

HomePwn: Swiss Army Knife for Pentesting of IoT devices

Autores: Pablo González (pablo.gonzalezperez@telefonica.com)

Josué Encinar García (josue.encinargarcia@telefonica.com)

Lucas Fernández (lucas.fernandezaragon@telefonica.com)

Executive Summary

El mundo hiperconectado es una realidad desde hace unos años. La irrupción de millones de dispositivos, de diferente índole, han provocado que la seguridad sea distinta entre cada uno de ellos. La utilización de diferentes tecnologías entre estos dispositivos hace que la seguridad sea heterogénea. Bluetooth Low-Energy, WiFi, NFC son solo algunos de los ejemplos de las tecnologías utilizadas por millones dispositivos que nos rodean como sociedad. Muchos de ellos se encuentran en el hogar o en nuestras oficinas. Los ataques que una empresa puede sufrir pueden llegar a través de una mala configuración de éstos y del aprovechamiento por un atacante para lograr acceso a otros recursos. HomePwn es un framework que proporciona funcionalidades para auditar y hacer pentesting sobre dispositivos conectados a Internet y que disponen de diferentes tecnologías: WiFi, Bluetooth Low-Energy, NFC, etc.

1.- Introducción

La irrupción de los dispositivos conectados y de las diferentes tecnologías para gestionar o conectar dispositivos llegó hace unos años. Cada vez son más tecnologías las que existen a nuestro alrededor. El crecimiento de los dispositivos conectados a Internet es una realidad y es un crecimiento muy grande, exponencial.

El paradigma de Internet de las cosas es un paradigma amplio, lleno de diferentes protocolos. En una porción del Internet de las cosas o IoT se puede decir que existen los dispositivos más cercanos a la sociedad y que están constantemente conectados a Internet y a los seres humanos. La seguridad de las tecnologías que interactúan con estos elementos es fundamental para poder asegurar el correcto funcionamiento de éstos.

El pentesting en el mundo de los dispositivos conectados a Internet más cercanos al ser humanos, en algunas ocasiones también llamados dispositivos de IoT, cobra importancia. Esta importancia la gana debido al volumen de dispositivos, a las diferentes implementaciones que se hacen y a la falta de seguridad en muchos casos. Como ejemplo, se tiene varios dispositivos con el mismo objetivo y que tienen implementaciones de seguridad muy diferentes, en algunos casos ni siquiera existente.

El presente trabajo presenta un nuevo marco de trabajo en el que se proporcionarán las funcionalidades suficientes para poder descubrir, evaluar y auditar tecnologías implementadas en dispositivos de IoT.

Hoy en día, no hay que olvidar, que las empresas disponen en sus centros de trabajo o en sus oficinas de un gran número de estos dispositivos. Con el ya famoso BYOD, *Bring Your Own Device*, las empresas abren un vector de ataque que se puede ver expuesto o incrementado con los diferentes dispositivos que los empleados pueden llevar a la oficina, ya sea en su cuerpo, en su llavero, en su mochila o en su ropa. Las diferentes tecnologías que se pueden utilizar pueden suponer un vector de ataque para los atacantes y los miembros de un Red Team.

2.- El gran volumen de tecnologías Vs Seguridad

La irrupción del paradigma IoT hace que millones de dispositivos se encuentren interconectados. La rapidez de muchos fabricantes por llegar los primeros al mercado hizo que la seguridad estuviera en un segundo lugar. Este hecho tiene, lógicamente, muchos aspectos negativos que con el paso del tiempo hace que se puedan ver.

La tecnología que rodea a los dispositivos IoT es muy diferente, NFC, RFID, Bluetooth, WiFi, etcétera. Existen implementaciones y metodologías para hacer estos dispositivos seguros, pero la existencia de millones de dispositivos previos dónde la seguridad era secundaria hace que, actualmente, se pueda encontrar muchos agujeros de seguridad en dispositivos en producción.

Esto cobra mucha relevancia cuando los dispositivos se encuentran en nuestro cuerpo, en nuestro hogar o en nuestras empresas. El pentesting a través de herramientas dedicadas cobra importancia.

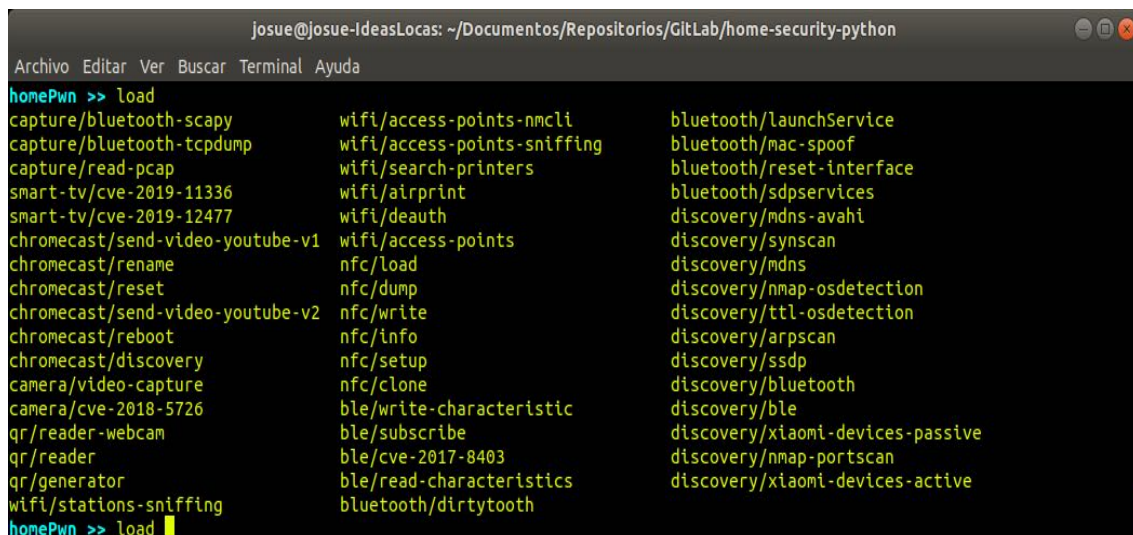
3.- HomePwn

HomePwn es un framework que proporciona funcionalidades para auditar y hacer pentesting sobre dispositivos que, por ejemplo, empleados de una empresa pueden utilizar en su día a día y que se encuentran en un ámbito de trabajo o en el mismo ámbito laboral.



HomePwn tiene una arquitectura modular en la que cualquier usuario puede ampliar la base de conocimiento sobre diferentes tecnologías. Principalmente tiene dos partes diferenciadas:

- Módulos de descubrimiento. En estos módulos se proporcionan funcionalidades relacionadas con la fase de descubrimiento, independientemente de la tecnología a utilizar. Por ejemplo, se puede utilizar para realizar escaneos WiFi con un adaptador en modo monitor, se puede realizar descubrimiento de dispositivos BLE, *Bluetooth Low-Energy*, que hay cercanos y ver el estado de conectividad de éstos, se puede utilizar para mediante protocolos como SSDP o *Simple Service Discovery Protocol* y MDNS o *Multicast DNS* descubrir servicios de dispositivos IoT de hogar u oficina.
- Módulos específicos de la tecnología a auditar. Por otro lado se encuentran los módulos específicos de la tecnología a auditar. Hoy en día, HomePwn puede realizar pruebas de auditoría con tecnologías como WiFi, NFC o BLE, es decir, existen módulos para cada una de estas tecnologías dónde se implementan diferentes vulnerabilidades conocidas o diferentes técnicas para poder evaluar el grado de seguridad de los dispositivos que implementan y se comunican con este tipo de tecnologías.

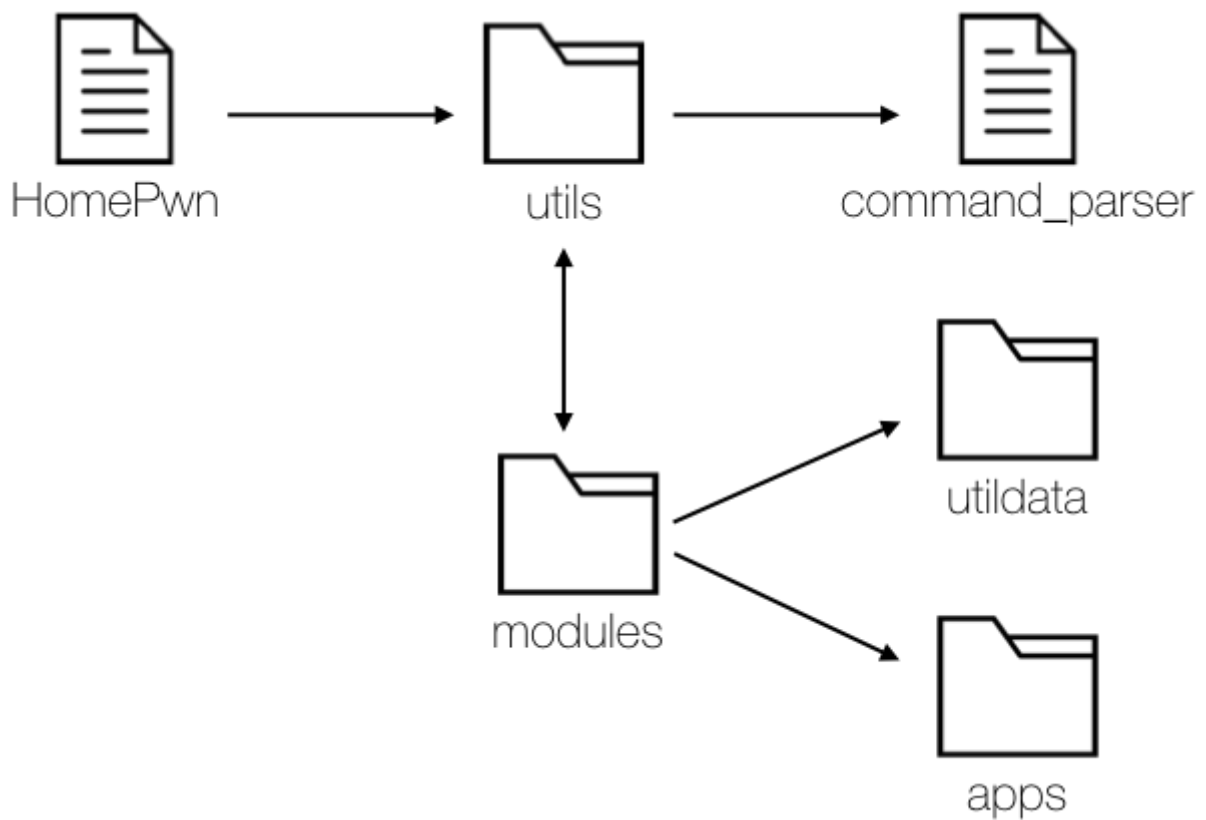


```
josue@josue-IdeasLocas: ~/Documentos/Repositorios/GitLab/home-security-python
Archivo Editar Ver Buscar Terminal Ayuda
homePwn >> load
capture/bluetooth-scapy          wifi/access-points-nmcli        bluetooth/launchService
capture/bluetooth-tcpdump        wifi/access-points-sniffing     bluetooth/mac-spoof
capture/read-pcap                wifi/search-printers            bluetooth/reset-interface
smart-tv/cve-2019-11336          wifi/airprint                   bluetooth/sdpservices
smart-tv/cve-2019-12477          wifi/deauth                     discovery/mdns-avahi
chromecast/send-video-youtube-v1 wifi/access-points              discovery/synscan
chromecast/rename                nfc/load                       discovery/mdns
chromecast/reset                 nfc/dump                       discovery/nmap-osdetection
chromecast/send-video-youtube-v2 nfc/write                      discovery/ttl-osdetection
chromecast/reboot                nfc/info                       discovery/arpscan
chromecast/discovery             nfc/setup                      discovery/ssdp
camera/video-capture             nfc/clone                      discovery/bluetooth
camera/cve-2018-5726              ble/write-characteristic        discovery/ble
qr/reader-webcam                 ble/subscribe                   discovery/xiaomi-devices-passive
qr/reader                        ble/cve-2017-8403               discovery/nmap-portscan
qr/generator                     ble/read-characteristics        discovery/xiaomi-devices-active
wifi/stations-sniffing           bluetooth/dirtytooth
homePwn >> load
```

4. Arquitectura

Se ha dotado a *homePwn* de una arquitectura modular, buscando un fácil mantenimiento y una rápida extensión y crecimiento de la herramienta. Partimos de un archivo con pocas líneas de código que se encarga de lanzar *homePwn* y dar paso al analizador de comandos, que ira distribuyendo el trabajo entre las distintas funciones con las que cuenta la herramienta. La

arquitectura de uso y relación entre partes de la herramienta a alto nivel se puede apreciar en la figura que sigue al párrafo.



La carpeta *modules* se ha dividido en diferentes subcarpetas, según la finalidad que tengan los distintos módulos, para que sea más sencilla la localización, dichas categorías son las que se listan a continuación:

- *ble*
- *bluetooth*
- *camera*
- *capture*
- *chromecast*
- *discovery*
- *smart-tv*
- *wifi*
- *nfc*
- *pc/sc*

Además, la carpeta *apps* está destinada a guardar binarios de posibles aplicaciones externas de las que tenga que hacer uso. Aparte, aunque se ha obviado del esquema, se ha facilitado una carpeta *files*, para que la aplicación guarde ahí por defecto posibles ficheros que generan los módulos, como pueden ser archivos *.pcap*.

La carpeta *utils* tiene funcionalidad auxiliar de ayuda a la herramienta, y que es muy diversa, por ejemplo, contiene el cargador de módulos, el analizador de comandos o distinta funcionalidad para hacer uso en los módulos. Y por último, la carpeta *utildata*, es un lugar donde almacenar información útil y necesaria para los módulos.

5. Requisitos

Para hacer uso de *homePwn* es necesario correr *Linux* como sistema operativo y tener instalado en el equipo una versión de *Python 3.6* o superior. A parte, se necesita hacer la instalación de distintas librerías tanto para *Linux*, como para *Python*, para facilitar el proceso en distribuciones *Debian* y derivadas, todo ello se recoge en un fichero llamado *install.sh* y solo necesitaremos ejecutarlo como sigue:

```
sudo ./install.sh
```

Para poder usar toda la funcionalidad de *homePwn*, se necesita tener una tarjeta de red que permita el modo promiscuo, disponer de un lector *NFC* y contar con *bluetooth*, esto último, si se hace uso de una máquina virtual, requerirá de un *usb bluetooth* externo para un correcto funcionamiento. No obstante, al ser módulos independientes, el no tener alguno de estos 3 requisitos solo limitará a no poder usar los módulos afectados.

6. Cómo empezar a usar *homePwn*

Nos situamos en la carpeta de la herramienta y ejecutamos el fichero *homePwn.py*, gran parte de la funcionalidad requiere ser *root*, por lo tanto, el comando ira precedido de un *sudo*:

```
sudo python3 homePwn.py
```

Si se cumple con todos los requisitos comentados en el apartado anterior, se podrá apreciar el *banner* y el *prompt* de la herramienta, tal y como se muestra en la próxima figura.



Aquí solo se indica cómo empezar *homePwn*, más adelante habrá un apartado donde se verán casos de uso de la herramienta, y va a permitir entender mejor su uso.

7. Cómo hacer un módulo

La funcionalidad principal de *homePwn* viene a través de los módulos, no existe un límite, se pueden crear tantos como se quiera, su creación es rápida y sencilla, la clase se tiene que llamar *HomeModule* y debe de heredar de la clase *Module*. Estos nuevos módulos se irán agregando a la carpeta de la categoría a la que corresponda, si esta no existe, es tan fácil como crear una nueva carpeta dentro de *modules*.

7.1 Clase *Module*

Esta clase padre es el archivo *_module.py* dentro de la carpeta *modules*. Y nos interesa revisar su constructor.

```
class Module(ABC):
    def __init__(self, info, options):
        self.options = options
        self.info = info
        self.name = ""
        self.args = {}
        self.init_args()
        self.update_global()
        self.update_options()
```

Una parte importante son los parámetros que se recibe con el constructor. Corresponden a la información y las opciones del módulo personalizado que se va a crear y se podrá ver posteriormente dicha información en la terminal, así como poder establecer los valores a las opciones del módulo.

- *info*: Es un diccionario que va a contener los siguientes campos: *Name*, *Description*, *Author* y que puede traer otros como *References*, *OS*, *privileges*, o los que sean necesarios.
- *options*: Es un diccionario que contiene el nombre de la opción y un objeto '*Option*'. Con estos atributos se pone un valor por defecto, una descripción de lo que hace esa opción y por último

En el constructor se pueden apreciar llamada a otras funciones, que se utilizar para actualizar las opciones con la configuración global que vaya generando el usuario. Y el atributo *name*, que sirve para dar un nombre al módulo.

7.2 Clase *HomeModule*

La clase *HomeModule* heredará de la clase *Module*, y tendremos que aportar los campos *info* y *options* vistos en el apartado anterior. A parte, la clase padre obliga a implementar la función *run*, que será la encargada de lanzar el módulo. Además, tiene una función a implementar como opcional, *update_complete*, que va a servir para ampliar el autocompletado con el tabulador de la herramienta. En la siguiente figura se puede ver un ejemplo de módulo.


```

class HomeModule(Module):

    def __init__(self):
        information = {"Name": "Discovery Bluetooth ",
                       "Description": "Discover devices with active Bluetooth",
                       "privileges": "root",
                       "OS": "Linux",
                       "Author": "@josueencinar"}

        # -----name-----default_value--description--required?
        options = {"timeout": Option.create(name="timeout", value=5, required=True),
                   "rssi": Option.create(name="rssi", description='dB signal to filter (min value, example -60)')}

        # Constructor of the parent class
        super(HomeModule, self).__init__(information, options)

    # This module must be always implemented, it is called by the run option
    def run(self):
        if not is_root():
            return
        print("Searching BLE devices...\n")
        try:
            timeout = int(self.args["timeout"])
        except:
            timeout = 5
        scan = Scan()
        devices = scan.scan_devices(timeout=timeout)
        scan.show_devices(devices, self.args["rssi"])

```

Para gestionar las opciones se hace uso de Option, para ello dentro del módulo hay que importar:

```
from utildata.dataset_options import Option
```

Existe un número limitado de opciones predefinidas, pero si no existe en el dataset, se creará una genérica, los parámetros que puede recibir son los siguientes:

- *name*: único campo obligatorio, la opción que se va a crear.
- *value*: facilitará un valor por defecto.
- *description*: las opciones predefinidas ya tienen una descripción general, pero se le puede pasar otra, si se considera que se necesita concretar más.
- *required*: indica si una opción es de obligada configuración antes de poder lanzar el módulo, por defecto esta en false.
- *match_pattern*: si una opción solo puede tomar ciertos valores, se especifica con este parámetro, en las predefinidas ya se tiene en cuenta, pero se pueden modificar. Para ello consultar las expresiones regulares, un ejemplo de validación de MAC: `r"^(?:[0-9a-fA-F]:?){12}$"`

En la siguiente figura se puede ver una clase del *dataset*:

```

class MAC(GenericOption):
    def __init__(self, value=None, required=False, description="Mac address", match_pattern=r"^(?:[0-9a-fA-F]:?){12}$"):
        key="mac"
        super(MAC, self).__init__(key, value, required, description, match_pattern)

```

En el anterior ejemplo no se necesita ampliar el autocompletado, en caso de necesidad, se tiene que comenzar con hacer el siguiente *import*:

```
from utils.shell_options import ShellOptions
```

Y como ejemplo se deja parte de un módulo que se amplía el autocompletado a la opción *type*, facilitando al usuario elegir la opción correcta. Se puede apreciar que el primer parámetro es la opción, y el segundo es una lista con las opciones a completar.

```
class HomeModule(Module):  
  
    def __init__(self):  
        information = {"Name": "BLE Characteristics",  
                        "Description": "Get characteristics from a BLE device",  
                        "privileges": "root",  
                        "OS": "Linux",  
                        "Author": "@josueencinar"}  
  
        # -----name-----default_value--description--required?  
        options = {"bmac": Option.create(name="bmac", required=True),  
                   "uuid": Option.create(name="uuid", required=True, description='Specific UUID for a characteristic'),  
                   "type": Option.create(name="type", value="random", required=True, description='Device addr type')  
                   }  
  
        # Constructor of the parent class  
        super(HomeModule, self).__init__(information, options)  
  
        # Autocomplete set option with values  
        def update_complete_set(self):  
            s_options = ShellOptions.get_instance()  
            s_options.add_set_option_values("type", ["random", "public"])
```

La extensión es rápida y fácil, la longitud y la complejidad vendrá dada de la tarea a llevar a cabo. Al agregarla en la carpeta modules la herramienta la reconoce automáticamente.

7.3 Custom Thread

Cuando se está trabajando con tareas que llevarán un largo periodo de tiempo, es interesante que se pueda hacer en segundo plano y se pueda seguir avanzando, por ello se ha creado la posibilidad de crear hilos para ejecutar funciones de *Python* (*new_process_function*) o del sistema operativo (*new_process_command*), que recibirán la función o el comando y el nombre que se le quiere dar a la tarea. Un ejemplo de llamada:

```
new_process_function(py_function, name="MyThread")
```

Si es necesario hacer un volcado de datos, cerrar un fichero, etc. se puede hacer uso de un tercer parámetro, *seconds_to_wait*, que va a esperar x segundos a terminar el proceso. Si se requiere de un comando de sistema, se haría como sigue:

```
def run(self):
    command = f"tcpdump -i {self.args['iface']} -w {self.args['file']}"
    new_process_command(command, "tcpdump")
```

Estas dos funciones se pueden importar del paquete *utils.custom_thread*.

Las tareas en segundo plano no se quedan en el limbo y se pueden revisar y matar, se verá en el siguiente apartado.

8. Conociendo homePwn

Vamos a ver el uso general de la herramienta, y en el próximo punto veremos algunos escenarios relacionados con el pentesting donde se pueda usar la herramienta. Para empezar la ayuda con los comandos disponibles:

```
Core commands
=====
```

Command	Description
-----	-----
help	Help menu
load <module>	Load module
modules [category]	List all modules or a certain category
find <info>	Search modules with concret info
exit quit	Exit application
banner	Show banner
set <option> <value>	Set value for module's option
unset <option>	Unset value for module's option
run	Run module
back	Unload module
show [options info]	Show either options or info
global <option> <value>	Set global option
export	Save global options
import	Load global options (previously exported)
tasks <show/kill> [id]	Show tasks running or kill a task
theme dark light default	Change the theme of the tool
# <command>	Grant terminal commands

Para saber los módulos que existen, se puede hacer uso de *modules*, que listara absolutamente todos los módulos de la herramienta, pero también se puede especificar más, y se podrá buscar por una categoría concreta (se usa el tabulador para que el autocompletado sea de ayuda), como se puede ver en la siguiente figura:

```
homePwn >> modules discovery

Modules list
-----
discovery/mdns-avahi
discovery/synscan
discovery/mdns
discovery/nmap-osdetection
discovery/ttl-osdetection
discovery/arpscan
discovery/ssdp
discovery/bluetooth
discovery/ble
discovery/xiaomi-devices-passive
discovery/nmap-portscan
discovery/xiaomi-devices-active
-----
Modules count: 12
```

Se puede hacer uso del comando *find*, para buscar módulos que contengan un dato concreto, su uso es muy sencillo y se puede ver a continuación:

```
homePwn >> find ble
Searching: ble
-----
wifi/deauth
ble/write-characteristic
ble/subscribe
ble/cve-2017-8403
ble/read-characteristics
discovery/ble
-----
Modules count: 6
```

Para cargar un módulo, se hace uso del comando *load*, para ver las opciones se hace uso de *show options*, para revisar la información se ejecuta *show info*, y para ver ambas simplemente *show*. Para la configuración de las opciones se cuenta con 2 comandos:

- *set*: configura la opción de manera local
- *global*: la configuración se produce a nivel de aplicación, si se carga un módulo con la misma opción, se carga este valor por defecto. Por ejemplo, en el caso de configurar *IP* o *MAC* puede ser de gran ayuda.

Las opciones globales, se pueden importar y exportar con los comandos *import* y *export*. Si se quiere dejar una opción sin valor se hace uso de *unset*. En la siguiente imagen se puede ver un ejemplo de carga de módulo y configuración.

```

homePwn >> load discovery/ble
Loading module...
[+] Module loaded!
homePwn (ble) >> set rssi -60
rssi >> -60
homePwn (ble) >> show options
Options (Field = Value)
-----
|
|_ timeout = 5 ((seconds) Timeout to wait for search responses)
|_ rssi = -60 (dB signal to filter (min value, example -60))

```

Anteriormente, se vio como se podían crear funciones que corrieran en segundo plano, para revisar dichas tareas y acabar con alguna se hace el uso del comando *tasks* acompañado de la acción a realizar, se puede ver en la siguiente figura:

```

homePwn (bluetooth-scapy) >> run
Starting to capture Bluetooth packets
Task running in background... use 'tasks list' to check
homePwn (bluetooth-scapy) >> tasks list

Index (Thread)
-----
|_ 1 = blueScapy 22874 (Alive)

homePwn (bluetooth-scapy) >> tasks kill 1
Writing 81 packets in ./files/blueScapy.pcap
[+] Done!
Task 1 - blueScapy has been killed
homePwn (bluetooth-scapy) >> 

```

La opción *run*, también se ha visto en la anterior figura, por último el comando *back*, nos sirve para hacer un ‘unload’ del módulo, *banner* nos muestra la introducción de la herramienta, *theme* cambia los colores de la herramienta, *exit* y *quit* termina la sesión de la herramienta y anteponer el símbolo *#* permite ejecutar un comando del sistema, nos puede ayudar para ver la *IP* o la interfaz del *bluetooth* por ejemplo, se muestra en la siguiente figura:

```
homePwn >> # hciconfig

hci0:  Type: Primary  Bus: USB
      BD Address: 64:80:99:D8:5D:50  ACL MTU: 1021:5  SCO MTU: 96:6
      UP RUNNING PSCAN ISCAN
      RX bytes:2279 acl:0 sco:0 events:218 errors:0
      TX bytes:33775 acl:0 sco:0 commands:194 errors:0


homePwn >> #ifconfig wlo1

wlo1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.95.253.226  netmask 255.255.252.0  broadcast 10.95.255.255
      inet6 fe80::7ce6:9dc6:1c82:3737  prefixlen 64  scopeid 0x20<link>
      ether 64:80:99:d8:5d:4c  txqueuelen 1000  (Ethernet)
      RX packets 49755  bytes 52056120 (52.0 MB)
      RX errors 0  dropped 0  overruns 0  frame 0
      TX packets 21532  bytes 5127484 (5.1 MB)
      TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Con esta visión global ya se puede hacer uso de homePwn, ya se puede pasar a ver casos de uso dentro de un Pentesting.

9. Escenarios de Pentesting con *homePwn*

Como ejemplo, en esta sección se va a ver una serie de escenarios que se pueden llevar a cabo, pero la herramienta da para mucho más.

9.1 ‘Ataque’ a dispositivos BLE

Como primer escenario, vamos a poner de objetivo los dispositivos BLE, y como en todos los ‘ataques’, el primer paso es llevar a cabo un reconocimiento y fijar el objetivo, para ello se hace uso del módulo discovery/ble, que va a requerir altos privilegios para su uso, en la figura que se muestra a continuación se pueden ver las opciones:

```

homePwn >> load discovery/bl
discovery/bluetooth discovery/ble
homePwn >> load discovery/ble
Loading module...
[+] Module loaded!
homePwn (ble) >> show options
Options (Field = Value)
-----
|
|_timeout = 5 ((seconds) Timeout to wait for search responses)
|_[OPTIONAL] rssi = None (dB signal to filter (min value, example -60))

```

Por defecto trae un *timeout* de 5 segundos, y la opción de *rssi* es opcional, por si se quiere filtrar según la cercanía de los dispositivos. Si lo lanzamos con *run*, veremos los dispositivos en el rango, se puede ver en la siguiente figura (ordenados por RSSI):

```

homePwn (ble) >> run
Searching BLE devices...

```

RSSI	Addr	Manufacturer	Name	Connectable	AddrType
[-40 dB]	46:f2:b3:	Apple	unknown	✓	random
[-44 dB]	68:00:5f:	Apple	unknown	✓	random
[-55 dB]	22:47:72:	Microsoft	unknown	X	random
[-55 dB]	18:7a:93:02:24:84	unknown	AMIYJ_2484	✓	public
[-56 dB]	39:34:80:	Microsoft	unknown	X	random
[-57 dB]	d0:03:df:	Samsung Elect.	unknown	X	public
[-59 dB]	6f:d6:8e:	Apple	unknown	X	random
[-63 dB]	60:cb:dc:	Apple	unknown	✓	random
[-64 dB]	1c:04:75:	Microsoft	unknown	X	random

El resultado tiene la información necesaria para pasar a ver las características que tiene un dispositivo, para ello se elige uno que tenga el tic verde en *Connectable*, en esta ocasión se escoge el dispositivo **AMIYJ_2484**, se trata de un tile, para proceder, se carga el módulo *ble/read-characteristics* y se configura la mac y el tipo de dirección (ver en la siguiente figura).

```

homePwn (ble) >> load ble/read-characteristics
Loading module...
[+] Module loaded!
homePwn (read-characteristics) >> set bmac 18:7a:93:02:24:84
bmac >> 18:7a:93:02:24:84
homePwn (read-characteristics) >> set type public
type >> public

```

El siguiente paso es lanzar el módulo y revisar las características, que indicará si se pueden escribir, leer, etc.

```

homePwn (read-characteristics) >> run
Trying to connect 18:7a:93:02:24:84. (Attempt: 1)
[+] connected
Device Name
|_ uuid: 00002a00
|_ handle: 0x2 (2)
|_ value: AMIYJ_2484
|_ properties: READ
Appearance
|_ uuid: 00002a01
|_ handle: 0x4 (4)
|_ value: Couldn't read
|_ properties: READ

```

En la anterior figura se ven las 2 primeras propiedades que se obtienen, pero hay más, las más interesantes son las de escritura, que se muestran en la siguiente figura:

```

fff2
|_ uuid: 0000fff2
|_ handle: 0x54 (84)
|_ properties: WRITE NO RESPONSE WRITE
fff4
|_ uuid: 0000fff4
|_ handle: 0x61 (97)
|_ properties: NOTIFY
fff5
|_ uuid: 0000fff5
|_ handle: 0x64 (100)
|_ properties: WRITE NO RESPONSE WRITE

```

El tile que se ha marcado como objetivo emite sonidos a través de la característica fff2, si se investiga con un sniffer se puede ver el tipo de información que envía y se puede hacer el envío desde homePwn modificando este dato, para ello se carga *ble/write-characteristic*, donde se configurara *mac*, *uuid* de la característica a escribir, tipo de dirección, los datos a escribir y el tipo de encode, se puede ver en la siguiente figura:


```
homePwn >> load ble/write-characteristic
Loading module...
[+] Module loaded!
homePwn (write-characteristic) >> global bmac 18:7a:93:02:24:84
bmac >> 18:7a:93:02:24:84
homePwn (write-characteristic) >> set uuid 0000fff2
uuid >> 0000fff2
homePwn (write-characteristic) >> set type public
type >> public
homePwn (write-characteristic) >> set data 0xAA0304FFFF
data >> 0xAA0304FFFF
homePwn (write-characteristic) >> set encode hex
encode >> hex
homePwn (write-characteristic) >> run
Trying to connect 18:7a:93:02:24:84. (Attempt: 1)
[+] connected
[+] Good! It's writable!
[+] Done!

Disconnected
homePwn (write-characteristic) >> █
```

Es importante especificar el tipo de dirección correcto, ya que si no, no se producirá la conexión. El dato escrito al Tile, tiene unas partes importantes el **04** va a indicar el número de pitidos que emitirá, en este caso 4 y **FFFF** va a indicar la intensidad, en esta ocasión la máxima.

9.2 Phishing a un dispositivo Bluetooth

La herramienta permite generar perfiles de *bluetooth* falsos, ya sea copiando los valores de un dispositivo en la red, o especificando directamente las características, lo que va a permitir engañar a usuarios para que se conecten a un determinado *bluetooth*, cuando en realidad es un *PC*. Para ello, cargar el módulo *bluetooth/mac_spoof*. Con este módulo se podrá ver si los usuarios pican o confían en determinados dispositivos. Como ayuda de este módulo se ha usado la herramienta [spooftooph](#).

En el ejemplo se va a ‘clonar’ un dispositivo, por eso primero se realiza un descubrimiento de dispositivos bluetooth.

```
homePwn >> load discovery/bluetooth
Loading module...
[+] Module loaded!
homePwn (bluetooth) >> set timeout 5
timeout >> 5
homePwn (bluetooth) >> run
Searching devices...
found 4 devices
-----
70:F8:2F:BD:57:ED - PowerLocus (0x240404)
F4:60:E2:D0:70:AF - nook (0x5a020c)
44:03:2C:7D:21:F6 - joy-fedora (0x1c010c)
28:16:AD:7D:DA:06 - PC-518723 (0xa010c)
homePwn (bluetooth) >> █
```

Aquí el objetivo será PowerLocus, unos auriculares *Bluetooth*, al módulo se le puede pasar el nombre o la *MAC*, preferiblemente esta última. A continuación, la figura de uso del módulo, donde se puede ver el cambio de dirección, la clase, el nombre y el servicio queda en segundo plano. Por último, la conexión de un dispositivo.

```
homePwn (bluetooth) >> load bluetooth/mac-spoof
Loading module...
[+] Module loaded!
homePwn (mac-spoof) >> set iface hci1
iface >> hci1
homePwn (mac-spoof) >> set bmac 70:F8:2F:BD:57:ED
bmac >> 70:F8:2F:BD:57:ED
homePwn (mac-spoof) >> run
Searching bluetooth devices to check MAC...
A nearby device has been found
Trying to change name and MAC
Manufacturer: Cambridge Silicon Radio (10)
Device address: C4:7C:8D:67:51:05
New BD address: 70:F8:2F:BD:57:ED

Address changed
[+] Done!
Starting Bluetooth service to allow connections
iface >> hci1
name >> PowerLocus
class >> 0x240404
Task running in background... use 'tasks list' to check

[+] Agent registered in background
homePwn (mac-spoof) >>
Device Authorized EC:8C:00:00:00:A1

Device Authorized EC:8C:00:00:00:A1
homePwn (mac-spoof) >>
```

Para verificar, se hace uso de *hciconfig* para ver el antes y el después de la ejecución del módulo, se puede apreciar en la figura que se deja a continuación (revisar interfaz *hci1*):

```
josue@josue-IdeasLocas:~/Documentos/Repositorios/GitLab/home-security-python$ hciconfig
hci1:   Type: Primary  Bus: USB
        BD Address: C4:7C:8D:67:51:05  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING
        RX bytes:688 acl:0 sco:0 events:49 errors:0
        TX bytes:3163 acl:0 sco:0 commands:48 errors:0

hci0:   Type: Primary  Bus: USB
        BD Address: 64:80:99:D8:5D:50  ACL MTU: 1021:5  SCO MTU: 96:6
        UP RUNNING PSCAN ISCAN
        RX bytes:59298 acl:245 sco:0 events:1428 errors:0
        TX bytes:38189 acl:246 sco:0 commands:277 errors:0

josue@josue-IdeasLocas:~/Documentos/Repositorios/GitLab/home-security-python$ hciconfig
hci1:   Type: Primary  Bus: USB
        BD Address: 70:F8:2F:BD:57:ED  ACL MTU: 310:10  SCO MTU: 64:8
        UP RUNNING PSCAN ISCAN
        RX bytes:1346 acl:0 sco:0 events:93 errors:0
        TX bytes:4426 acl:0 sco:0 commands:92 errors:0

hci0:   Type: Primary  Bus: USB
        BD Address: 64:80:99:D8:5D:50  ACL MTU: 1021:5  SCO MTU: 96:6
        UP RUNNING PSCAN ISCAN
        RX bytes:59298 acl:245 sco:0 events:1428 errors:0
        TX bytes:38189 acl:246 sco:0 commands:277 errors:0
```

Confirmada la dirección, se puede pasar al siguiente escenario.

9.3 Descubrimiento de dispositivos

Se puede hacer uso de distintos módulos, ya se vio el descubrimiento de dispositivos *Bluetooth*, aquí se va a ver el uso de *SSDP*, y el uso de MDNS. También hay descubrimientos específicos para dispositivos Xiaomi.

9.3.1 Descubriendo dispositivos a través de SSDP.

El primero en hacer las pruebas es al descubrimiento *SSDP*, que se puede configurar con el tipo de servicio a buscar (por defecto *ssdp:all*), y el *timeout* para realizar la búsqueda, en la siguiente figura se puede ver un ejemplo, que ha sido recortado por motivos de tamaño:

```

homePwn (ssdp) >> set service ssdp:all
service >> ssdp:all
homePwn (ssdp) >> run
ip
|_ 192.168.1.1
friendlyName
|_ DrVenkman
manufacturer
|_ Microsoft

ip
|_ 192.168.1.2
friendlyName
|_ SalÃ³n
manufacturer
|_ Google Inc.

```

El listado puede crecer mucho, según los dispositivos conectados y el tipo de servicio que se configure.

9.3.2 Descubriendo dispositivos a través de MDNS

Seguimos descubriendo dispositivos a través de *MDNS* en el siguiente ejemplo.

```

homePwn >> load discovery/mdns
Loading module...
[+] Module loaded!
homePwn (mdns) >> show options
Options (Field = Value)
-----
|
|_service = _http._tcp.local. (Service type string to search for. (_service._protocol))

homePwn (mdns) >> run
Searching. Press q to stop
172.16.17.0/24:1 BrightSign Web Service._http._tcp.local. BrightSign-L3D576003388.local.
10.95.10.10:1 Sketch Mirror (Macbook L...a)._http._tcp.local. Macbook-...a.local.
10.95.10.10:1 Sketch Mirror (mac-g...e)._http._tcp.local. mac-g...e.local.
10.95.10.10:1 Sketch Mirror (28AP05881)._http._tcp.local. 28AP05881.local.
10.30.10.10:1 Sketch Mirror (NTV - M...e G...z)._http._tcp.local. NTV-M...e G...z.local.
10.95.10.10:1 Sketch Mirror (Mac de S...o)._http._tcp.local. Mac-de-S...o.local.
10.95.10.10:1 Sketch Mirror (28AP05028)._http._tcp.local. 28AP05028.local.
10.95.10.10:1 Sketch Mirror (Mac-509948)._http._tcp.local. Mac-509948.local.
10.95.10.10:1 Sketch Mirror (mac-517042)._http._tcp.local. Mac-Violeta.local.
10.95.10.10:1 Sketch Mirror (MacBook Pro de ...n)._http._tcp.local. MacBook-Pro-de-...n.local.
10.95.10.10:1 Sketch Mirror (MAC505602)._http._tcp.local. MAC505602.local.
10.95.10.10:1 Sketch Mirror (28AP05590)._http._tcp.local. 28AP05590.local.
10.95.10.10:1 Sketch Mirror (MacBook Pro de ...a)._http._tcp.local. MacBook-Pro-de-...a.local.
10.95.10.10:1 Sketch Mirror (E...o)._http._tcp.local. E...o.local.

```

Este módulo se queda corriendo hasta que pulsemos la q. Existe otra opción para descubrimiento de *MDNS* haciendo uso de la herramienta *avahi*.

9.4 Captura de paquetes bluetooth y lectura de los pcaps

En la herramienta se da la opción de capturar paquetes *bluetooth* de 2 maneras, una haciendo uso de *tcpdump* y otra a través de *scapy*, que es la que se va a ver en este caso. Para ello cargamos *capture/bluetooth-scapy* y configuramos donde queremos guardar el fichero (por defecto en la carpeta *files*) y la interfaz de *bluetooth* (si es *hci0*, se pone 0). La tarea se queda en segundo plano.

```
homePwn >> load capture/bluetooth-scapy
Loading module...
[+] Module loaded!
homePwn (bluetooth-scapy) >> show options
Options (Field = Value)
-----
|
|_file = ./files/blueScapy.pcap (Remote host IP)
|_iface = 0 (Bluetooth interface (Example: hci0 is 0))

homePwn (bluetooth-scapy) >> run
Starting to capture Bluetooth packets
Task running in background... use 'tasks list' to check
homePwn (bluetooth-scapy) >> █
```

Cuando se quiera parar la captura, se consulta el listado de tareas corriendo, y se usa *task kill ID* para ‘matarla’, al finalizar, informará del número de paquetes capturados y el fichero donde se han guardado.

```
homePwn (bluetooth-scapy) >> tasks list

Index (Thread)
-----
|_ 1 = blueScapy 7556 (Alive)

homePwn (bluetooth-scapy) >> tasks kill 1
Writing 475 packets in ./files/blueScapy.pcap
[+] Done!
Task 1 - blueScapy has been killed
homePwn (bluetooth-scapy) >> █
```

Si se quiere leer el archivo *pcap*, se carga el modulo *capture/read-pcap*. Se tiene que configurar la ruta del fichero a leer y el número de paquetes que se van mostrando, por defecto 0, que mostrará todos, si son muchos no es la mejor opción.

```

homePwn >> load capture/read-pcap
Loading module...
^[[A[+] Module loaded!
homePwn (read-pcap) >> set file ./files/blueScapy.pcap
file >> ./files/blueScapy.pcap
homePwn (read-pcap) >> set pkts 2
pkts >> 2
homePwn (read-pcap) >> run

###[ HCI header ]###
  type      = Command
###[ HCI Command header ]###
  opcode    = 0x2005
  len       = 6
###[ LE Set Random Address ]###
  address    = 02:12:f4:56:dd:05

--show more-- (press q to exit)

```

El paquete capturado también se puede leer en herramientas como *Wireshark*.

9.5 Trabajando con NFC

Otra de las características más destacables de la herramienta es la capacidad de interactuar con dispositivos RFID y NFC. De estos últimos podemos realizar varias operaciones con cualquier Tag NFC que sea compatible con NDEF (NFC Data Exchange Format).

Así con el debido lector NFC conectado en nuestra herramienta, podremos obtener la información precisa acerca de un tag, como el tipo, su identificador de fábrica, si tiene habilitada la lectura y escritura, los diferentes registros que pueda tener...

```

homePwn >> load nfc/info
Loading module...
[+] Module loaded!
homePwn (info) >> run
Waiting for tag...
Type2Tag 'NXP NTAG215' ID=043787E2356281
NDEF Capabilities:
  readable = yes
  writeable = yes
  capacity = 492 byte
  message = 18 byte
NDEF Message:
[+] record 1
  type = 'urn:nfc:wkt:U'
  name = ''
  data = b'\x04lucferbux.dev'

```

Además, este módulo cuenta con una opción para activar el modo *verbose* y ver en qué espacios de la memoria PROM se encuentran los registros, o incluso detectar registros eliminados que continúan en memoria.

No solo podemos visualizar la información, también podremos escribir nuestros propios registros en los tags con permisos de escritura, para eso tenemos el módulo de `nfc/write` en el que podremos definir el tipo de formato NDEF que queremos usar, así como el dato a enviar al tag.

```
homePwn >> load nfc/write
Loading module...
[+] Module loaded!
homePwn (write) >> set ndef_type uri
ndef_type >> uri
homePwn (write) >> set append True
append >> True
homePwn (write) >> set data https://boomernix.com
data >> https://boomernix.com
homePwn (write) >> run
Tag found, writting record...
[ndef.uri.UriRecord('https://lucferbux.dev')]
Done
homePwn (write) >> []
```

Además de esto, también podremos guardar y cargar el contenido de un tag, permitiendo así clonar nuestra etiqueta en varios dispositivos. Con `nfc/dump` crearemos el archivo con la configuración de nuestro tag, que podremos recuperar con `nfc/load`.

9.6 Secuestrando a un Chromecast

Con la herramienta es posible buscar clientes conectados a una determinada red y hacer un ‘ataque’ de deautenticación, para ello se tiene que disponer de una tarjeta de red que permita trabajar en modo promiscuo, por ejemplo las *Alfa network*s. Dentro de la herramienta el primer paso es cargar el módulo `wifi/stations-sniffing`, indicarle la interfaz de red con la que trabajar, y opcionalmente si queremos fijar un canal.

```
[+] wlx00c0ca81fb80 channel: 8

  ch      Client      BSSID (ESSID)
[*] 1 - C4:9D:      D1 (Microsoft Co) - 9C:1F:7:62 (Intra      iFi)
[*] 11 - B8:8A:      B0 (Intel Corpor) - 9C:1F:5:41 (Intra      iFi)
[*] 6 - 34:29:      40 (Chengdu Mero) - 84:A:3E (MOVIS      )
[*] 6 - 74:DA:      05 (          ) - 98:9:57 (MOVISTAR      )
[*] 11 - 38:53:      BE (Apple, Inc. ) - 9C:1:41 (Intran      Fi)
[*] 11 - B4:9D:      26 (          ) - 9C:1:41 (Intran      .Fi)
[*] 6 - D0:2B:      02 (Apple, Inc. ) - 7A:8:F4 (Wifi      )
[*] 11 - B8:53:      52 (Apple, Inc. ) - F8:8:E9 (          )
[*] 1 -          :      C9 (Google      ) - 9C:1:57 (          Mobile)
[*] 6 - F0:D:      D1 (Motorola Mob) - 98:9:57 (MOVISTAR_      )
[*] 6 - 74:      :      8F (Apple, Inc. ) - 98:9:57 (MOVISTAR      )
[*] 6 - 24:      :      34 (SAMSUNG ELEC) - 98:9:57 (MOVISTAR      )
[*] 6 -          :      07 (Xiaomi Commu) - 98:9:57 (MOVISTAR      )
[*] 11 -          :      99 (Google      ) - 9C:1:41 (Intr      WiFi)
[*] 6 - 34:29:      4D (Chengdu Mero) - 84:A:3E (MOVIS.      )
[*] 1 - 4C:34:      51 (Intel Corpor) - 9C:1:21 (Intra      iWiFi)

^C
[!] Closing
```

Se ha modificado la imagen para evitar filtrar la dirección de los dispositivos y puntos de acceso.

Una vez encontremos el objetivo, en este caso el chromecast tomamos nota del *BSSID* de la red, pero ¿cómo sabemos la MAC del *Chromecast*? Muy fácil, usando los módulos de descubrimiento, *chromecast/discovery* o *discovery/ssdp* se puede obtener la IP, si realizamos un ping a la dirección y consultamos la tabla ARP llegamos hasta ese dato. Aquí se omite el proceso.

Con esto pasamos a cargar el módulo *wifi/deauth* que sirve para echar a un cliente de la red (o a todos si se indica el *broadcast*), para ello se configura *bssid*, dirección del objetivo y número de paquetes a enviar '*count*' (-1 para entrar en un bucle infinito), la tarea entrará en segundo plano y se podrá consultar y terminar según lo visto en escenarios anteriores.

```
homePwn >> load wifi/deauth
Loading module...
^[[A[+] Module loaded!
homePwn (deauth) >> set iface wlx00c0ca81fb80
iface >> wlx00c0ca81fb80
homePwn (deauth) >> set bssid 9C:1F:67:
bssid >> 9C:1F:67:
homePwn (deauth) >> set client 5:
client >> 5:
homePwn (deauth) >> set count -1
count >> -1
homePwn (deauth) >> run
Task running in background... use 'tasks list' to check
Sending deauth packets (from 9C:1F:67: to 5:)
homePwn (deauth) >>
```

Una vez echado el *Chromecast* de la red en la que estaba conectado, es posible tratar de traerlo a nuestro lado y configurarlo.

9.6 Secuestrando a un Chromecast

Con la herramienta es posible buscar clientes conectados a una determinada red y hacer un 'ataque' de deautenticación, para ello se tiene que disponer de una tarjeta de red que permita trabajar en

```
[+] wlx00c0ca81fb80 channel: 8

  ch      Client      BSSID (ESSID)
[*] 1 - C4:9D:      D1 (Microsoft Co) - 9C:1F:67:62 (Intra      iFi)
[*] 11 - B8:8A:      B0 (Intel Corpor) - 9C:1F:67:41 (Intra      iFi)
[*] 6 - 34:29:      40 (Chengdu Mero) - 84:A:3E (MOVISTAR_
[*] 6 - 74:DA:      05 (          ) - 98:9:57 (MOVISTAR_
[*] 11 - 38:53:      BE (Apple, Inc. ) - 9C:1F:67:41 (Intran      Fi)
[*] 11 - B4:9D:      26 (          ) - 9C:1F:67:41 (Intran      .Fi)
[*] 6 - D0:2B:      02 (Apple, Inc. ) - 7A:8:F4 (Wifi
[*] 11 - B8:53:      52 (Apple, Inc. ) - F8:8:E9 (I
[*] 1 -  :          C9 (Google      ) - 9C:1F:67:57 (i
[*] 6 - F0:D:      D1 (Motorola Mob) - 98:9:57 (MOVISTAR_
[*] 6 - 74:      8F (Apple, Inc. ) - 98:9:57 (MOVISTAR_
[*] 6 - 24:      34 (SAMSUNG ELEC) - 98:9:57 (MOVISTAR_
[*] 6 -      D7 (Xiaomi Commu) - 98:9:57 (MOVISTAR_
[*] 11 -      99 (Google      ) - 9C:1F:67:41 (Intr      WiFi)
[*] 6 - 34:29:      4D (Chengdu Mero) - 84:A:3E (MOVISTAR_
[*] 1 - 4C:34      51 (Intel Corpor) - 9C:1F:67:21 (Intra      iWiFi)

^C
```

modo promiscuo, por ejemplo las *Alfa networks*. Dentro de la herramienta el primer paso es cargar el módulo *wifi/stations-sniffing*, indicarle la interfaz de red con la que trabajar, y opcionalmente si queremos fijar un canal.

Se ha modificado la imagen para evitar filtrar la dirección de los dispositivos y puntos de acceso.

Sabiendo la red objetivo, se toma nota del BSSID, pero ¿cómo sabemos la *MAC* del *Chromecast*?

Se tendrá que revisar la compañía a la que pertenece el cliente, en este caso Google, y si hay varios habrá que ir probando, o directamente configurar la dirección *broadcast* en el siguiente módulo

Teniendo los dos datos recopilados en el paso anterior, para echar al *Chromecast* de la red es necesario cargar el módulo *wifi/deauth*, donde se tendrá que configurar *bssid*, dirección del objetivo (*broadcast* para todos los clientes) y número de paquetes a enviar '*count*' (-1 para entrar en un bucle infinito), la y tarea entrará en segundo plano y se podrá consultar y terminar según lo visto en escenarios anteriores.

```
homePwn >> load wifi/deauth
Loading module...
^[[A[+] Module loaded!
homePwn (deauth) >> set iface wlx00c0ca81fb80
iface >> wlx00c0ca81fb80
homePwn (deauth) >> set bssid 9C:1F:67:5:C9
bssid >> 9C:1F:67:5:C9
homePwn (deauth) >> set client 9C:1F:67:5:C9
client >> 9C:1F:67:5:C9
homePwn (deauth) >> set count -1
count >> -1
homePwn (deauth) >> run
Task running in background... use 'tasks list' to check
Sending deauth packets (from 9C:1F:67:5:C9 to 9C:1F:67:5:C9)
homePwn (deauth) >>
```

Una vez echado el *Chromecast* de la red en la que estaba conectado, el dispositivo levantará un punto de acceso y será posible llevarle a nuestra red, configurarle y enviarle contenido, para ello se puede hacer uso de los módulos de envío en la categoría '*chromecast*' (según versión del dispositivo), o si se prefiere hacerle un cambio de nombre, también existe la opción.