Wireless security

Laboratory for the class "Cybersecurity" (01UDR)
Politecnico di Torino – AA 2021/22
Prof. Antonio Lioy

prepared by:
Diana Berbecaru (diana.berbecaru@polito.it)
Andrea Atzeni (andrea.atzeni@polito.it)

v. 1.2 (04/11/2021)

Contents

1 Purpose of this laboratory

In this laboratory, you will perform exercises aimed to experiment with wireless networks tools to understand in more detail their working principles.

1.1 Additional software tools

The tools listed below will be used as well throughout this laboratory:

Wireshark - is an open-source tool (having a user-friendly graphical interface) which allows capturing network traffic.

Home page = https://www.wireshark.org/

1.2 Additional (useful) commands

To experiment with a wireless network, the commands to manipulate the wireless network card can be useful

iwconfig - is a command-line program to get and set wireless network card parameters (like, for example, the network SSID). Its purpose is somehow similar to the `ifconfig` command for the wireless case.

> NOTE
>
> This command has been very relevant in the past, but it may not fully interoperate with modern Linux wireless network driver (e.g. nl80211 family). So it is shown here for completeness (and in many tutorials you still can find it) but you should prefer the next one.

IW

`iw` is a replacement of `iwconfig` for modern drivers. it provides the same functionalities of `iwconfig`, but with different command syntax and improved compatibility with modern hardware.

`iw [option] command`

where *option* can be, among the others

`--version` print version information and exit

`--debug` to enable netlink message debugging.

and most relevant *commands* for this laboratory can be:

`help [command]` to print usage for all or a specific command, e.g. ”help wowlan” or ”help wowlan enable”

`dev` *devname* `info` to show information for *devname* interface

`dev` *devname* `link` to show information for wireless connections associated to *devname*

`dev` *devname* `connect` *ESSID* to connect *devname* interface to the *ESSID* wireless LAN

`dev` *devname* `disconnect` to disconnect *devname* from an AP.

Aircrack-ng suite

The `Aircrack-ng suite` is a set of open-source command-line tools to monitor, attack, test and crack 802.11 WEP and WPA networks. in the following we will mention the most useful for this laboratory (but feel free to explore more of them and more in depth the ones presented). Home page = https://www.aircrack-ng.org/.

`airmon-ng [cmd] interface [channel]`

This script enables and disables the monitor mode on wireless interfaces. Without parameters, it will show the available interfaces. Another brutal task often accomplished is to kill programs that can interfere with the wireless card operation in background (and list them beforehand).
cmd may be one of `start`, `stop` and `check`. `start` enables the monitor mode on the wireless *interface* (on the wireless *channel* if specified), `stop` disables the monitor mode on the *interface*, while `check` is involved in the list and kill of other process that can modify the wireless card in background and interfere with the behaviour of `airmon`.

NOTE

The wireless interface used in `start` and `stop` is tipically different, since `airmon` creates a new interface with *mon* at the end to use for the monitor mode, and disables the old one. so `start` uses the original, while `stop` uses the new one (with 'mon' suffix)

`airodump-ng [options] interface [,interface,...]`

`airodump-ng` is used for sniffing of raw 802.11 frames. Typical use of the sniffed result is input for `aircrack-ng`.
options relevant for this laboratory are:

- `-w` *prefix* indicates the prefix of the dump files containing the sniffed result. `Airodump` captures the packet and also creates a set of files suitable for further operations. All these files share the same *prefix*.
- `--help` displays the usage and available options.

`aircrack-ng [options] file(s)`

is a command-line tool to crack WEP/WPA-PSK key
options relevant for this laboratory are

- `-w` *dictionary* specifies a path to a dictionary file for wpa cracking. Can be specified "-" to use stdin. A list of relevant wordlists can be found at http://www.aircrack-ng.org/doku.php?id=faq#where_can_i_find_good_wordlists
- `--help` displays the usage and available options

while the *file(s)* are files with the captured packets that aircrack-ng can work on offline

Mininet-wifi

`Mininet` is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet networks run real code, like Linux network applications as well as the real Linux kernel and network stack. Mininet is a powerful way to experiment with networks, but the limitation of the original version is that it does not support Wireless networks.
Mininet-WiFi is a fork of the Mininet network emulator, extended by adding virtualized WiFi Stations and Access Points based on the standard Linux wireless drivers. New components have been added to support the addition of these wireless devices in a Mininet network scenario and to emulate the attributes of a mobile station such as position and movement relative to the access points, as well as security functionalities.
To activate `mininet-wifi` and create a minimal topology you can run the following command:

`sudo mn --wifi [command_opts]`

With no *command_opts* it creates a virtual topology with one AP and two wireless stations connected. Simply by varying the command line option, different topologies can be easily instantiated.
Some *command_opts* relevant for this laboratory are:

- `-h, --help` show the help message with the list of possible options
- `--topo=`*TOPO[,param]* where *TOPO* can assume many possible value according to possible topology (`single` is for a topology with a single access point and all stations connected to it, and the optional *param* can be the number of desired stations)

- `-v` *VERBOSITY*, `--verbosity=`*VERBOSITY* where *VERBOSITY* can be one of `debug`, `info`, `output`, `warning`, `error`, `critical` (most detailed `debug`)
- `-x` spawn xterm command shells for each node
- `-c` clean possible interrupted and dirty instations of Mininet-wifi. Since it is an experimental project, it can become unstable and need some clean-up from time to time (if you experience some "strange" behaviour, try this command)

2 Mininet-Wifi

2.1 Mininet WiFi set-up and basic operations

For this laboratory, you will have to simulate the presence of a simple wireless network through `Mininet-Wifi`. First step is to download a lightweight VM specifically shaped for running `Mininet-Wifi`. You can download from the original site at

- https://mininet-wifi.github.io/get-started/

or from the internal Torsec storage server (which is the preferred option at Labinf) at

- https://storage-sec.polito.it/external/kali/2021/mn-wifi-vm.ova

Then you can import it in a Virtualbox, repeating the same steps explained for importing a Kali VM (refer to *lab00* for details) and finally you can start the VM in Virtual Box where you should see the initial `Mininet-Wifi` screen, as shown in Fig. 1 (note that the background image can change according to the version, but the instructions remain valid)

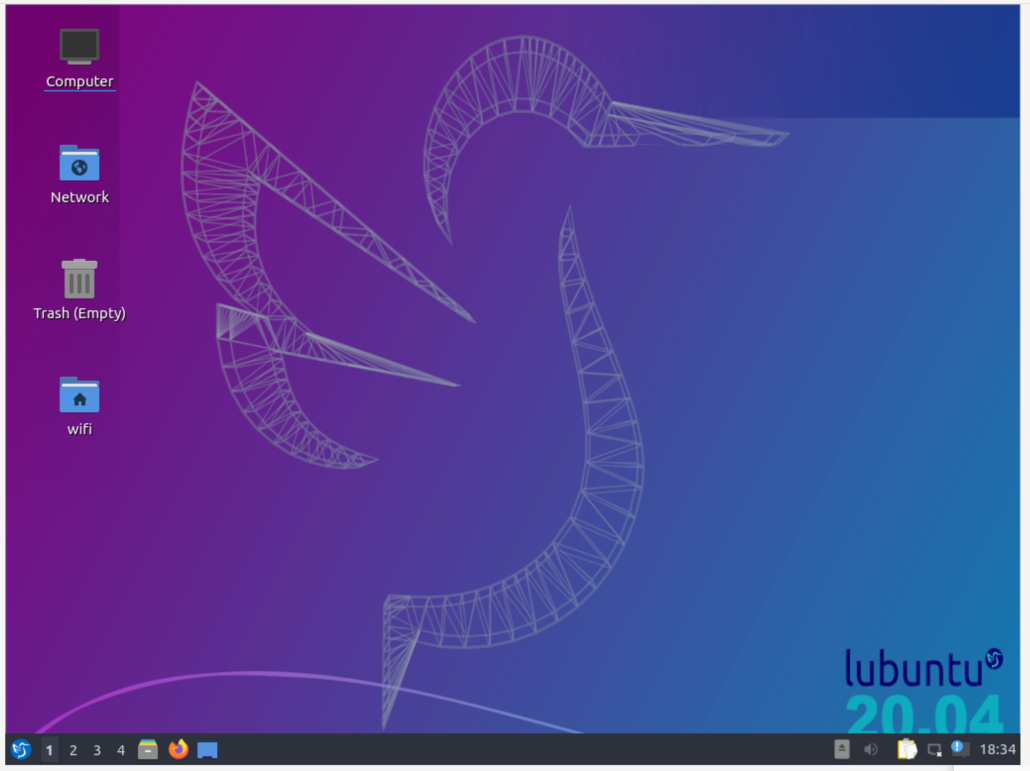

Figure 1: Initial Mininet-Wifi screen.

Now, you can open a command shell (we suggest to open "QTerminal" as shown in Fig. 2, since it is one of the most user-friendly available in this environment) and be ready to create your virtual but realistic wireless network.

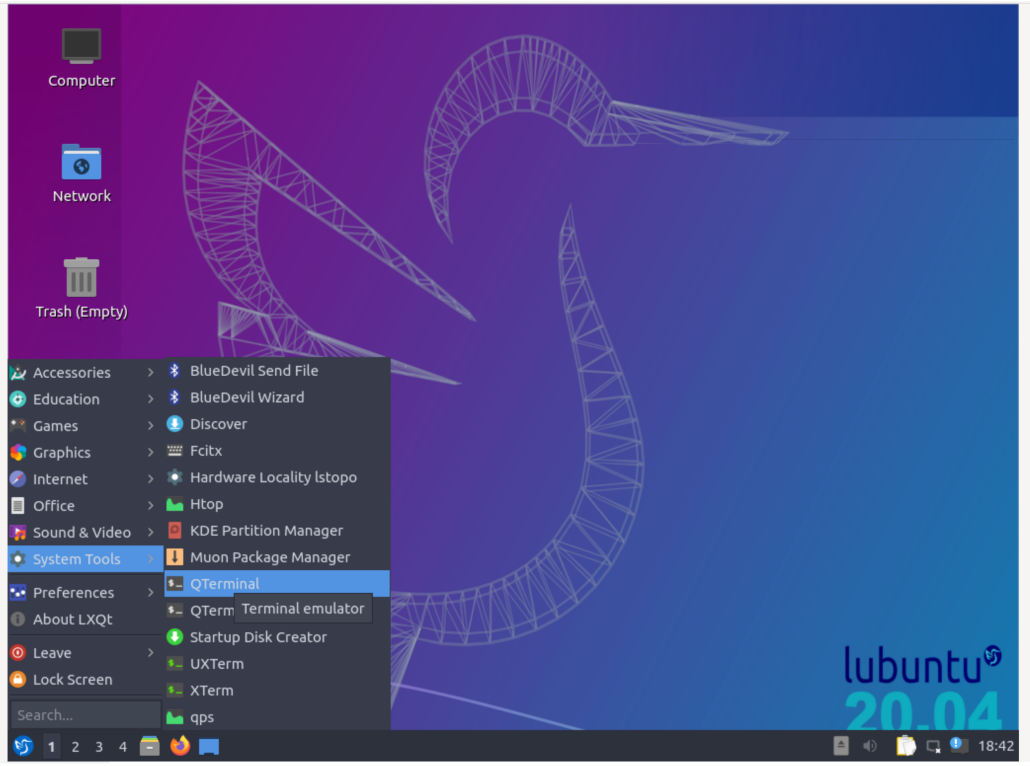

Figure 2: Opening a QTerminal in Mininet-Wifi screen.

2.2 Experiment with a simple wireless topology

In the opened terminal, to create a WLAN, run the following command

```
sudo -E mn --wifi --topo=single,3 -x
```

When asked for the password, insert the "wifi".

According to the result of command, which set-up a wireless network, what kind of topology do you guess Mininet-Wifi created?

→

Any of the five created terminals can act as the control shell of a different virtual node. Using the usual terminal `ip` command to query the network parameters, can you identify the different wireless interfaces and the respective addresses of the nodes?

→

In any terminal, you can run the following command to see the characteristics of the wifi interface:

```
iw iwface info
```

where *iwface* stands for the wifi interface in the specific terminal. For example, in the terminal for `sta1` you would run:

```
iw sta1-wlan0 info
```

or if you want to concentrate on the link characteristics

```
iw iwface link
```

You can note the interface type (set to "managed" in the stations and to "master" (AP) in the Access Point terminal), as well as the SSID of the network ("my-ssid").

Now, initiate the transfer of some packets between two stations, e.g. you can run `ping` commands from `sta1` to `sta2`, using the terminal of `sta1` with the command:

```
ping IPaddress_sta2
```

How you could identify and analyse the packets exchanged between the two stations? What is the data length of a transmitted packet?

→

What kind of protocol has been used at layer 2?

→

Since you have three nodes, try also to run `wireshark` on `sta3` when pinging `sta1` and `sta2`. Can you see the packet exchange?

→

2.3 Sniffing on a Wireless network

In the previous point, you should have noticed that many details of a Wireless communication are hidden in a "normal" sniffing on a wireless card. If you want to access actual details of a 802.11 exchange, you have to put your interface in monitor mode.

For example, you can transform `sta3` in a network sensor. To do that, in `sta3` command shell create a wifi interface in "monitor mode" exploiting the functionalities of `airmon` tool: run the command

```
airmon-ng start sta3-wlan0
```

Press "y" when asked to resolve some issues. In the output, you should see a message stating *"(...monitor mode vif enabled for ...sta3-wlan0mon)"*.

If you check the station 3 network interfaces with the command `ip addr show`, you should see that it has been created a new interface in monitor mode, called *"sta3-wlan0mon"* (and the older one has been disabled). In brief, airmon creates an interface with the original wifi interface name (e.g. *sta3-wlan*) followed by the suffix "mon" (for monitor mode). You should be able to see the output similar to the one shown in the Fig. 3

Next, in `sta3` restart `wireshark`, and select the *sta3-wlan0mon* interface and start capturing packets (Fig. 4).

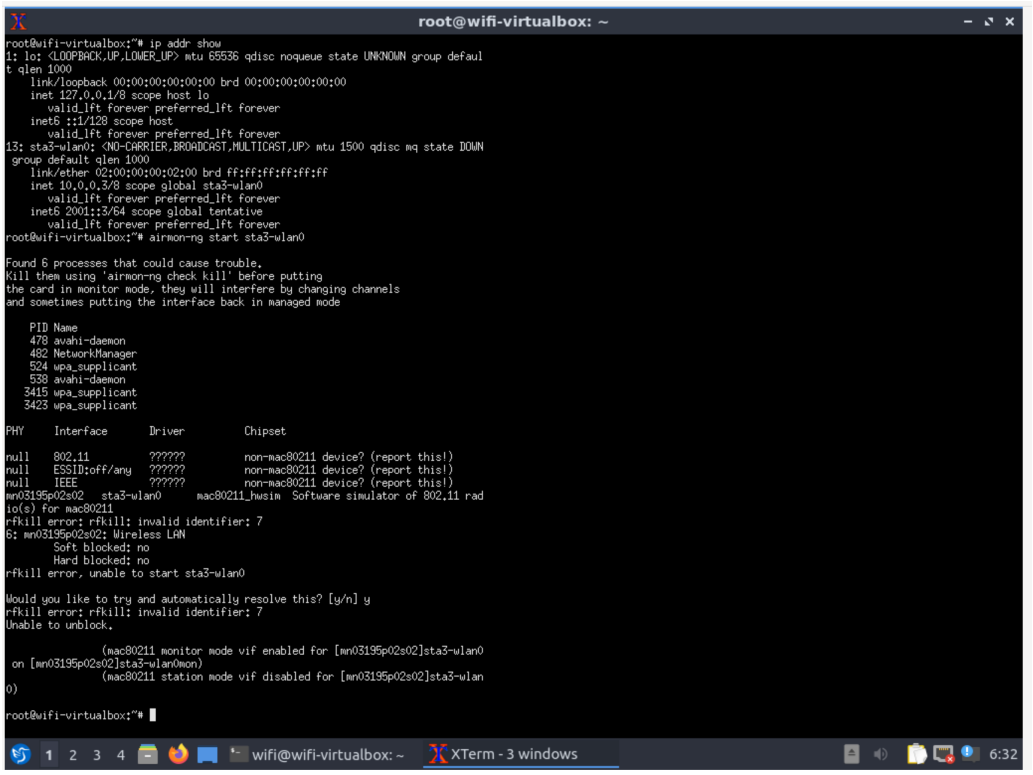

Figure 3: Setting the wifi interface in monitor mode.

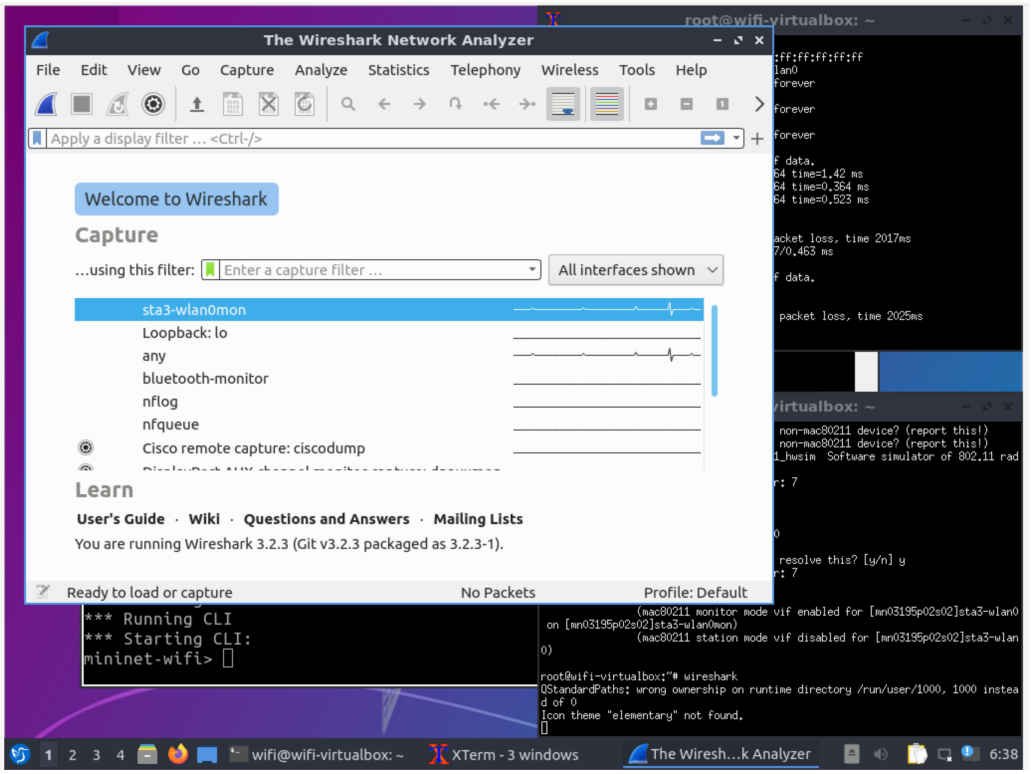

Figure 4: Sniffing 802.11 traffic with wireshark.

In this way, you will be able to see 802.11 “real” packets and if you try to ping `sta2` from `sta1` you should see the captured packets in the `wireshark` window of `sta3`.

Once you have finished, go back to the mininet_cli console and run the command:

```
exit
```

as shown in Fig. 5

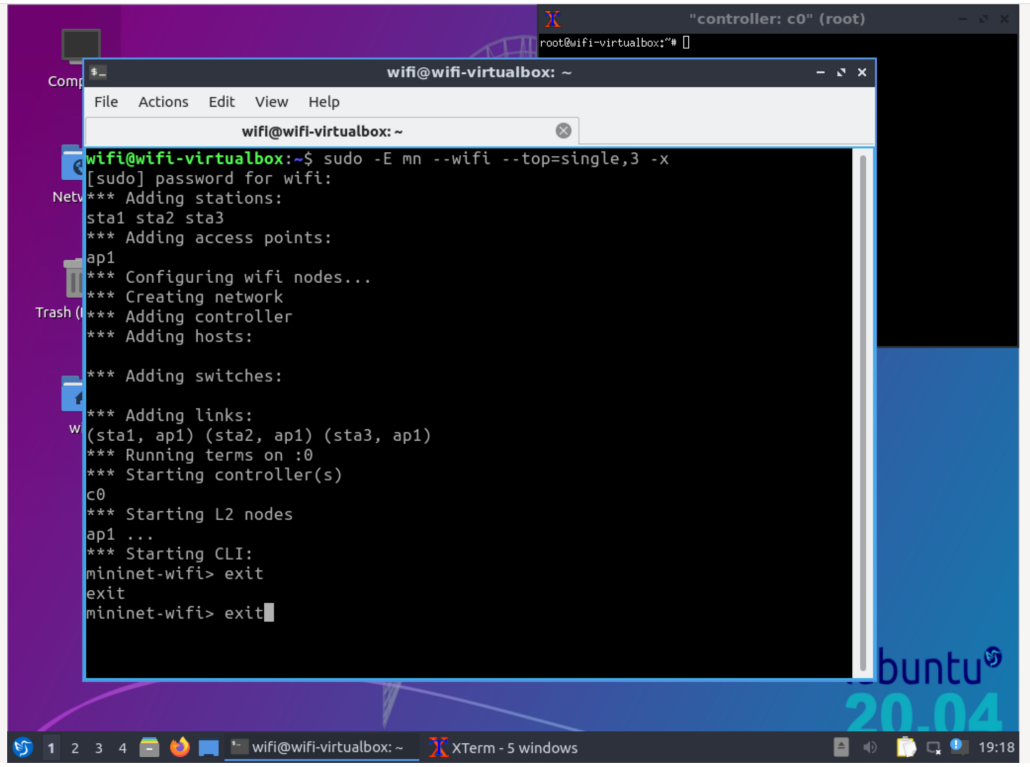

Figure 5: Closing the initial Mininet-WiFi topology.

This will close all the terminals corresponding to the five nodes and go back the initial setting.

2.4 WEP connections

Now it is time to create a WEP connection and examine it. To do so, you can take advantage of the support material, where you can find a python script that can create a suitable network topology. Extract `authentication_3_wep.py` from `CYB_lab03_support.zip` and execute the script with the following command:

```
sudo -E python authentication_3_wep.py
```

That will result in the opening of four terminal windows

In this case, can you derive the resulting network topology?

→

Launch the command "`ip addr show`" in each terminal, and note the corresponding *staX-wlan0* interfaces.

For simplicity, we assume to use `sta3` to sniff the 802.11 traffic, in particular the traffic exchanged between `sta1` and `sta2`. Thus, go into the terminal corresponding to `sta3`, and set the wifi interface in monitor mode and then start `wireshark` to be able to see 802.11 traffic sniffed in real time.

We can setup a WEP "secure" connection thanks to `iw`. You can setup the shared key to be used in a WEP connection (same for AP and different stations) exploiting the following commands in the respective stations:

```
iw sta1-wlan0 connect simplewifi key d:0:abcdeabcde
```

```
iw sta2-wlan0 connect simplewifi key d:0:abcdeabcde
```

Then ping from `sta1` to `sta2`:

```
ping IPaddress_sta2
```

Stop the capture, if you want, on `sta3` and analyse the sniffed traffic.

What kind of differences can you notice in respect of the sniff you performed without WEP?

→

Can you identify the Initialization Vectors inside the communication? Are you able to notice any correlations between different IVs?

→

When finished, type `exit` in the Mininet Command-Line interface to shut-down the topology.

2.5 WEP attack

2.5.1 IV collision problem

In this excercise we will exploit the short Initialization Vector to derive the key in a WEP exchange.

Re-establish an appropriate wireless topology by exploiting `authentication_3_wep.py` and `iw` as in the previous section Again, let's use `sta3` as monitor (activate it with `airmon` if needed), then launch `airodump` with the following command:

```
airodump-ng --bssid AP_MAC -w capturefile sta3-wlan0mon
```

This starts a selective sniffing on `sta3`, where you can notice the sniffed packets and some general parameters of the wireless network as output of the command. The sniffed packets will be saved in *capturefile*. The captured packets will be used as input to try to crack the WEP key.

In the output of `airodump`, you can notice the number of captured beacon and data packets. According to the present packet acquisition, how much time do you expect to wait before a significant possibility to discover the WEP key

→

If you generate some packet exchange from `sta1` and `sta2`, e.g. by pinging each other, what changes do you expect, from the attack perspective?

→

Do you think you can do something to speed-up the process?

→

To perform the actual crack, you can use the following command:

```
aircrack-ng -b AP_MAC capturefile
```

and check if you have collected a sufficient amount of packets.

2.6 WPA2 connections

Now it is time to create a WPA connection and examine it. To do so, in the support material you can find a python script that can create a suitable network topology. Extract `authentication_3_wpa.py` from `CYB_lab03_support.zip` and execute the script with the following command:

```
sudo -E python authentication_3_wpa.py
```

This command will create a network topology similar with the one in the previous exercise, which is composed of three stations, `sta1`, `sta2`, `sta3` and `AP1`, but in this case `sta3` is not connected to the AP. For the three stations, 3 terminal windows are opened, as shown in Fig. 6.

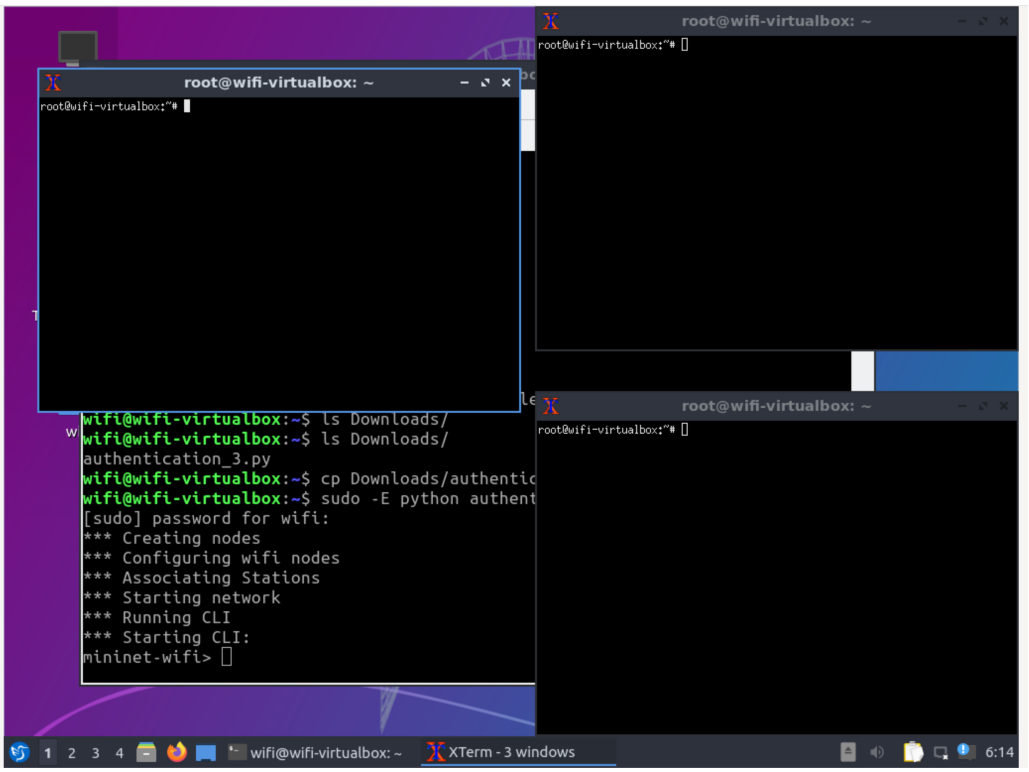

Figure 6: Mininet-Wifi topology (traffic protected with WPA2).

As in the previous exercises, we will use `sta3` to sniff the 802.11 traffic, in particular the traffic exchanged between `sta1` and `sta2`.

Also in this case, what is the network topology?

→

Let's use `sta3` as monitor (set the wifi interface appropriately). next, run `wireshark` (you should be able to see 802.11 traffic sniffed in real time) and then go to the `sta1` terminal and run a ping command to `sta2`.

Stop the capture and analyse the sniffed traffic. You should note in the `wireshark` window of `sta3` that you cannot see anymore ICMP traffic in clear, as the packets have been protected. Are you able to identify with what kind of protocol?

→

Now, let's use the wpa command line interface to disconnect and connect `sta2` to the network

Start `wireshark` in the `sta3` terminal (if not running). Go to the terminal of `sta2`, and run:

```
wpa_cli
```

as shown in Fig. 7.

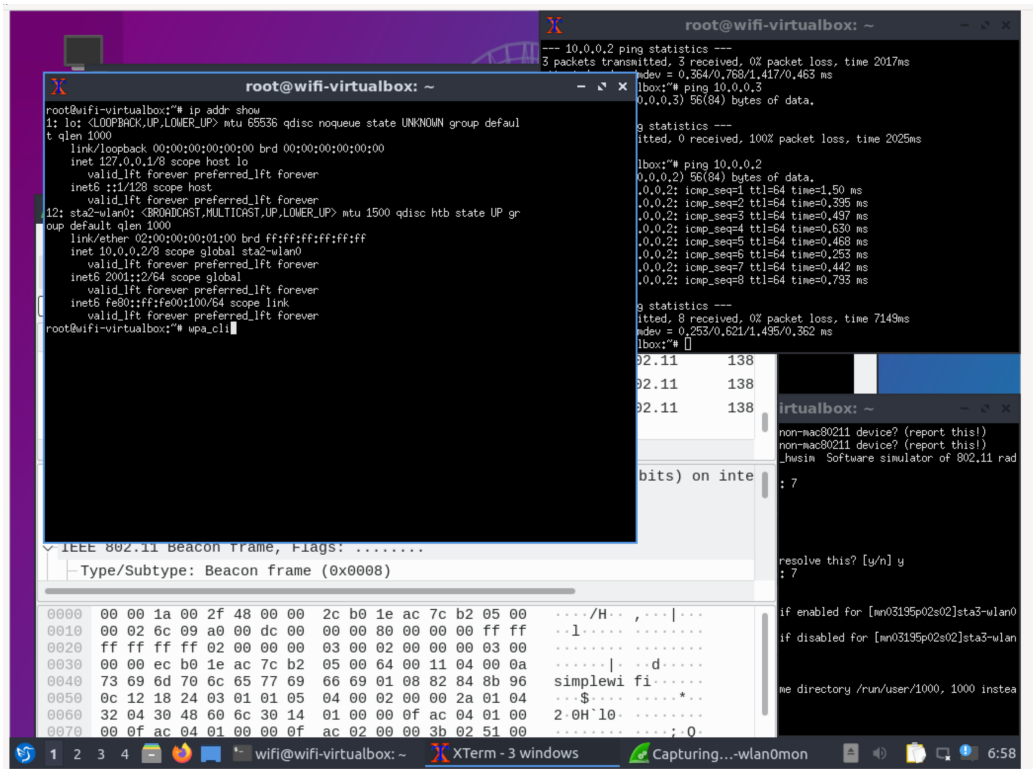

Figure 7: Running wpa_cli in sta2 terminal.

You should see "Interactive mode" and a command line shell. In this shell, run the command:

```
disconnect
```

followed by the command:

```
reconnect
```

as shown in Fig. 8. Then run `quit` to exit from the interactive command shell in `sta2`.

Go to the wireshark window and stop the capture.

What's the effect, from the network point of view, of the `disconnect` and `reconnect` commands?

→

Analyse the EAPOL messages and respond to the following questions:

Where is the authentication information (in which part of the 802.11 packet) ?

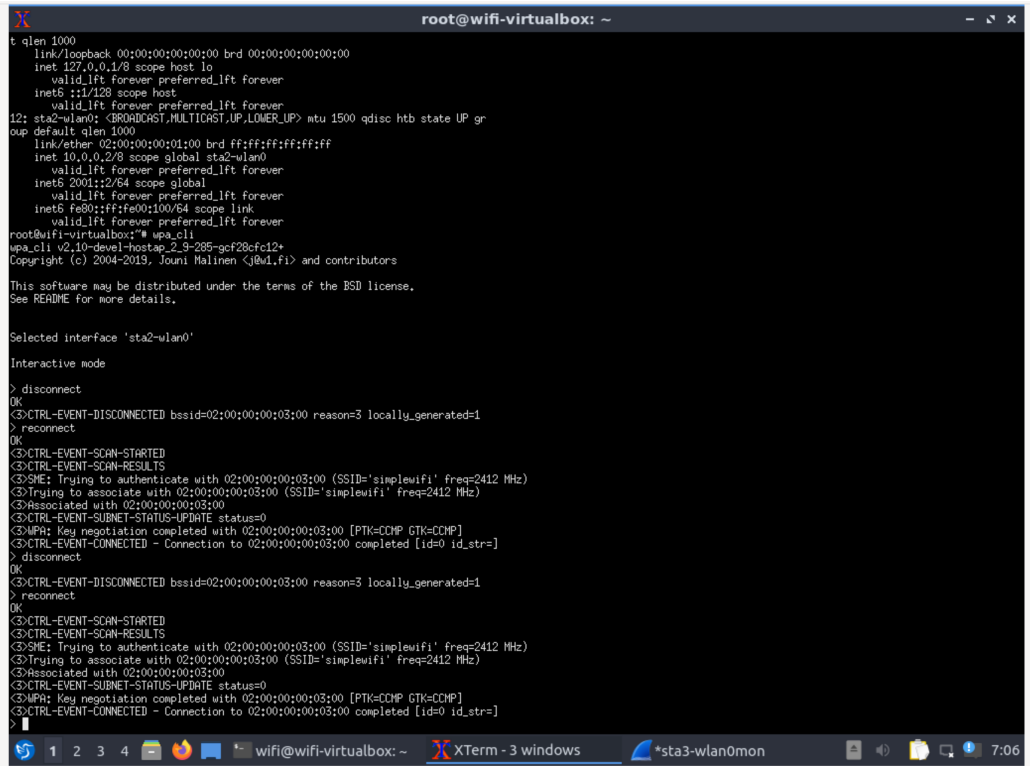

Figure 8: Disconnect and reconnecting to the AP (in sta2).

→

Where is the value of the Anonce in the 4-way handshake ? Who sends the Anonce ?

→

Where is the value of Snonce in the 4-way handshake ? Who sends it ?

→

Which is the RSN IE (Information Element) of the station (in terms of supported Group Cipher Suite and Authentication Key Management) ?

→

Which algorithm has been used for the derivation of the MIC in the Messages 2,3, and 4 of the 4-way handshake ?

→

Where is the enc(GTK) of the 4-way handshake ?

```
→
```

2.7 WPA attacks

2.7.1 Dictionary attack

In `sta3`, launch `airodump` with the following command:

```
airodump-ng -w filepsk sta3-wlan0mon
```

Wait for a while and observe the traffic intercepted.

In `sta2` terminal, run again (as in the previous exercise):

```
wpa_cli
```

In the "Interactive mode" shell run (a couple of times) the command:

```
disconnect
```

and then

```
reconnect
```

In the airodump window, you should note WPA2 (for ENC) and CCMP (for CIPHER). Press Ctrl-c to stop airodump. Airodump has created several files named `filepsk*`, among them there should be a file named `filepsk-01.cap` containing the traffic just captured by airodump.

At this point, imagine to be an attacker that wants to perform a dictionary attack on the traffic intercepted with airodump. Create your own dictionary, by creating a file named `dictionary.txt`, and insert in this file possible passwords (one per line), and see the result (we advice to insert also some pretty common ones, like "`qwerty`", "`12345`" "`123456789a`" "`qwertyuiop`" (who knows, maybe is one of that...).

Next, in `sta3` launch a dictionary attack with `airodump`, by executing the command:

```
aircrack-ng -w dictionary.txt filepsk-01.cap
```

If you are lucky, you may see an output similar to the one shown in Fig. 9, indicating that you have found the right key!

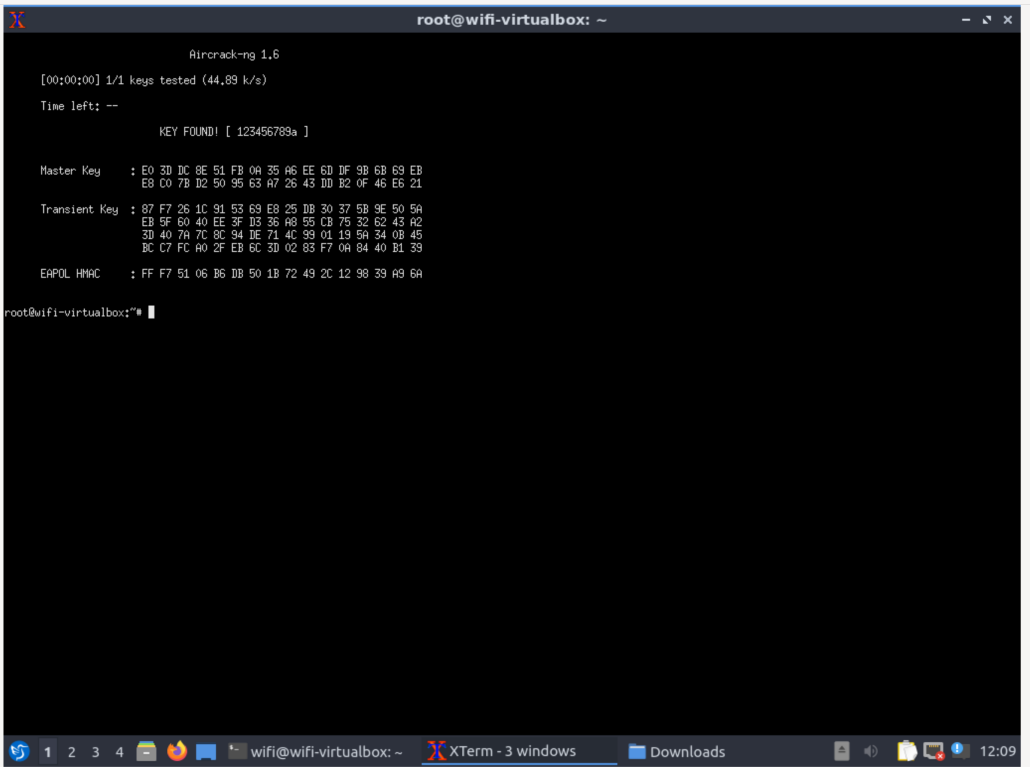

Figure 9: Result of the dictionary attack.