UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Enginyeria Minera, Industrial i TIC

Configurable Wi-Fi AP

Imad Salmi

Roger Miquel

Robert Miralles

Q7 - Sistemes de Comunicació

Enginyeria de Sistemes TIC

Index

1. Introduction

This project involves developing a Wi-Fi Access Point (AP) with features to access control and monitoring. The tool used is Hostapd, a widely used open-source user space daemon for AP and authentication servers.

As the project says, it is based on “Time of day” AP, based on a list of MAC addresses and time periods when those MAC addresses are allowed to use services. The other times they are not allowed to connect. When the “end time” arrives, only the corresponding users are disconnected.

Also, we have to monitor the AP, notifying the “admin” when the “devices of interest” connect. Device is identified by its MAC address, the system should allow a way to provide the list of “devices of interest” and the way of notification is up to the user.

2. Strategy

To do this project we have used the laptop as an AP, using the wireless interface. To give the AP power to this, we have used the hostapd application and DNSMASQ, this second one is used to give a range of IPs to the STA clients.

To allow only privileged users to enter, we will configure a file where only the MACs that the user wants will appear. If the MACs are not in the list, that client will not be able to connect. In addition, we implement that access can only be granted in a time slot set by the user.

3. Implementation

First of all, we download the Hostapd and DNSMASQ applications. To do this we use the following command:

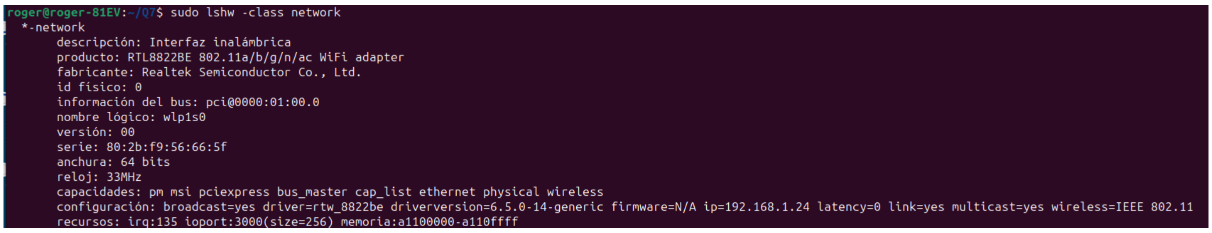
```
Unset
sudo apt-get install hostapd dnsmasq
```

To detect the network cards that our equipment has and how the system recognises them, we use the following code command:

```
Unset
lshw -class network
```

When executing this command we can see that our equipment consists of only one interface which is located under the lines where <*-network> is placed. This line of code provides us with information such as the descriptive name of the interface, its identifier, the virtual name and its type.

This command will give us something like the following:



```
roger@roger-81EV:~/Q7$ sudo lshw -class network
  *-network
       descripción: Interfaz inalámbrica
       producto: RTL8822BE 802.11a/b/g/n/ac WiFi adapter
       fabricante: Realtek Semiconductor Co., Ltd.
       id físico: 0
       información del bus: pci@0000:01:00.0
       nombre lógico: wlp1s0
       versión: 00
       serie: 80:2b:f9:56:66:5f
       anchura: 64 bits
       reloj: 33MHz
       capacidades: pm msi pciexpress bus_master cap_list ethernet physical wireless
       configuración: broadcast=yes driver=rtw_8822be driverversion=6.5.0-14-generic firmware=N/A ip=192.168.1.24 latency=0 link=yes multicast=yes wireless=IEEE 802.11
       recursos: irq:135 ioport:3000(size=256) memoria:a1100000-a110ffff
```

As you can see in the image, <wlp1s0> is the name of the Wi-Fi interface. There are other methods to achieve the same goal, but we used this.
Since we don't have any other interface, we can't provide the internet to our AP users or connect to the internet to send messages like mail.

Then, we build the AP infrastructure, but it does not have the capacity to distribute the IPs to the different clients by itself. This will be done with DHCP, we will do it with DNSMASQ. First of all, the hostapd configuration goes executing a the following command:

```
Unset
sudo emacs /etc/hostapd/hostapd.conf
```

To configure the Hostapd configuration we have the following information:

```
Unset
interface=wlp1s0 #Name of your Interface
driver=nl80211
```

```
auth_algs=1
ignore_broadcast_ssid=0
hw_mode=g  # Could be 'g' for 2.4GHz, 'a' for 5GHz
ssid=SisCom_AP  # Name of the Wi-Fi net
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
macaddr_acl=1
wpa=2
wpa_passphrase=Socrative  #Password
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
ctrl_interface=/var/run/hostapd
```

This configuration sets up the wireless interface, with the MAC list, AP name, password and security parameters. To say which archive we will open when we close the AP we serve the next order:

```
Unset
sudo emacs /etc/default/hostapd
```

When we are in this configuration, on the line where DAEMON_CONF -> Assign the path to the fitxer, which in this case is: /etc/hostapd/hostapd.conf.

The moment you have the AP configured, the client will try to access it. To do so, if it is correct, the DNSMASQ server will give it an IP. Then, to be able to reach different clients you have to configure the DHCP server with DNSMASQ, the file to configure is the following:

```
Unset
sudo emacs /etc/dnsmasq.conf
```

In this configuration you have to add the IP address games to assign the clients, the network mask, the link port… The configuration used is as follows:

```
Unset
interface=wlp1s0
dhcp-range=192.168.1.2,192.168.10.10,255.255.255.0,12h
```

In the configuration we established the port to 5353 because in my case the port 53 was in use. The dhcp-range is the range of IP that dnsmasq will give to the clients. And the 12h is to refresh this IPs.

Also it is very important to do the list with the MACs allowed. We can create a file in the same repository that the python program runs:

```
Unset
sudo emacs mac_lists.txt
```

In this file we will introduce the MAC’s addresses that have permissions to connect to the AP and their time permission ranges.
One example of that file is this one:

```
Unset
2a:73:b5:a0:d3:4a 08:00-21:00
28:16:7f:6f:6d:fe 04:00-23:00
```

To make our AP visible it is necessary to restore dnsmasq before executing the start hostapd command. To do these two actions we do the following:

```
Unset
sudo service dnsmasq restart

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

The code that control the access of our AP is the following one:

```python
Python
import datetime
import subprocess
import time

interface = "wlp1s0"
dispositivos_antiguos=[]

def obtener_dispositivos_conectados():
	try:
	resultado = subprocess.check_output(['arp',
'-a']).decode('utf-8')
	dispositivos =[]
	lineas=resultado.splitlines()
	for linea in lineas:
		palabras=linea.split()
		for palabra in palabras:
			if len(palabra) == 17:
			dispositivos.append(palabra)

	print(f"Dispositivos conectados: {dispositivos}")
	return dispositivos
	except Exception as e:
	print(f"Error al obtener la lista de dispositivos
conectados: {e}")
	return []

def notificar_administrador(mac_dispositivo):
	print("S'ha connectado el dispositivo de interés con MAC:
",mac_dispositivo)

def cargar_horarios_desde_archivo(archivo):
	horarios={}
	try:
	with open(archivo, 'r') as file:
		for linea in file:
			partes = linea.strip().split(' ')
			mac = partes[0]
			horario = partes[1]
			horarios[mac] = horario
	except Exception as e:
	print(f"Error al cargar los horarios desde el archivo:
```

```
{e}")

        return horarios

def obtener_ip(mac):
        ip=""
        try:
        resultado = subprocess.check_output(['arp',
'-a']).decode('utf-8')
        lineas=resultado.splitlines()
        for linea in lineas:
                palabras=linea.split()
                for palabra in palabras:
                        if palabra == mac:
                        ip_par=palabras[1]

        for letra in ip_par:
                if letra not in "()":
                        ip+=letra
        return ip


        except Exception as e:
        print(f"Error al intentar obtener la IP: {e}")


def verificar_acceso(mac,horarios):
        hora_actual = datetime.datetime.now().time()

        if mac in horarios:
        rango_horario = horarios[mac]
        inicio,fin = rango_horario.split('-')
        inicio = datetime.datetime.strptime(inicio,
"%H:%M").time()
        fin = datetime.datetime.strptime(fin, "%H:%M").time()

        if inicio <= hora_actual <= fin:
                return True

        return False
```

```
def echar_dispositivo(mac, ip, interface):
      try:

      subprocess.call(['sudo', 'arp', '-d', ip])

      subprocess.call(['sudo', 'hostapd_cli', '-i', interface,
'deauthenticate', mac])

      print("Dispositivo con MAC:" ,mac, "e
IP:",{ip},"desconectado exitosamente")

      except Exception as e:
      print(f"Error al desconectar el dispositivo: {e}")

def monitorear_dispositivos_de_interes():


      global dispositivos_antiguos
      while True:
      horarios = cargar_horarios_desde_archivo("mac_list.txt")
      dispositivos_conectados =
obtener_dispositivos_conectados()


      for mac_dispositivo in dispositivos_conectados:
            if mac_dispositivo not in dispositivos_antiguos:
                  if mac_dispositivo in dispositivos_de_interes:
                  print("Se ha conectado el dispositivo de
interes: ",mac_dispositivo)

            if mac_dispositivo not in horarios or not
verificar_acceso(mac_dispositivo,horarios):
                  ip = obtener_ip(mac_dispositivo)

echar_dispositivo(mac_dispositivo,ip,interface)

      dispositivos_antiguos=dispositivos_conectados
      # Espera antes de volver a verificar
      time.sleep(5)  # Espera 5 segundos
```

```
if __name__ == "__main__":

      # Lista de dispositivos de interés (direcciones MAC)
      dispositivos_de_interes = ['2a:73:b5:a0:d3:4a',
'28:16:7f:6f:6d:fe']
      print("Dispositivos de interés:
",dispositivos_de_interes)

      # Inicia el monitoreo
      monitorear_dispositivos_de_interes()
```

This python script monitors the presence of certain devices in our AP actions accordingly. Here is an explanation of the functions and general purpose of the script:

```Python
import datetime
import subprocess
import time
```

The script uses Python's datetime, subprocess, and time modules to work with dates and times, execute system commands, and manage times, respectively.

```Python
interface = "wlp1s0"
dispositivos_antiguos = []
```

We have defined the network interface (wlp1s0) of our laptop and a list (dispositivos_antiguos) to store the MAC addresses of previously detected devices.

```Python
def obtener_dispositivos_conectados():
```

Use the arp -a command to get a list of devices connected to the network. It extracts the MAC addresses of the devices and prints them on the console.

Python
```python
def notificar_administrador(mac_dispositivo):
```

Prints a message indicating that a device of interest has been connected.

Python
```python
def cargar_horarios_desde_archivo(archivo):
```

Reads a file (mac_list.txt) that contains MAC addresses and their associated times, and returns a dictionary with this information.

Python
```python
def obtener_ip(mac):
```

Use the command arp -a to obtain the IP associated with a specific MAC address.

Python
```python
def verificar_acceso(mac, horarios):
```

Checks if a device has access based on its MAC address and the associated time range.

Python
```python
def echar_dispositivo(mac, ip, interface):
```

Disconnect a device from the network using arp -d and hostapd_cli commands.

Python
```python
def monitorear_dispositivos_de_interes():
```

Main cycle that continuously monitors the presence of devices of interest, verifies their schedules and takes measures accordingly. It is repeated every 5 seconds.

Finally, the loop defines the list of devices of interest in a list and calls the monitor_dispositivos_de_interes() function to start monitoring.

4. Conclusions

In this project we have learned how to create an access point with our equipment, using tools such as hostapd and DNSMASQ. We have gained knowledge with the different configurations of these, where each variable can have a very different meaning and alter the entire configuration.

We have also created a python script to deal with the MAC addresses that the user wanted, allow them to access depending on the time zone and monitor the devices.

On the other hand, we thought we had to give the internet to the access point by bridging or NATing the ethernet interface. We wasted a lot of time doing it and we didn't succeed. In addition we touched many configurations that later caused us problems for the development of the access point with Hostapd.

In the end, doing the project without giving internet access to the access point made it much more understandable, we learned how to configure the different configuration files that we were asked to configure and how to go step by step to build a complex project.