My neighbor's flat smells like data

Gerard Fuguet (gerard@fuguet.cat)

Abstract

Electromagnetic waves are present in any part of the space that humans are involved to. They are apparently invisible to our eyes, we can't feel the sense of their presence without the correct tool for decodify them, but… even with it, is still impossible the reading of the content that hides us, because they are *"embarrassing"* and do not want to show themselves to anyone. Wi-Fi waves are similar to the microwave waves. Microwave can reveal their aspect with the help of its smell. Wi-Fi has the ability to be almost unseeable to others with encryption protocols, helping it a good camouflage. Also has the option to spread confidential data *"to the four winds"* when the communication channel is established as OPEN (and doesn't matter if it's produced in a short period of time). The processes that we focus on this white paper is to capture confidential data on the fly of a smart switch (IoT) device.

We will demonstrate how to get easily the Wi-Fi's name (SSID) and password of the owner's device although the password and encryption are very secure/strong. This situation motivated me to write it.

The intention is that vendors be aware and take action against this. We want to demonstrate the danger of open wireless to the consumers with a theory and practical approach view (simple as possible).

Table of Contents

1. Motivation

Free wireless networks have always been very tentative for those who want internet connection in a place that don't carry it. A tourist is a typical profile that looks for it when travels to a country with a non-data coverage plan. People living in Europe (for example), between the EU countries, there is no extra charge with their mobile operator [1], so it could reduce the needs to have a non-owned Wi-Fi (free or not, secure or not). Nowadays there are alternatives to avoid the use of external Wi-Fi's; buying a prepaid data plan, use an achieve roaming plan, ...

There is a strong Cyber consciousness and is increasing with the pass of the years but, seems is not enough because most vendors, who should help setting a good example, make serious mistakes in security IT terms. Connecting to a wireless that is not yours, is not a recommendation even if is secure or not (you are not the manager of that legitimate Wi-Fi and you don't know what is moving in background) but what is happen if the device establish a temporarily free Wi-Fi for the first setup? Need to connect to a Wi-Fi network, and OK if this network is belonging to us but it's OPEN, that's the problem and this paper is being written due to this. You also could say if the config takes some small time, is difficult intercept important owners data, right?

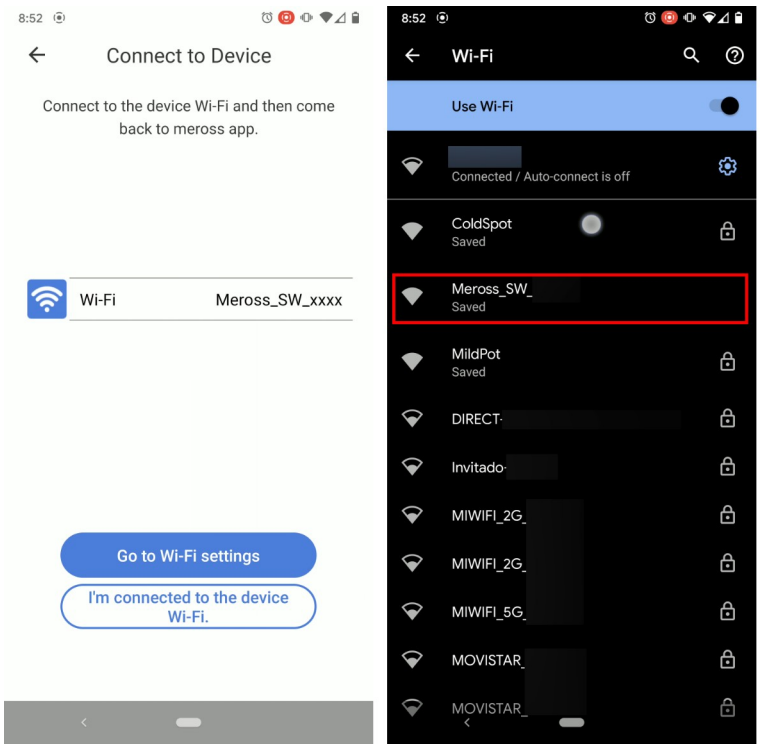

Figure 1: At left; part of the initial setup of *Meross* app showing the Wi-Fi schema. At right; the AP created by app with no padlock (means it is OPEN)

In the above figure, app does not tell the Wi-Fi to connect is OPEN (no security) but you can be aware doing the inspection with the scan functionality of the smartphone itself, in this example, through an Android device.

I want to explain a realistic case to try to stop the way of these type of configurations.

2. SmartLife

The "*smart*" word appears and is adding in conventional devices expanding the options to turn on and off (not limited too) from many sources not only by physical human interaction. Typical home appliances are made to be part of our lives with a clear mission, help in our tasks and reduce the time of home maintenance to invert this time in other things we like more. Remote control concept was very useful to control TV channels because if not, every time need the effort to get up to change looking for a desirable channel. The TV remote control is still present as TV screen concept but you also can do the same control with your phone (with some app). The distance in that case is imposed by your network not by the infrared ray emitted, so you don't need reside at front for swap between TV channels. The use of a smartphone to do this is very similar because, the phone is near to you like the remote controller, phone can be the first choice and the controller the backup in case of a fault.

Home automation was not as plug-and-play at the beginning, some additional elements were necessary. In nowadays, this is more easy because the infrastructure is almost in any home, you need not more than a Wi-Fi point of access with Internet connection. But let's the focus on the *smart wall switch* light part.

2.1. The Smart Wi-Fi Wall Switch

You know how is the behavior of a normal light wall switch, it have a simple mechanism to accomplish their function and can't ask it for more. Sometimes simple is better because, as less sophistication, less problems/troubles/headaches. The design is very closed so, any ad-don/new functionality, could also be hard to implement. What if we are interest in apply a scheduler to turn on and turn off light in some specific time? Is an amazing thing and useful if we are at outsides to convince thieves (as an example) we "*could be at inside*" but the installation turn into a great challenge for someone who has never been in contact with the electricity world. On the contrary, the *smart light switch* can do your life easier, this is one of the great advantage, can do the same with a pair of (basically) things, your smartphone and have an Internet connection available -a piece of cake, right?-.

2.1.1. Set it UP

The requirements of the installation of this particular *Meross* device are clear and well defined. You will need:

1. An account (free, of course!)

2. An SmartPhone (*Android* or *iOS*, doesn't matter!)

3. Internet connection (comes with your home gateway!)

4. Have a neutral wire on your electrical installation (what & why?)

For the last point 4. the reason is because this new device have electronic inside and needs to be powered, so the electrical circuit needs to be close by a positive (+) and a negative (-) or in terms of electricity, live (+) and neutral (-). Keep in mind, without the

success of this point nº 4. the other before points are useless. A basic knowledge is needed, but don't worry, if you want test this *PoC*, you are under good hands! ;) .

Is almost impossible do the best tutorial that feed all of all to everyone homes… each case could be very different, most important is understand the things and you will always win.

- As step 1, need locate the old switch to replace. What if more than one switch controls the behavior of the light? In this situation, you are dealing with a 2-way switch (and you will need to use the same product as mine, an *MSS550X EU/UK* version [2]). If you have two of 2-way, you will need to replace one (not both!) and if you have three vs one light, need take care not use the cross switch for be replacing. Let's explain the following electrical schematic:

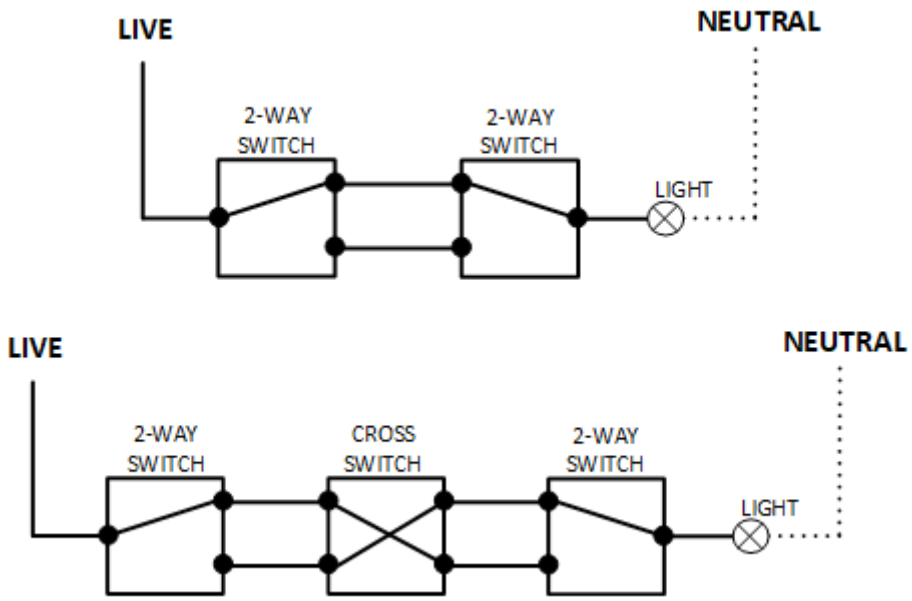

Figure 2: At top; a schema with two switches against one light. At bottom; same schema concept plus a cross switch

In the case of the two 2-way switches (top schema), there are 2 cables interconnected between them, this is because if one of them is clicked, light toggles status (if *ON* then *OFF*, if *OFF* then *ON*). For this reason, only one of them need to be replaced.

In the case of two 2-way and one cross (or more!) switch exist in your home to control same light, you can still replace one of the 2-way switch, but not the cross switch. This scenario can handle more than one cross switch if you have more than three switches. The formula, (having 3 switches as minimum) to calculate it is: nº of switches – 2 (for example, if you need 6 switches in total, 4 cross switches will needed).

- Step 2: Once identified the traditional 2-way replaceable switch, you need find the neutral cable in a near fuse box but before, remember to turn off the general power electricity! And pass the cable with a help of a tool to push-pull it.

Figure 3: A type of fuse box on a Spanish home

The cables colors usually are; **blue** for neutral, **brown**/**black**/**grey** for live/phase [3]. Generally, between 2-way and/or cross switches the preferred colors are **grey** or **black**.

When neutral cable is passed to the switch box, rest of the cables are used to be connected to the smart switch.

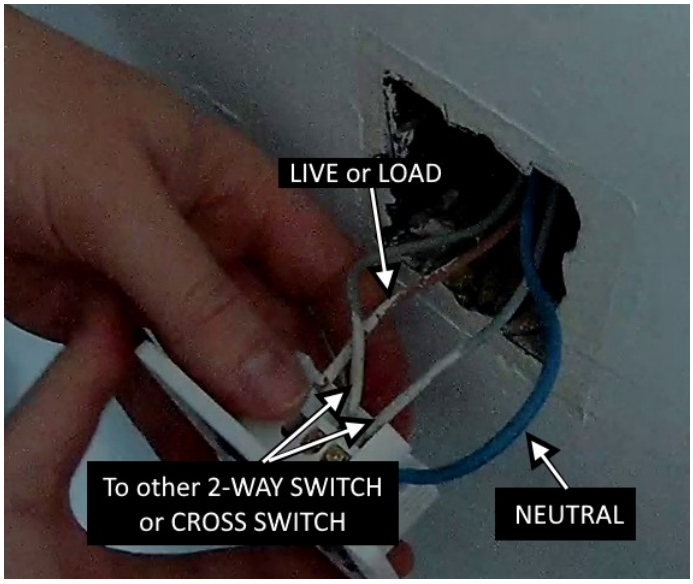

Figure 4: Taking advantage of the previous cables plus neutral

The connection on *MSS550X* is as follows; **Grey** cables to the ***L1*** and ***L2***, **brown** to ***C*** and **blue** to ***N***.

- Step 3: This is the software part (tested under version *2.26.2*). Download the *Meross* app [4] [5] and sign up with a valid email address (if you want recover the password if it someday missing ;)).

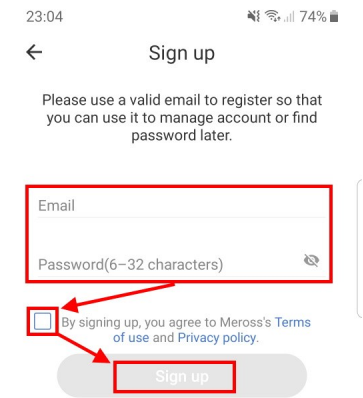

Figure 5: *Meross* app: Sign up process under Android version

Choose a different password (for security reasons) for the email that will be used as your *Meross* account, and you will have the privilege to add *Meross* devices.

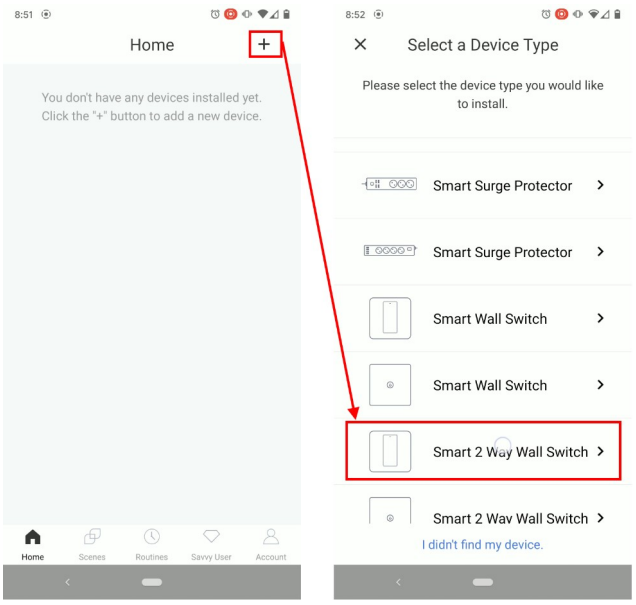

Figure 6: *Meross* app: Adding a device under Android version

Tap the plus button and search for the *Smart 2 Way Wall Switch*.

At this point, you will receive some instructions on how to install it at hardware part, but you are at software, so sure you did it! You can skip it to gain some time until you

reach the part of the Wi-Fi connection that you saw in figure 1, then next figure will continue to guide you:

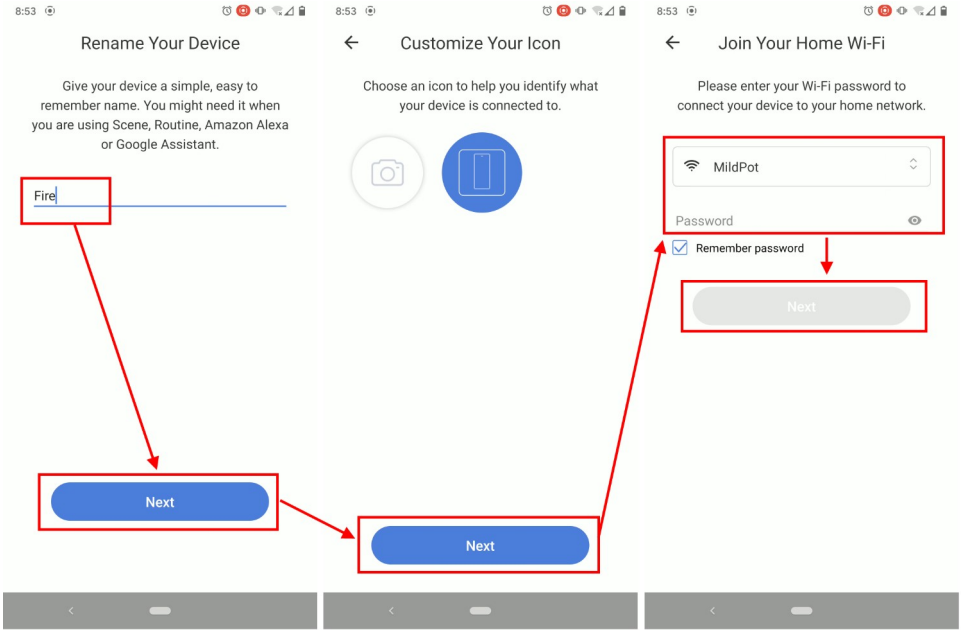

Figure 7: *Meross* app: Rename, customize device and join Wi-Fi under Android version

Almost done! You can now rename it, *Customize Your Icon* (not mandatory) and find your home Wi-Fi network including the correct password in order to show "*some light*" to the device that is looking for an exit to Internet. If all goes well, you will be congratulated by app. That was easy right? The light now turn on & off magically through the app. Having a smartphone nowadays is very important if you want to enjoy the IoT world!

3. Bridging the Data

Set UP is complete but at this point you are asking ¿How it works at network level? ¿What are their interactions between app and the *smart wall switch*? The answer is to use *"something-in-the-middle"* tactic in order to sniff data pass-through.

There are many methods to get success with it (each has its advantages and disadvantages), one affordable for inspect into mobiles is the use of a proxy [6], but it requires some set up in your smartphone and sometimes can't be implemented as a global proxy for catching all packets… for this reason, I prefer something more transparent with *"0-config"* on involved actors and ensuring nothing is lost in the way. *Machine-in-the-middle*, as explained in *Wireshark* Wiki's [7], is another concept similar to *Man-in-the-Middle (MitM)* that basically consist in converting your machine into a network switch, so can leading the layer 2.

The disadvantage is the configuration you need in machine and the additional hardware (depending on the medium the devices uses). You may need an additional Ethernet NIC

if in both around devices has this medium but, in this case, all is through Wi-Fi (smartphone and the *smart wall switch*).

3.1. Machine-in-the-Middle

Let's see how to achieve the implementation under *Kali Linux*.

As explained before, all the transmission is through Wi-Fi, it means we will need that 1 network NIC act as Access Point and other act as client/station mode.

Before to explain the methodology, is important to know that you cannot use Wi-Fi interface in client/station mode for build the bridge because it hasn't the 4-address format into the 802.11 frames [8] so... How can we use other Wi-Fi acting as client under this limitation? Using a device with these capabilities connected to the Ethernet, in my case, using a *Gargoyle* router (specifically, a gateway model *WZR-HP-G300NH2 and software version 1.12.0*), an *OpenWRT* based system [9].

In figure 1 of the chapter 1, we saw that *MSS550X* creates an AP (*HotSpot*). The idea is to know what is moving between smartphone (using the official *Meross* app) and the *smart wall switch*. When the *HotSpot* disappears, the *Meross* device connects to the Wi-Fi that we told via app, this conversation won't be covered in this paper but we will offer it the option of going online for have a *"happy ending"* (complete the joining process successfully).

To make things easier and for prepare the scenario better, let's do a checklist for all needed elements we need to implement in the solution:

- **SmartPhone:** Connects to the controlled Hacker's Laptop (SSID named *ColdSpot*)

- **Hacker's Laptop:** *Machine-in-the-Middle*

Bridge: *br0* {*eth0* & *wlan0*}, *wlan0* acting as AP, *eth0* acting as Wi-Fi client through *Gargoyle* router

SSID: *ColdSpot*

- ***Gargoyle*** **Router:** Acting as Wi-Fi client

Bridge Management IP: 192.168.3.1

- ***Meross*** **device:**

SSID: *Meross_SW_XXXX*

(for the moment, we don't know the IP, network…)

- **Home Router/Gateway:** Brings Internet to *Meross* device

IP: 192.168.1.1

Network: 192.168.1.0/24

DHCP: ON

SSID: *MildPot*

Now let's draw a diagram to have a good shot of all devices:

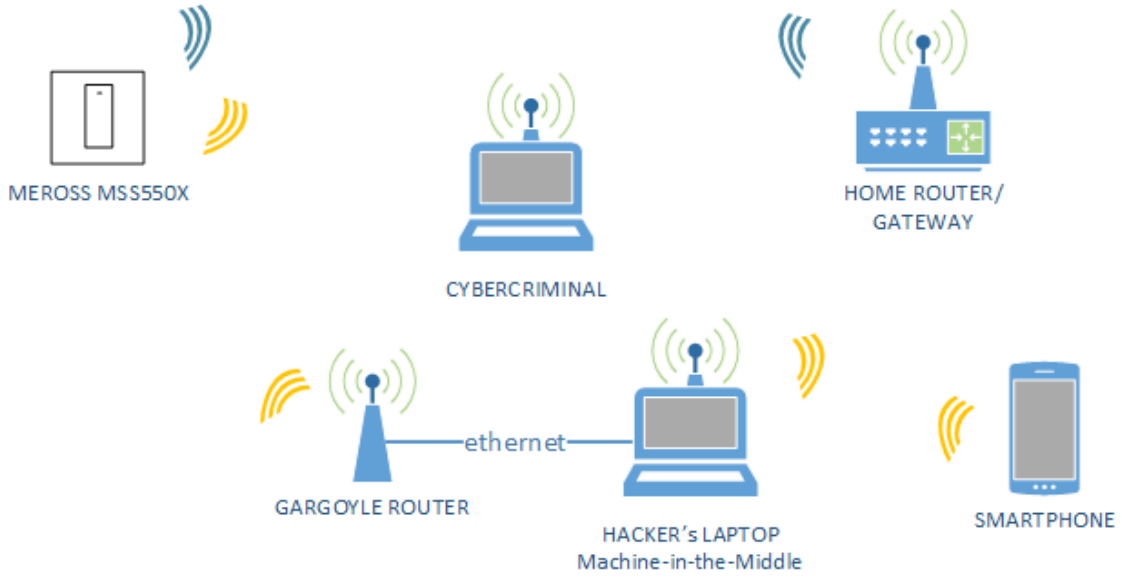

Figure 8: *Machine-in-the-Middle* scenario

Hey! Why is here a Cybercriminal object? Well… There can always be one stalking us sniffing over the air!

The yellow waves are the traffic involved between smartphone and *MSS550X*. The blue ones are traffic between *Meross* and gateway that happens when the set up is complete (not sniffed). The first device to interact is the smartphone that instead of doing the connection against the *Meross* SSID, it will connect to the hacker's SSID and packets are forward to the network of *Meross* through the Wi-Fi client established by the *Gargoyle*. Responses will occur in the reverse order using the same infrastructure.

Moving now into the practical mode.

3.1.1. Catching the Data

The *Machine-in-the-Middle* infrastructure is clear now according to figure 8, we need configure to get it working in a *Kali*. Let's do it step by step:

0. If *MSS550X* is already set up, remove it from the app (you will need Internet access) so that you can configure it again for *sniffing*.

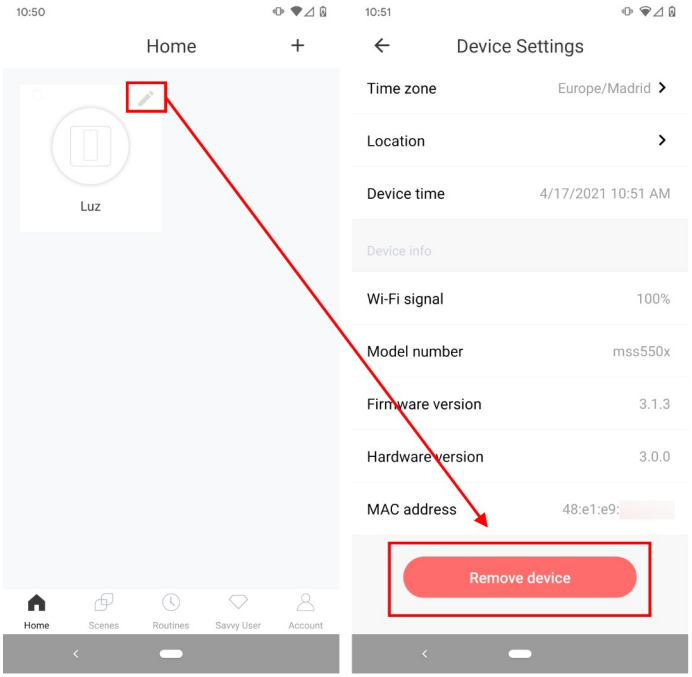

Figure 9: Removing the device on app under Android version

The account (and credentials) will remains, you don't need a log out or clear the app cache, remember that our objective is to catch the data between the mobile phone (*Meross* app) and the *smart switch.*

Prepare *Gargoyle* router; before powering on your computer, connect the Ethernet cable to it and turn on the *Gargoyle*.

Step 1: *PoC* under...

1. Reviewing the OS version where *PoC* is running. Invoke first the *"super user"*.

```
┌──(kali㉿kali)-[~]
└─$ sudo su
┌──(root💀kali)-[/home/kali]
└─# # PoC under ...

┌──(root💀kali)-[/home/kali]
└─# grep VERSION /etc/os-release
VERSION="2020.4"
VERSION_ID="2020.4"
VERSION_CODENAME="kali-rolling"

┌──(root💀kali)-[/home/kali]
└─# uname -v
#1 SMP Debian 5.9.1-1kali2 (2020-10-29)

┌──(root💀kali)-[/home/kali]
└─# uname -r
5.9.0-kali1-686-pae
```

Figure 10: *Kali* – Elevate user and showing the OS version

Step 2: Preparing *Machine-in-the-Middle* environment

1. Get connected to Internet to refresh the *Kali* repositories for obtain later the *hostapd* [10] package.

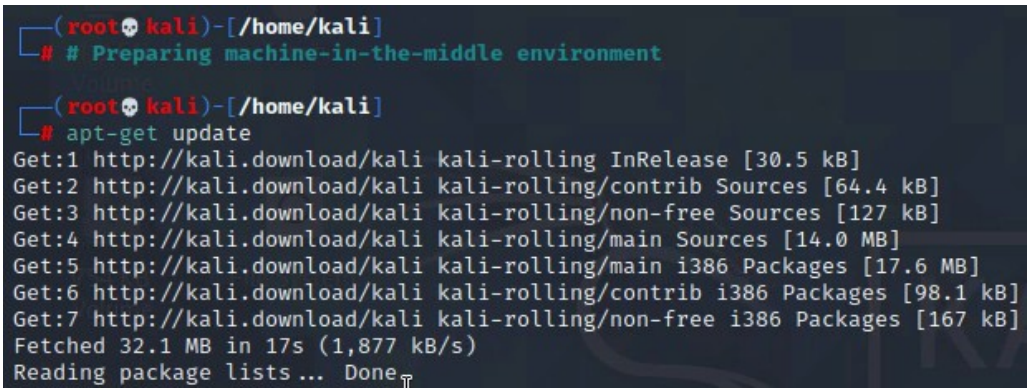

Figure 11: *Kali* – Update repositories

2. Install the *hostapd* package with the help of *apt-get* command so can bring up a *HotSpot*.

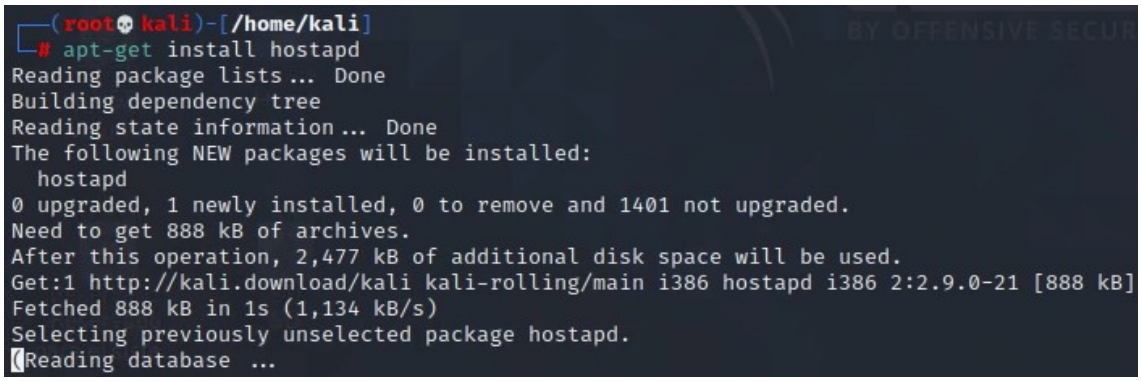

Figure 12: *Kali* – Installing *hostapd*

Step 3: Creating *hostapd.conf*

1. *Hostapd* needs minimal stipulated guidelines into a config file, then it will knows what is the name of the SSID, the working channel, encryption type and password for this AP (will be used later).

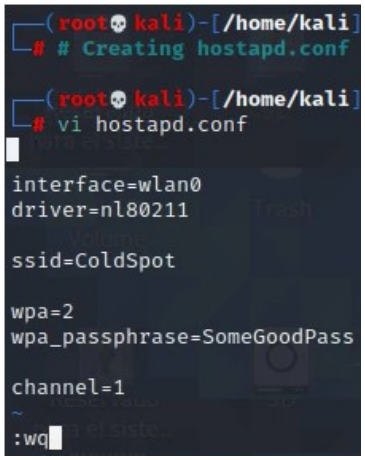

Figure 13: *Kali* – creating *hostapd.conf* via *vi* editor

Step 4: Discovering Network Parameters

1. Using the GUI, connect to the *Meross* Wi-Fi so know what is their IP.

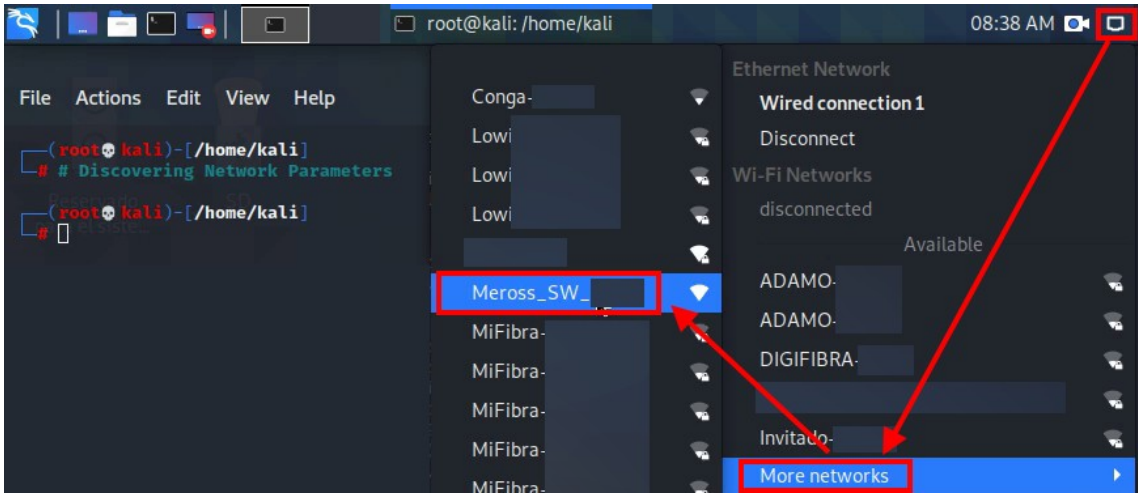

Figure 14: *Kali* – Connecting to the *Meross* Wi-Fi

Step 5: Our Gateway is their IP

1. Using *route* tools reveals our gateway (or gateways if we are connected to other networks), and also it tells us the IP of *MSS550X*.

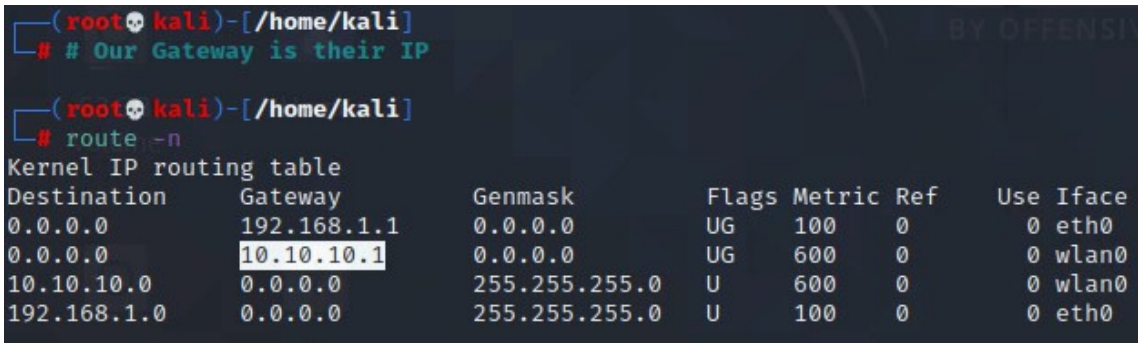

Figure 15: *Kali* – Route command to show gateway/s

2. Once the parameter is got, Wi-Fi can be disconnected.

Step 6: Tell *Gargoyle* (*OpenWRT* based System) to act as Wi-Fi client for connecting to the *Meross* Network

1. Internet connection is received through the *Gargoyle* configured as Wi-Fi client against our gateway router (we preferred to take advantage of the Ethernet port to have Internet connection). As can be seen in 3.1, the bridge IP is configured as 192.168.3.1, it uses *relayd* [11], a second virtual IP of the network to extend is assigned. If you don't remember or for to be sure it works through this mechanism, a *traceroute* can be done against the Internet gateway to be determined (not mandatory, of course).

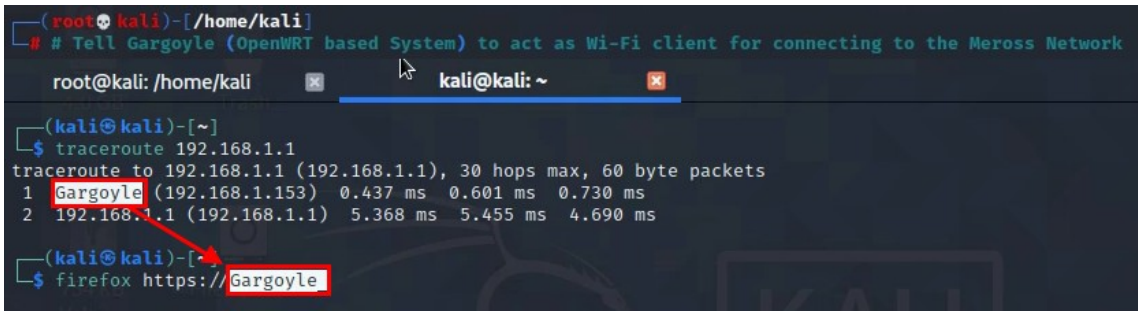

Figure 16: *Kali* – Determining the *Gargoyle's* IP and opening web for config

2. We use *hostname* instead of IP and new tab was opened with *CTRL+SHIFT+T* because of execution of *firefox*.

3. Once a login to the web page is successful, the *Gargoyle* device needs to be configured as Wireless *Bridge/Repeater* and it's time to put the *Meross* IP following the steps of next figure.

Figure 17: *Gargoyle* – Set UP the client Wi-Fi mode

4. When changes are saved, close *firefox* or kill process on terminal (*CTRL+C*).

Step 7: Disable IPv4 and IPv6 all interfaces via *Network Manager* for transparency (GUI mode)

1. Click to the network icon at *top – right* near the time, to editing the both network connections.

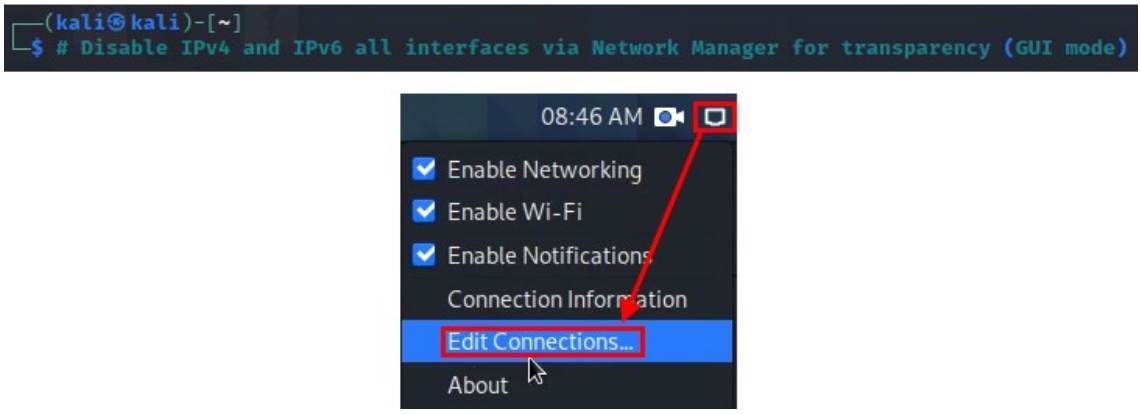

Figure 18: *Kali* – Editing network connections through GUI

2. Double click to edit the *Wired connection 1* and disable IPv4 & IPv6 protocols. Do the same for the Wi-Fi.

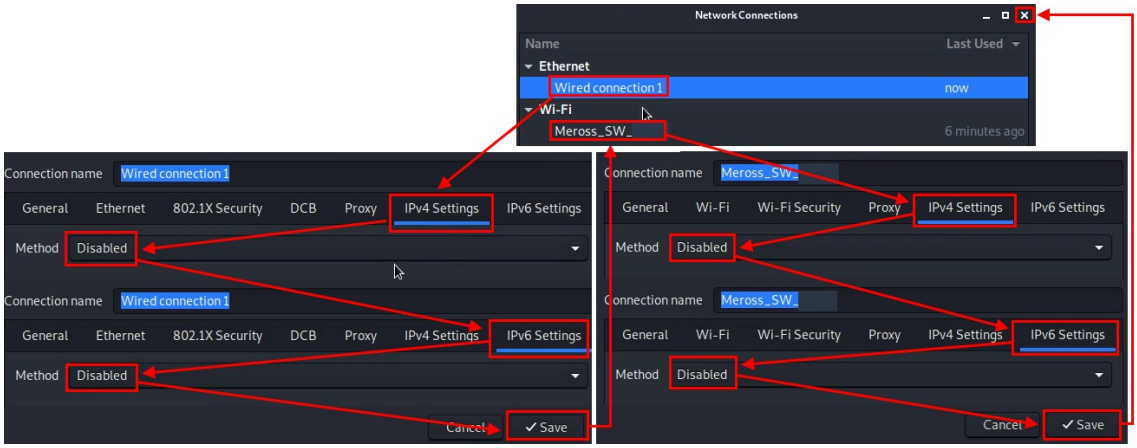

Figure 19: *Kali* – Disabling IPv4 & IPV6 protocols under *Network Manager*

3. Again, click to the network icon at *top – right* near the time to disable and repeat operation to enable (the network will be rebooted).

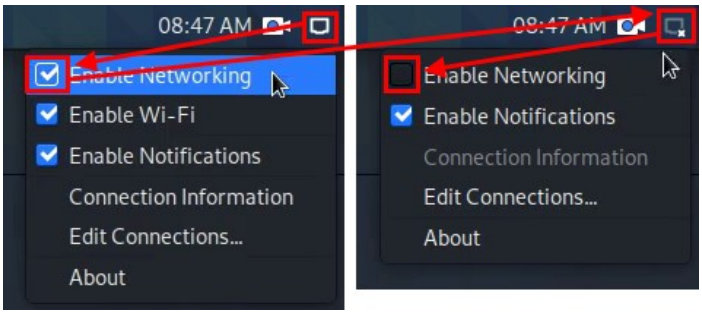

Figure 20: *Kali* – Rebooting network using *Network Manager*

Step 8: Avoid conflicts stopping *Network Manager*

1. Using *hostapd* and *Network Manager* can conflict if both are up & running [12]. *Network Manager* must be stopped in order to use the *HotSpot* functionality with other different software and avoid errors/problems.

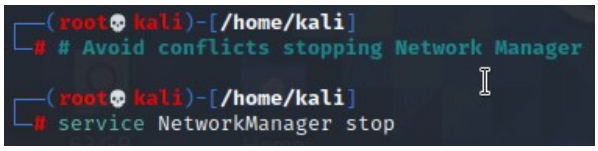

Figure 21: *Kali* – Stopping *Network Manager* service

Step 9: Execute now your *HotSpot*

1. Is the moment to execute *hostapd* tool with their config file we created before (*hostapd.conf*) so will brings up an AP with our desired parameters.

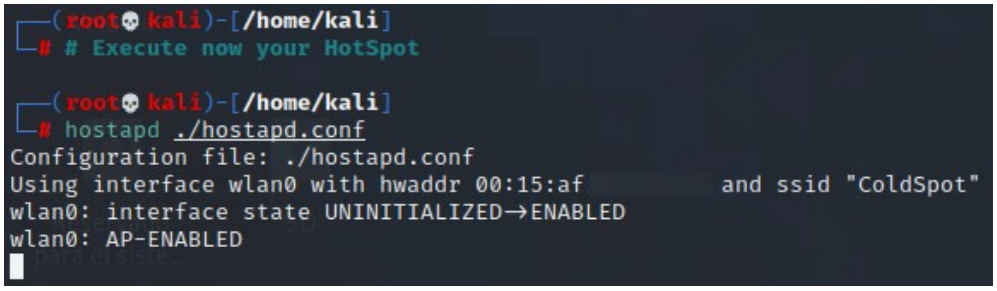

Figure 22: *Kali* – Executing *hostapd* with the config file

Step 10: Let's create the Bridge

1. In a new tab on terminal and under root user, *eth0* and *wlan0* will be part of new interface called *br0* (the bridge). Follow the next commands using *iproute2* [13] to create the bridge.

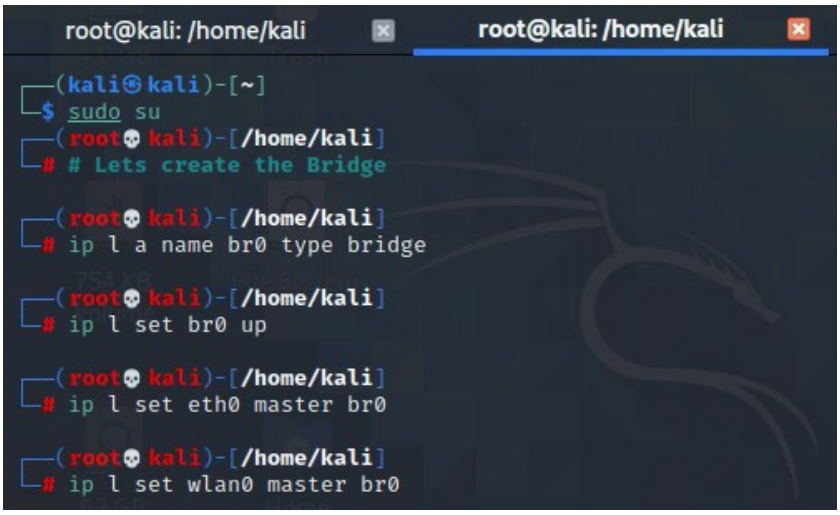

Figure 23: *Kali* – Creating a bridge using *iproute2*

Step 11: Checking the just created Bridge

1. Before moving forward, need to know if the bridge is stable and support some *"heavy weight"*. After issue the following command, if you can see forwarding on both interfaces, the bridge is working well.

Figure 24: *Kali* – Checking the state of the bridge

Step 12: Time to Monitoring :)

1. Mounting a bridge without sniffing it is useless, so start *Wireshark* to intercept one of the interfaces. You don't need monitor on both, all traffic that goes from phone to *Meross* and reversal is passing for both interfaces, so monitoring in one of them is enough (in this case we select *eth0*).

Figure 25: *Kali* – Starting *Wireshark*

2. Is the turn to the smartphone. Replicate the steps from figure 6 of the chapter 2.1.1. and onwards, then *Wireshark* will have something to digest :D !

At completion of the *MSS550X* set up, packets appears… Of all of them, the ones that catch our attention, are those of them that have the *HTTP* protocol. It appears to be *json* requests, subtrees are expanded to do a better inspection.

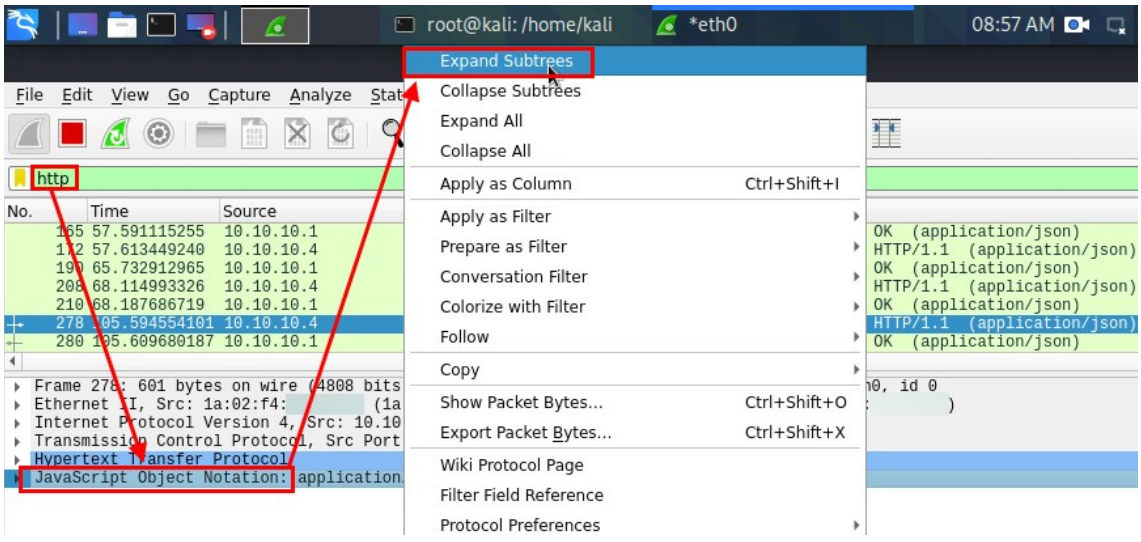

Figure 26: *Wireshark – json* requests in *HTTP* protocol

3. Looking for the last *POST* request method. A *right click* hit over *JavaScript Object Notation* will show the window which include *Expand Subtrees* option. The *json* in their format takes this form:

```
{
    "header": {
        "from": "http://10.10.10.1/config",
        "messageId": "                                ",
        "method": "SET",
        "namespace": "Appliance.Config.Wifi",
        "payloadVersion": 1,
        "sign": "                                ",
        "timestamp": 1616745231
    },
    "payload": {
        "wifi": {
            "cipher": 6,
            "password": "SXNJbXBvc3NpYmxl",
            "encryption": 7,
            "bssid": "00:22:07:        ",
            "channel": 6,
            "ssid": "TWlsZFBvdA=="
        }
    }
}
```


Figure 27: *json* request structure taken from the capture

4. What?! What our eyes are seeing? The password of our home gateway wireless is here? The *"ssid"* not seems to be ours… so what's going wrong? We can deduce *ssid* parameter is coded under *Base64* because the equal sign gives it away. Do you want to decode it quickly, offline and in a GUI mode? *Wireshark* can help you! A right click onto the incomprehensible string and...

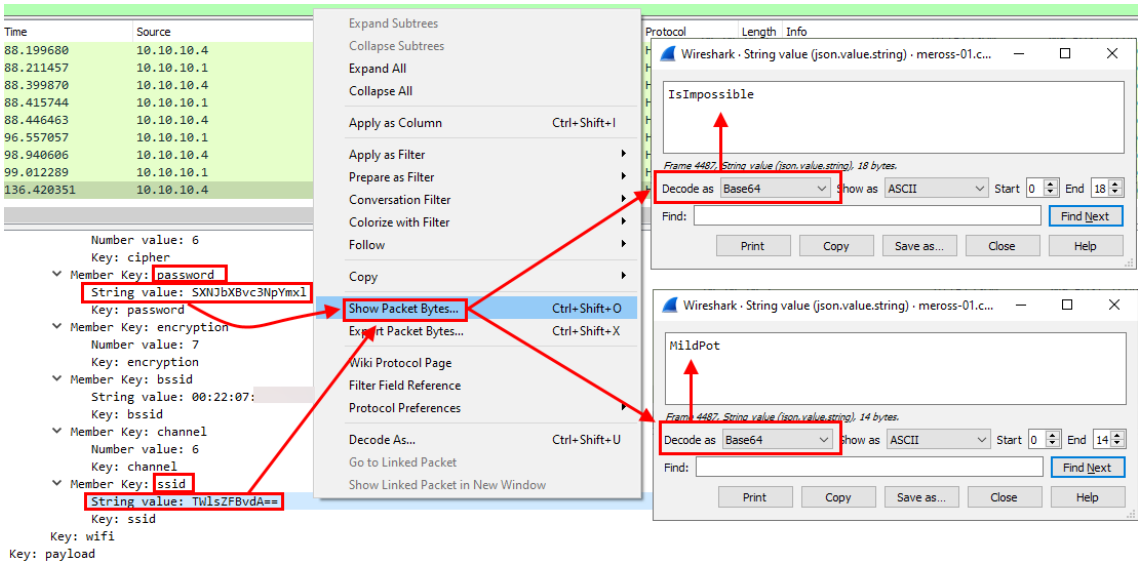

Figure 28: *Wireshark* – Decoding strings as *Base64*

This is a very bad idea… It uses an OPEN (non encrypted) Wi-Fi over an *"almost"* plain content? Yes, anyone sniffing at surroundings can easily get your Wi-Fi credentials. The crook's creativity may be endless.

3.1.2. Windows too

Windows can support a *Machine-in-the-Middle* setting creating a bridge between an Ethernet & Wi-Fi interfaces but not in the same way than *Kali Linux* does. Roles in interfaces change; The Wi-Fi will act as client and Ethernet act as *HotSpot* because "*layer 2 bridging is prohibited between the AP adapter and any other adapters in the system*" [14].

0. This *PoC* was tested under a version of *Windows 10* 32-bit and a 32-bit 3.4.4 of *Wireshark*.

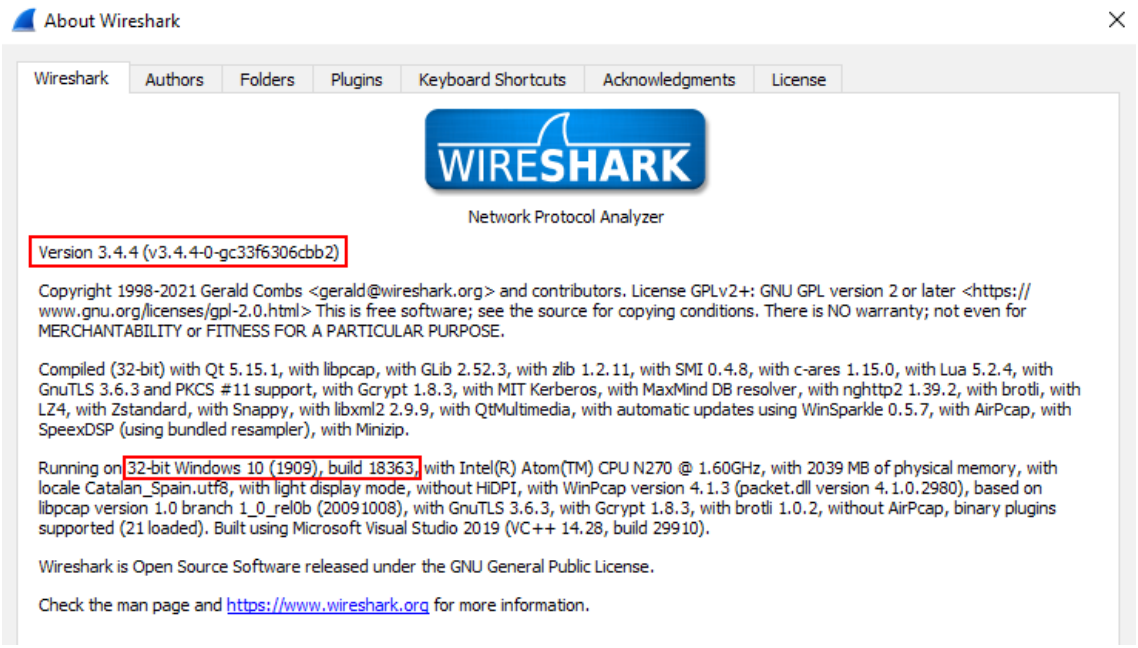

Figure 29: *Wireshark* – About screen showing their version + *Windows*

For bridge data capture to work, you need to use *WinPcap* instead of *Npcap* [15] to let physical ports be captured (forwarding/transit traffic).

Step 1: Installing the old and unsupported *WinPcap*

1. The official site still exists, you can download it and install on your *Windows 10*. Follow the easy *next-next* wizard.

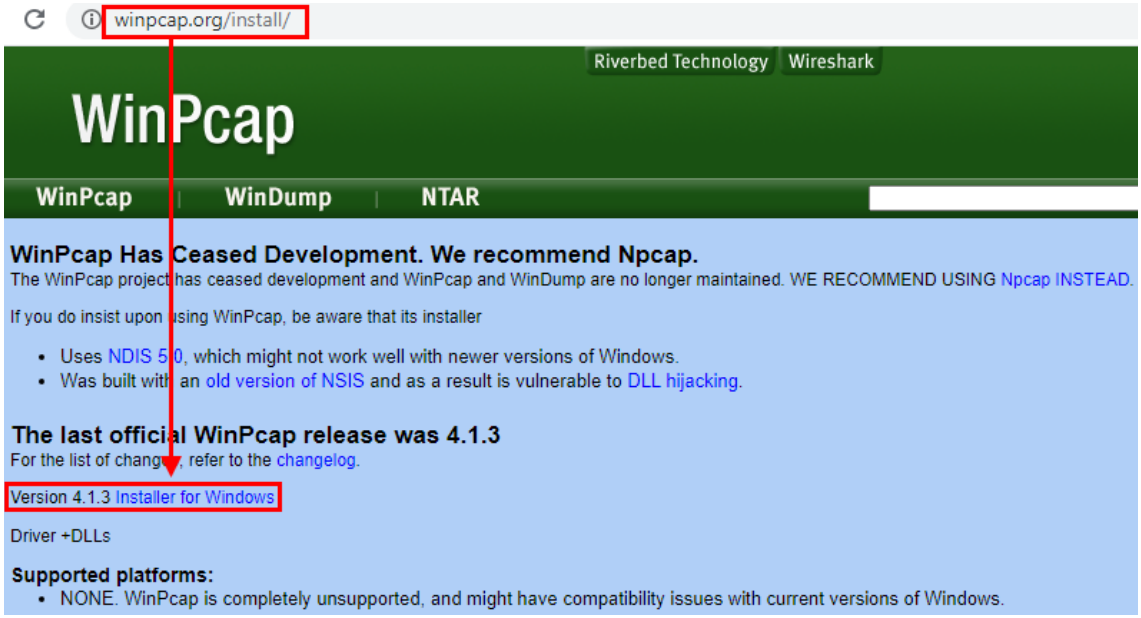

Figure 30: Downloading *WinPcap*

Step 2: Installing *Wireshark* without *Npcap*

1. In Wizard, be sure the checkbox *Install Npcap…* is unchecked and continue the installation as normal.

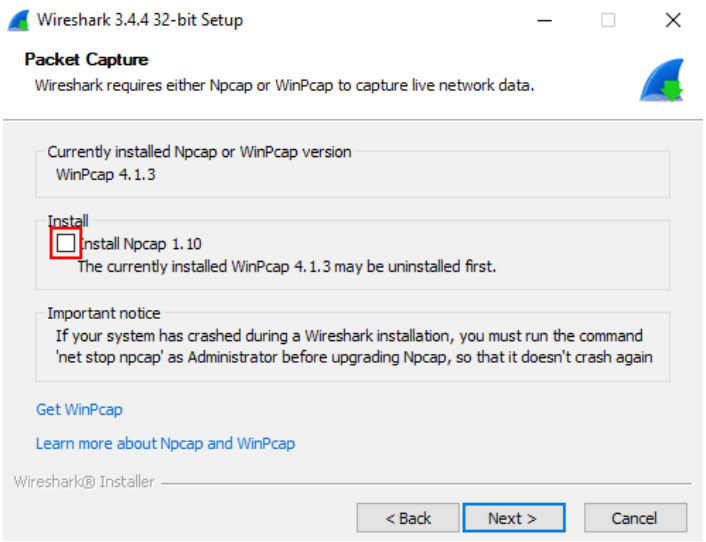

Figure 31: *Wireshark* – Install process maintaining *WinPcap*

Step 3: Prepare the *HotSpot* on Ethernet side

1. The role of the *Gargoyle* router is to behave like a gateway, the default config with the enabled AP.

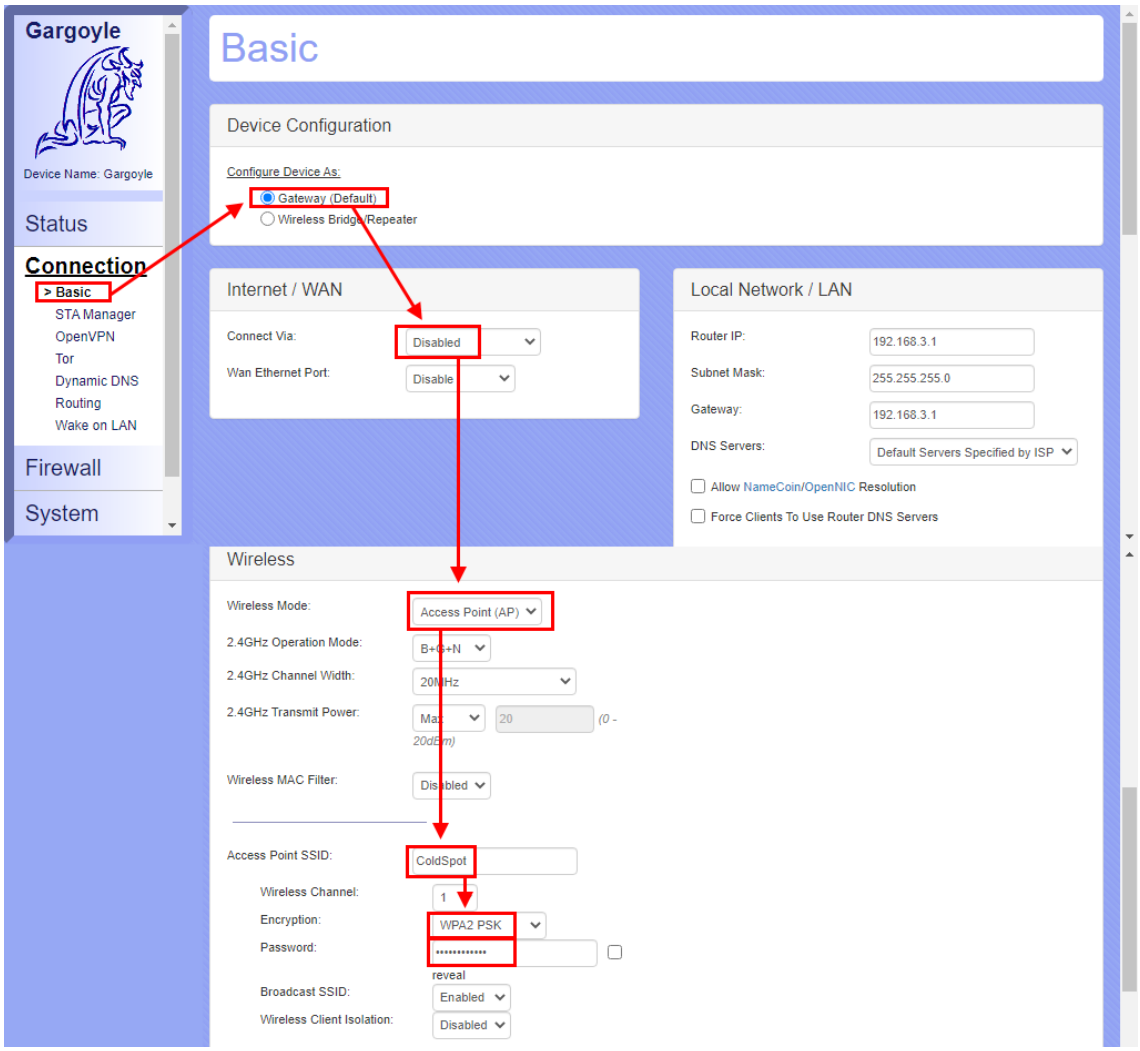

Figure 32: *Gargoyle* – Configure as gateway & set AP

2. And the DHCP is OFF, so IP address should be taken from *Meross* smart device.

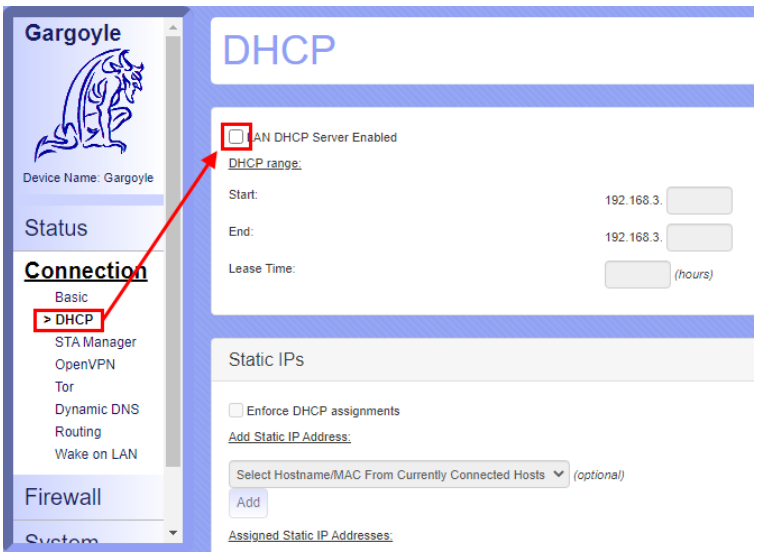

Figure 33: *Gargoyle* – Turn OFF DHCP

Step 4: Setting the Wi-Fi client on wireless interface

1. Simply, connect to the OPEN *MSS550X* Wi-Fi network. (there is no password, and is easy to identify, check the icon around available wireless!).

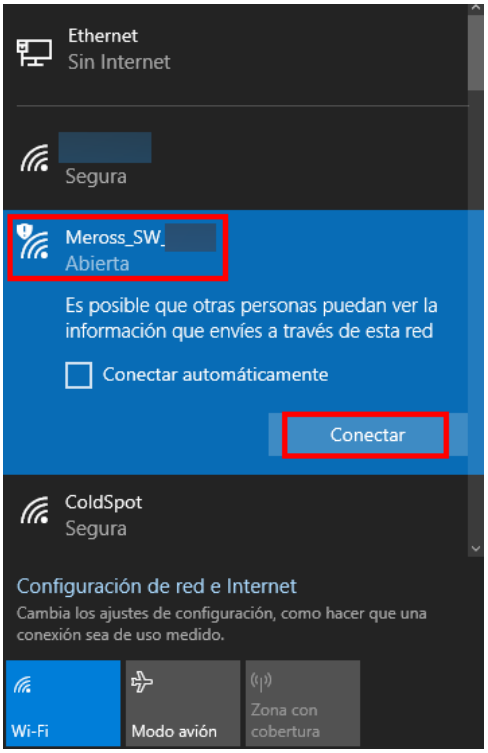

Figure 34: *Windows 10* – Wi-Fi network management

Windows here does a good job alerting with a warning sign in the Wi-Fi network chosen and with some description (here is in Spanish) saying: *"Other people might be able to see info you send over this network"…*

Step 5: Create the network *Windows* Bridge

1. In *Network connections*, select the 2 interfaces (Wi-Fi & Ethernet) – *right click over the Wi-Fi Network – Bridge Connections*.

Note: Very important right clicking over Wi-Fi (not over the Ethernet one), this will be the NIC node that will lead you with the Meross network, this will determine the success of the bridge creation.

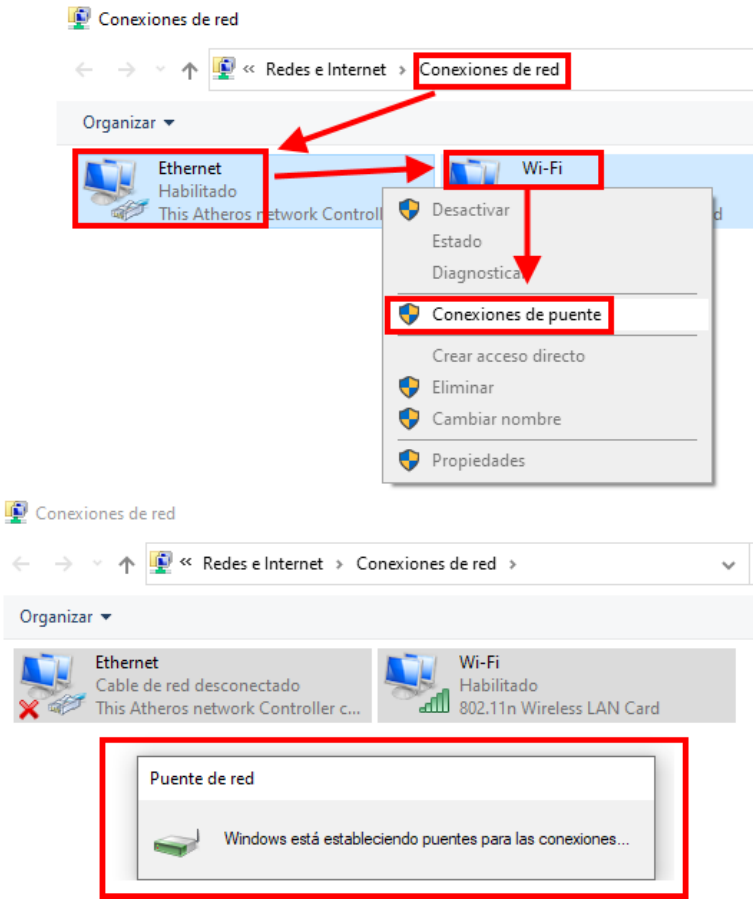

Figure 35: *Windows 10* – Bridge creation with 2 interfaces

As troubleshooting, sometimes the bridge network can fail in their creation, an error message appears (here is in Spanish) saying: *"An unexpected error occurred while configuring the Network Bridge"*. In order to solve it, try to choose the NIC that couldn't be joined to the bridge (the Ethernet). Before doing that, wait until the bridge is showing the SSID (means the direction of the connection is going well). Right click and check if the window option says *"Remove from Bridge"* or *"Add to Bridge"*, the last is telling network adapter is not in bridge so choosing *"Add to Bridge"* will do the trick. (in Spanish is "*Agregar al puente"*) any error shouldn't be appear.

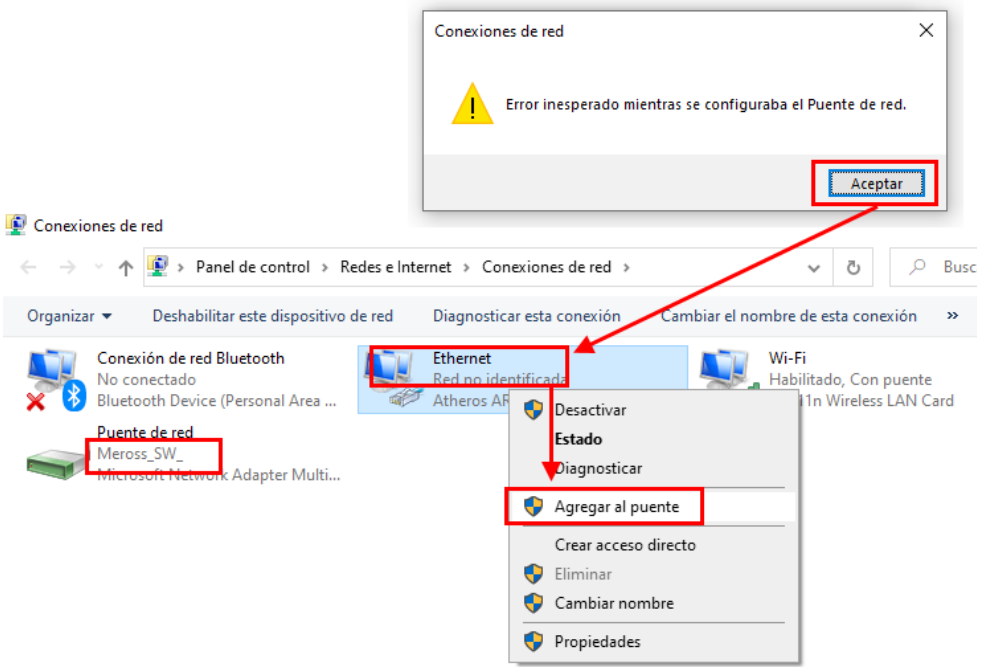

Figure 36: *Windows 10* – Troubleshooting when Bridge fails

If all goes well, you will see the Bridge interface with the SSID name of *Meross* (*Meross_SW_XXXX*).

Step 6: Ready to run *Wireshark*

1. Run *Wireshark* as usual and choose one of the physical interfaces, in this example Ethernet was selected (do a double click on interface and capture will start).

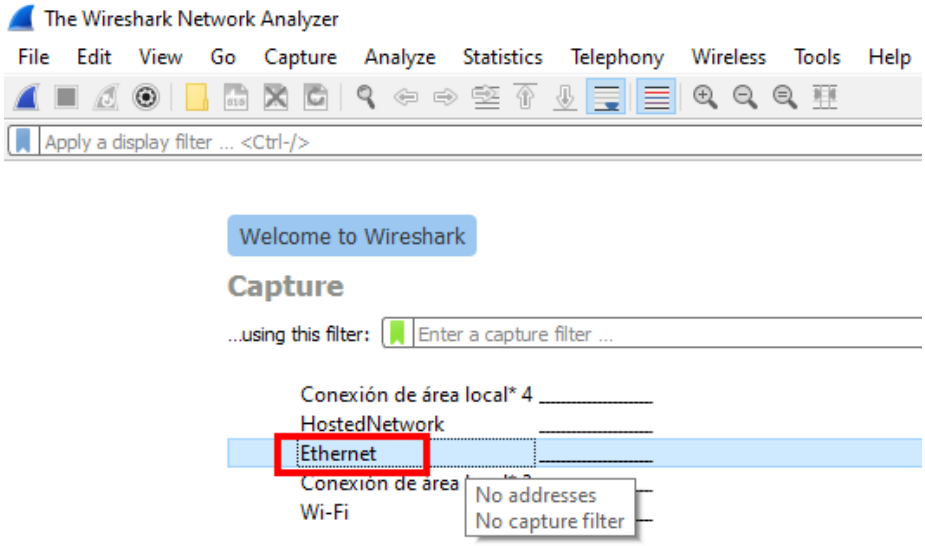

Figure 37: *Wireshark* – Start a capture on physical interface

2. Connect to the SSID network that will enter to the bridge scenario instead of connect to the real Meross Wi-Fi (*Meross_SW_XXXX*). We named it *ColdSpot*.

3. Smartphone gets an IP address from DHCP as usual but… maybe IP is the same as the bridge, why? This behavior is depend of how DHCP server is managing the leases. Maybe the bridge and/or *Meross* DHCP device service is getting same MAC coming from the bridge. It receives certain requests and chooses by default those of a specific correlation and possibly believes that it always comes from the same MAC address… Don’t worry, we have a solution for this. Set a static IP, look at the IP config of the bridge, put the following IP address on your smartphone (next available, if 2 put the end as 3, you can put other until 254, but let's be true to the queue) and packets will flow fluently.

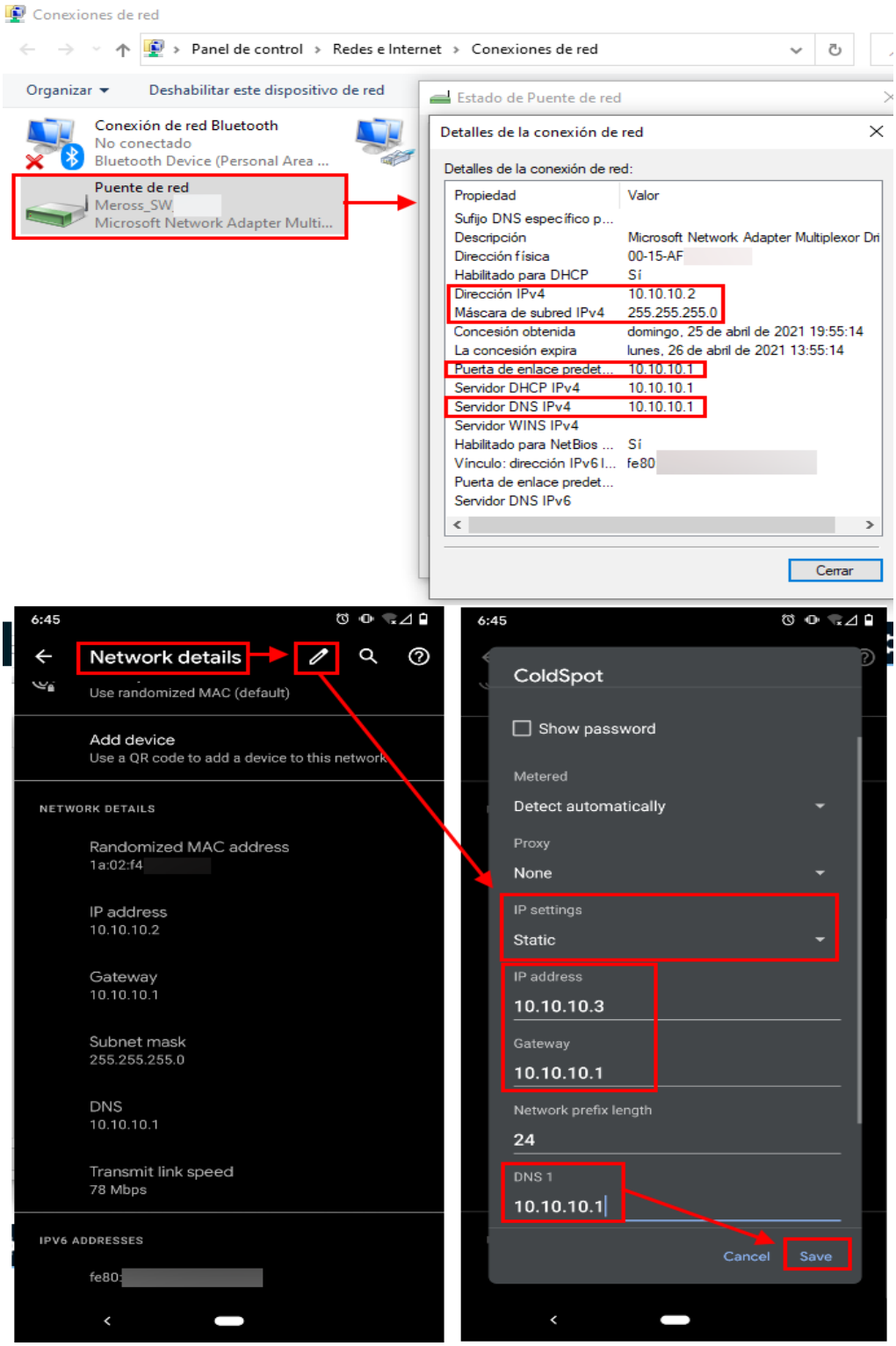

Figure 38: *Wireshark & Android* – Top; IP bridge info. Bottom; Set static IP

4. You know what to do here, but no problem to remind you! Remove device seen in figure 9 of chapter 3.1.1. and repeat steps from figure 6 of the chapter 2.1.1. and so on, same type of data will be captured (*HTTP/JSON*).

4. Cybercriminal is in the Air

Wi-Fi networks are in the air and they can take very wide directions. If you can read in many places like in your own OS that open Wi-Fi are not so good because traffic on them can be spied (like seen in *Windows 10* in figure 34) is for some good reason…

We want to show you a realistic tested situation about a hacker that made a bridge for testing their device (*Kali Linux* case, chapter 3.1.1.) that feeds at same time to a cybercriminal who is listening and filtering for a *Meross_SW_* SSID for malicious purposes.

Let's explain the easy steps for the malicious person.

0. A few requirements: A version of *Kali*. A wireless interface (if is laptop, can be the embedded one or use an external to be more *"able to hear"*).

Step 1: Putting the wireless interface into monitoring mode

1. Wi-Fi NIC needs entering into monitor mode to be able to capture data but before it, some wireless client process/es must be killed to avoid disruption. The *airmon-ng* also can help in this task.

2. Then, for starting with the default interface (if there is only one) can be started with same tool.

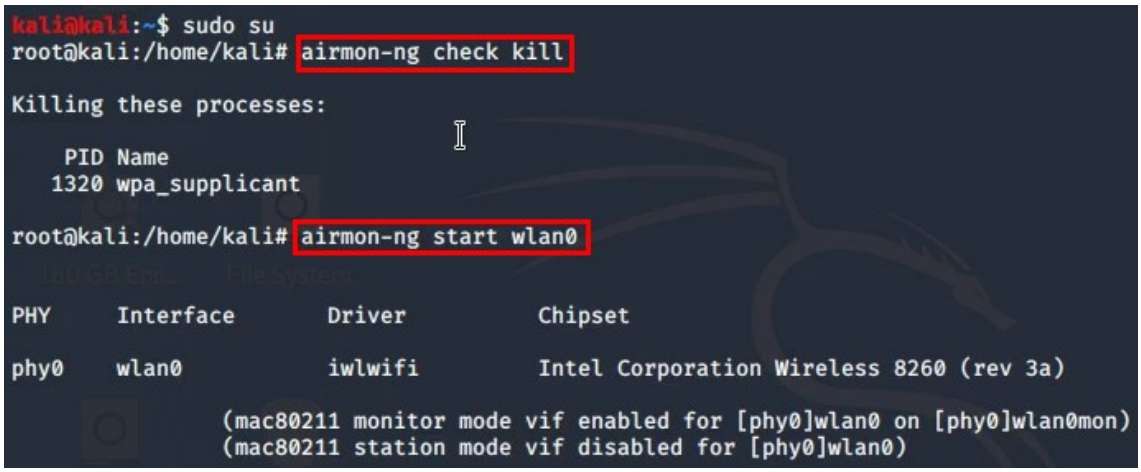

Figure 39: *Kali* – Enabling monitor mode with *airmon-ng*

The new interface called *wlan0mon* will be created and, it will be used for the following command/tool.

Step 2: Filtering the SSID by *Meross_SW_*

1. Is well known, these devices has their SSID starting by the same prefix and seems operates on channel 1. *Airodump-ng* tool can do this type of filtering.

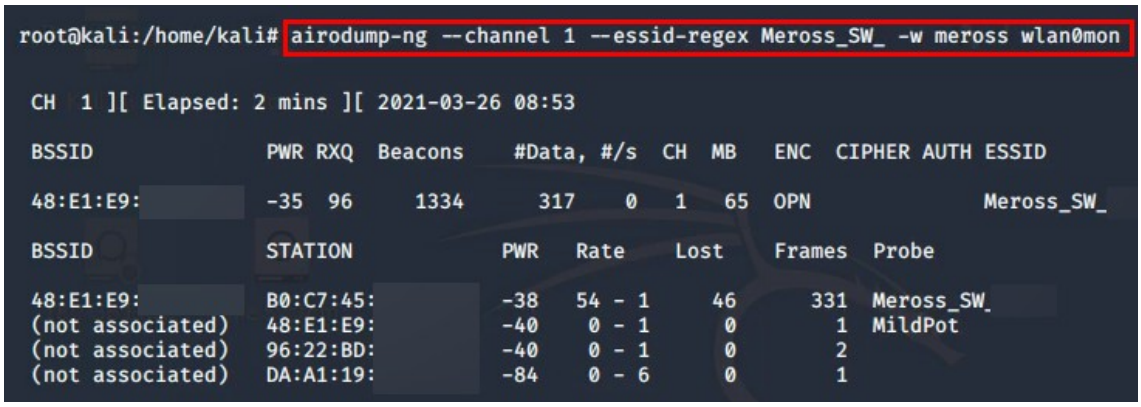

Figure 40: *Kali* – Filtering by SSID with *airodump-ng*

The option for filter the SSID is: *--essid-regex* and capture is being recorded to the file called *meross-01.cap* (*-w* is for tell to save current capture to a file and, it add prefix *-01* if there is only one file named *meross*). Note that, the client with MAC starting with *B0:C7:45* (the *Gargoyle*) is who's generating data.

Step 3: Let's see the collected data

1. Simply invoke *Wireshark* from terminal pointing the *meross* file.

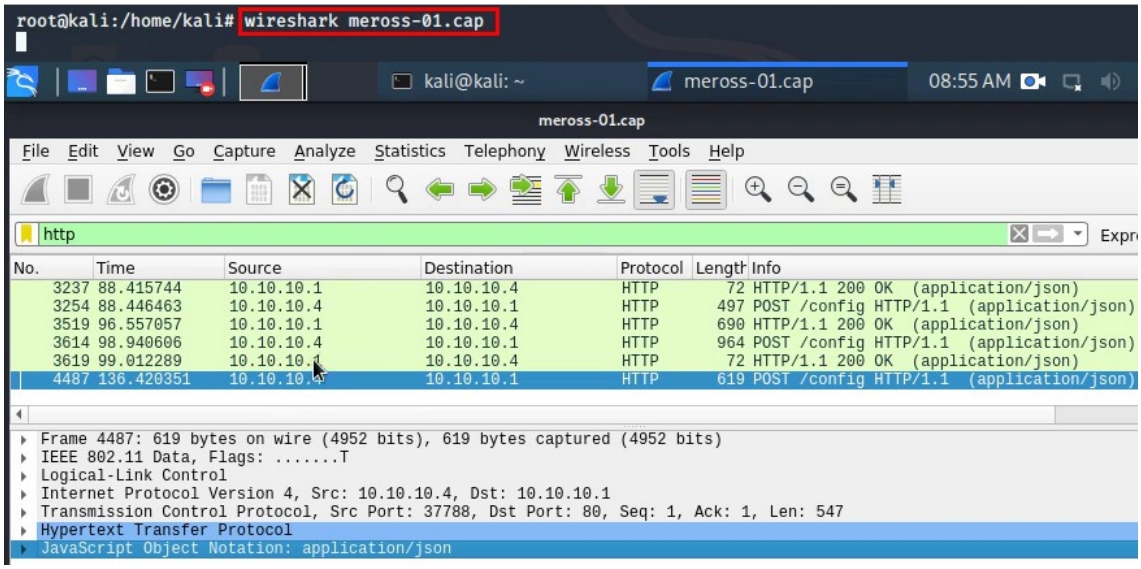

Figure 41: *Wireshark* – Inspecting *802.11* air data

This is a copy of the data observed in *Wireshark* of the figure 26 of chapter 3.1.1. but with some little differences in *OSI* layer, like the *IEEE 802.11* for example.

Someone you don't trust can take your Wi-Fi credentials!

Looking other example to understand it? See this video:

https://www.youtube.com/watch?v=pMzULxYDsNM

5. Conclusions

We have done a real simulation involving two roles:

- Role as pentester: The hacker is usually a curious person at IT level, always wanting to know how a newly acquired device works. The task employed was information gathering, hearing the device conversation by putting itself in the middle (seen from chapter 3.1. to 3.1.1.).

- Role as cybercriminal: The malicious person is always looking for take profit at the expense of others, trying hacking tactics (seen in chapter 4.).

In this short but deep story, the cybercriminal, using their social engineering, knows that their neighbor (the hacker) has a smart device which knows how it works because saw an interesting article on Internet [16]. That article not points to the same device like hacker has, but thinks that if it's from the same manufacturer it might work the same way, so the impatient person, suddenly becomes patient and turn on sniffer on their super Wi-Fi antenna in order to catch something in the air.

Is this enough to understand the danger that can create the open wireless weapon? You may try at night, at very late hours to perform the pairing to complete the set up if you think is only solution to avoid the problem. Cybercriminal could sleep but their weapon not! Time to think the possible solutions, because not only a user can suffer a security breach with normal use (without using *Machine-in-the-Middle*) even hacker was *pwned* in their own testings!

Let’s study what *Meross* app saw under both *Machine-in-the-Middle* scenario (*Kali Linux & Windows respectively*).

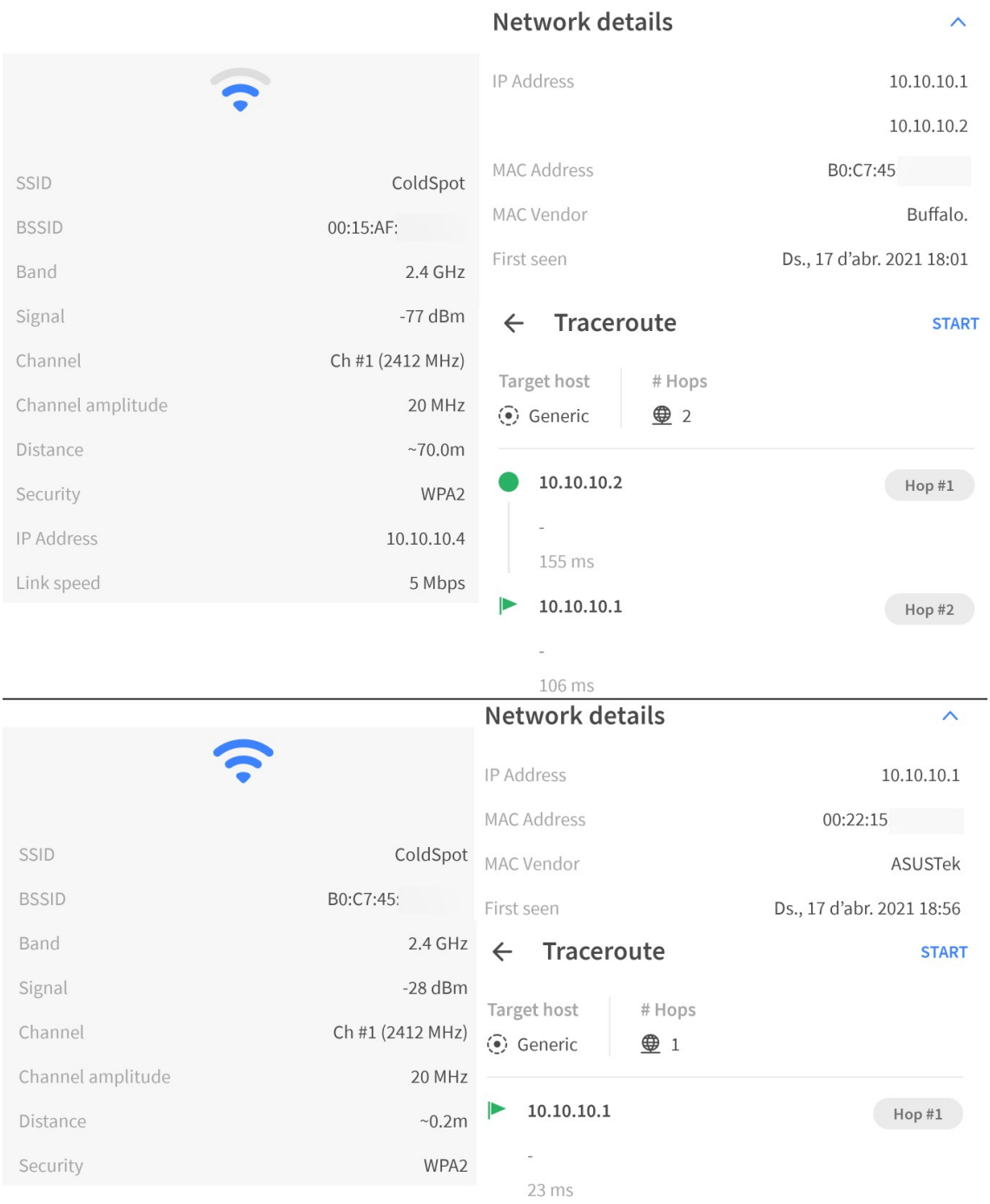

Figure 42: *Fing* – Network overview seen from *MSS550X*. At top; scenario of *Kali*. At bottom; scenario of *Windows 10*

Using *Fing* [17] (available as mobile app), we inspection MAC addresses of the SSID, gateway (*smart wall switch*) and a *traceroute* to the device IP to see the jumps until the *MSS550X*.

In both cases, the MAC of the SSID (BSSID) and of the smart device is not the one that belongs to it, and do not forget, that in both scenario, app accept and recognize the *MSS550X* completing their set up successfully. Neither the name of the SSID is equal at the beginning characters, but app accept it. What seems to be equal? At level/layer 3, the IP address of the end node/device, that is *10.10.10.1*.

The big faults under my point of view are:

- The non verification of some processes at network level.

and

- The use of non encrypted transmit channel.

If app would do a correct verify, it would not allow be sniffing by *Machine-in-the-Middle* method. But this won't solve the problem of revealing the clear data through the air…

Let's put in a list some security measures ordered from more important to less:

1. Use of encryption on Wi-Fi *HotSpot* created in smart device: Doing that, the malicious outsiders will see the air data in invisible mode.

2. Encryption of the content of user/SSID & password: If Wi-Fi fails encrypting all their traffic, at least, do the most confidential part of the content like the password of the SSID owner,… Maybe using *HTTP**S*** instead of simple and plain *HTTP* will be a good idea, or use both. Here is only a protection by *Base64* algorithm.

3. Check the MAC address by OUI: App may do a MAC check by their first 3 hexadecimal pairs and, if it's not related to *Meross*, thrown an error and stop communication. If this were so, these *PoC's*, both *Windows* & *Kali* would fail at sniffing.

4. SSID checking, channel Wi-Fi checking and type of security: The verification of the first words of the SSID, in conjunction with the channel data is running and type of security may be a good point to robustness it.

5. Allow only 1 IP lease on DHCP pool: The DHCP service of the smart wall switch device, may can be accept 1 IP address to give to a device. If more than 1 exists, drop connection for this secondary attempt. In both scenario, will fail because 2 IP are needed to flow data into the bridge.

6. Don't allow more than 1 hop on a *traceroute* check: App can do a verification doing a *traceroute*, if there is more than 1 hop, drop communication. As we see in figure 42, will get success under *Windows 10*.

The only verification app does is the gateway IP, the IP of *MSS550X*. We believe that if it met the first point, it would have enough protection to not to be sniffed in the air. But not forget, if only first point is maintained, device is sensible to *Machine-in-the-Middle* attacks, for a very good security, covering all points could be more than great (No sniff, No *MitM* attacks).

We tried to write this white paper in clear mode to understand the problematic this situation is and the danger could be producing. In nowadays there is some automatism

in lot o tasks, other are using as good purposes, but some of them, can be used to slowly hurt you without you noticing. Sometimes it is too late, sometimes it leaves consequences, take care of yourself and of the health of your IoT's, and educate them like a child.

Remember…

Be Good, Be Hackers.

6. References

[1] Roaming: Using a mobile phone in the EU - Your Europe. https://europa.eu/youreurope/citizens/consumers/internet-telecoms/mobile-roaming-costs/index_en.htm

[2] Meross: Simple Device, Simplify Your Life. https://www.meross.com/Detail/67/Smart%20Wi-Fi%202%20Way%20Wall%20Switch

[3] IEC 60446 - Wikipedia. https://en.wikipedia.org/wiki/IEC_60446

[4] meross - Apps on Google Play. https://play.google.com/store/apps/details?id=com.meross.meross&hl=en&gl=US

[5] Meross on the App Store. https://apps.apple.com/app/meross/id1260842951

[6] Configuring an Android Device to Work With Burp - PortSwigger. https://portswigger.net/support/configuring-an-android-device-to-work-with-burp

[7] CaptureSetup/Ethernet - The Wireshark Wiki. https://wiki.wireshark.org/CaptureSetup/Ethernet#Capture_using_a_machine-in-the-middle

[8] linux - Bridging wlan0 to eth0 - Server Fault. https://serverfault.com/questions/152363/bridging-wlan0-to-eth0

[9] Gargoyle Router Management Utility. https://www.gargoyle-router.com

[10] hostapd: IEEE 802.11 AP, IEEE 802.1X/WPA/WPA2/EAP/RADIUS Authenticator. http://w1.fi/hostapd

[11] bridge_repeater [Gargoyle Wiki]. https://www.gargoyle-router.com/wiki/doku.php?id=bridge_repeater

[12] How do I prevent Network Manager from controlling an interface? | CDRouter Suppor. https://support.qacafe.com/knowledge-base/how-do-i-prevent-network-manager-from-controlling-an-interface

[13] networking:iproute2 [Wiki]. https://wiki.linuxfoundation.org/networking/iproute2

[14] About the Wireless Hosted Network - Win32 apps | Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/nativewifi/about-the-wireless-hosted-network

[15] Wireshark Q&A - Capturing problem Man-in-the-middle ethernet bridge windows 10. https://osqa-ask.wireshark.org/questions/62555/capturing-problem-man-in-the-middle-ethernet-bridge-windows-10

[16] Device pairing · albertogeniola/MerossIot Wiki · GitHub. https://github.com/albertogeniola/MerossIot/wiki/Device-pairing

[17] Fing - IoT device intelligence for the connected world | Fing. https://www.fing.com