modes

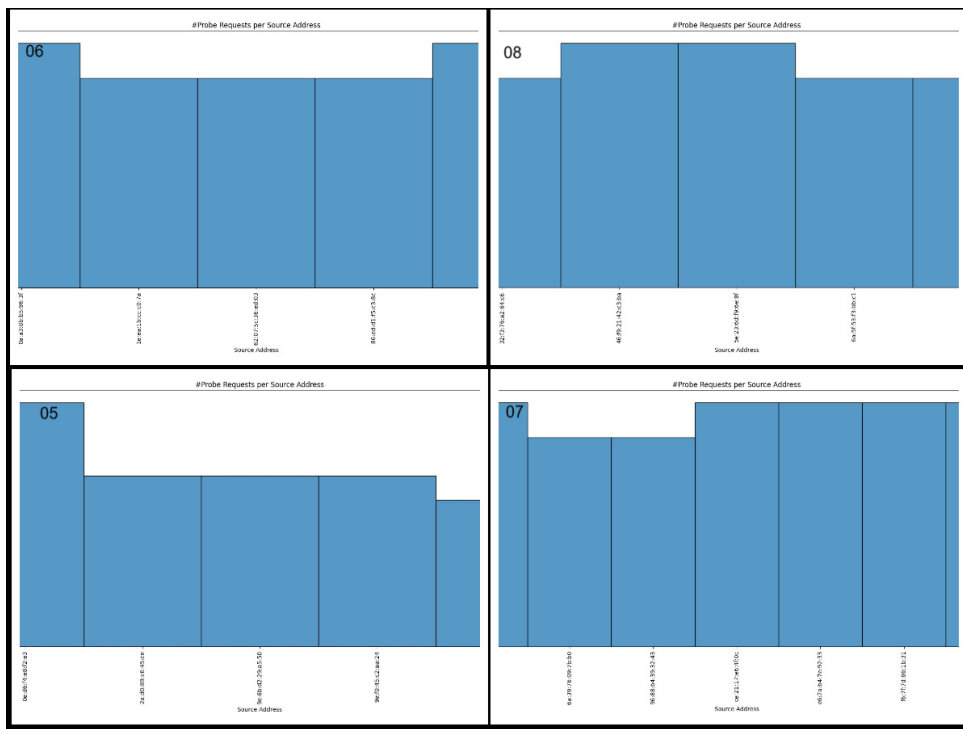


Figure 4.3: SamsungA53-StanbBy Modes

4.1.1. Samsung A53

The random MAC addresses generated are always burst-to-burst different, in particular all the six octets, including the OUI which always leads to a *NO COMPANY* result.

4.1.2. Huawei Mediapad T5

The random MAC addresses generated are always burst-to-burst different, but the first three octets are the same and the remaining ones differ: all the random source addresses generated apparently belong, as a vendor/manufacturer, to *Google, Inc.*

4.2. Ios-iPad

This class of devices is characterized by burst of probe requests of quite stable emitted power, so that the average RSSI value for each random MAC address is as well lower than the Android devices.



Bibliography

- [1] *[802.11] Wi-Fi Connection/Disconnection process*. URL: <https://community.nxp.com/t5/Wireless-Connectivity-Knowledge/802-11-Wi-Fi-Connection-Disconnection-process/ta-p/1121148>.
- [2] *Airmon-ng*. URL: <https://www.aircrack-ng.org/doku.php?id=airmon-ng>.
- [3] *IAT*. URL: <https://hatchjs.com/what-is-interarrival-time/>.
- [4] Luigi Atzori Lucia Pintor. “A dataset of labelled device Wi-Fi probe requests for MAC address de-randomization”. In: *Elsevier* (2022).
- [5] *MAC Lookup Online App*. URL: <https://maclookup.app/>.
- [6] *Monitor Mode guide*. URL: <https://www.youtube.com/watch?v=WfYxrLaqlN8>.
- [7] *OUI*. URL: https://en.wikipedia.org/wiki/Organizationally_unique_identifier.
- [8] *RSSI*. URL: https://en.wikipedia.org/wiki/Received_signal_strength_indicator.
- [9] *Wireshark*. URL: <https://www.wireshark.org/>.

List of Figures

| | | |
|-----|---|----|
| 2.1 | Connection process | 3 |
| 2.2 | Scanning methods | 4 |
| 3.1 | ip addr result | 8 |
| 3.2 | iwconfig result | 9 |
| 3.3 | check kill result | 9 |
| 3.4 | start result | 10 |
| 3.5 | checking start | 10 |
| 3.6 | airodump-ng | 11 |
| 3.7 | stop result | 11 |
| 3.8 | checking stop | 12 |
| 4.1 | Android-SamsungA53-Mode01, Variable emitted power | 19 |
| 4.2 | SamsungA53-Active Modes | 20 |
| 4.3 | SamsungA53-StanbBy Modes | 20 |
| 4.4 | Ios-iPad-Mode01, Stable emitted power | 21 |

