

MEMORIA FINAL

- **Descripción y objetivo inicial de la investigación**

Este proyecto de investigación se basa en encontrar algún tipo de vulnerabilidad en un sistema de control remoto de domótica , en este caso el dispositivo es **el Broadlink RM mini3 Universal** (https://www.amazon.es/Broadlink-RM-Mini3-Black-Bean-acondicionado/dp/B07GP73H91/ref=sr_1_3?_mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=365P1LES4HPWT&dchild=1&keywords=broadlink+rm+mini+3&qid=1609869782&s=electronics&prefix=broad%2Celectronics%2C198&sr=1-3)

Al principio de la investigación sin tener mucha idea de que podríamos encontrar el objetivo inicial era como he comentado antes encontrar alguna falla de seguridad tanto en la autenticación con la wifi , en la autenticación con usuario en la App propietaria , en los mensajes entre los dispositivos domóticos de la casa y el RM3 , en la manera de conectarse o incluso en la propia App.

En términos generales analizar todo el comportamiento de este dispositivo en todo el espectro de uso

- **Metodología o fases desarrolladas durante la investigación**

- Investigación previa

La primera fase fue recabar información sobre el dispositivo y sus posibles vulnerabilidades.

- Investigación del dispositivo

La siguiente fase fue encontrar información sobre el funcionamiento del dispositivo de manera normal, aquí en esta fase analice la app y cómo funcionaba.

La aplicación es sencilla , nos pide una cuenta y contraseña es decir, tenemos que registrarnos o podemos usar la de Facebook

Viendo que las funcionalidades que tiene la aplicación son las básicas en un dispositivo de este tipo y no te permite una mayor personalización empecé investigar si se podía manipular de otra forma más “profesional”.

- Trabajo sobre el dispositivo

La siguiente fase fue buscar si existía alguna forma de utilizar el dispositivo sin tener que utilizar la app.

Por último empecé a comprobar que se podía hacer y si podía aprovecharme de alguna de las características que disponía fuera de la app.

- Detalles de las acciones y fases desarrolladas, evidenciando los logros obtenidos

- **Investigación previa**

En esta parte de la investigación busque a priori si ya alguien había encontrado algunas vulnerabilidades

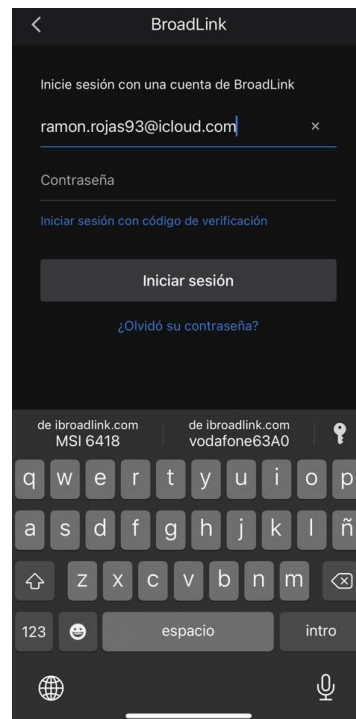
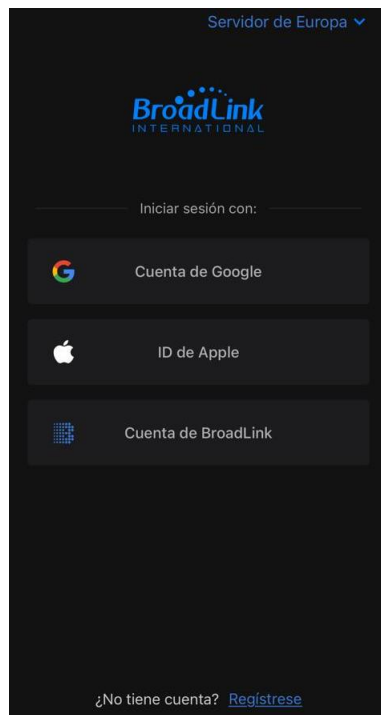
Esta parte de la investigación me dio muchas pistas sobre que puertos podría usar el dispositivo y gente que ya había preguntado si existía alguna forma de controlarlo fuera de la app, esto me llevo a varios GitHub.

- <https://github.com/felipediel/broadlink-hacktools> (Este parecía el más prometedor)
- <https://github.com/eschava/broadlink-mqtt>
- <https://github.com/radinsky/broadlink-http-rest>
- <https://github.com/davorf/BlackBeanControl>
- <https://github.com/mjg59/python-broadlink> (Este es el que acabe usando)

- **Investigación del dispositivo**

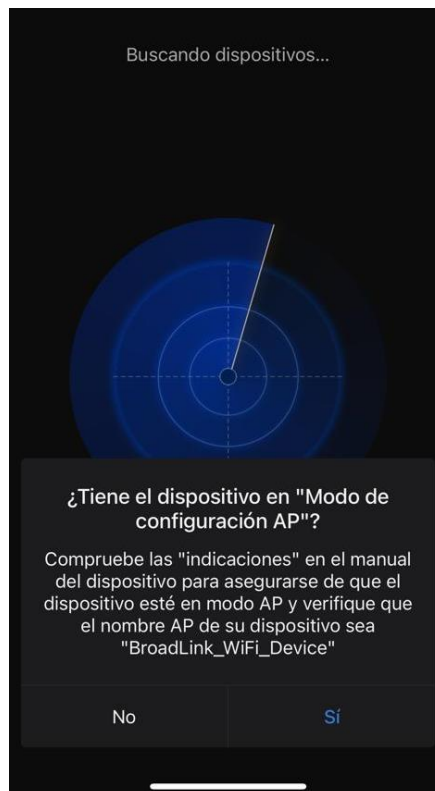
Siguiendo el manual de instrucciones , configure el dispositivo y la app

Lo primero crear una cuenta en la aplicación:

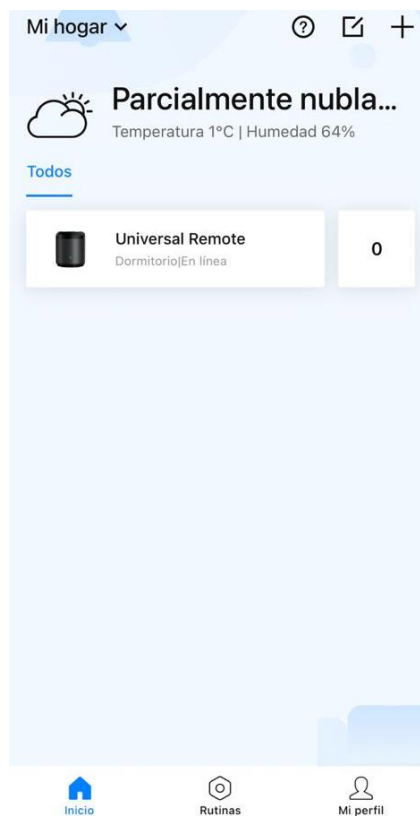


El dispositivo se pone en Modo AP y la aplicación lo busca: Cuando lo encuentra te pide que le des la contraseña de la Wifi al que tienes el móvil

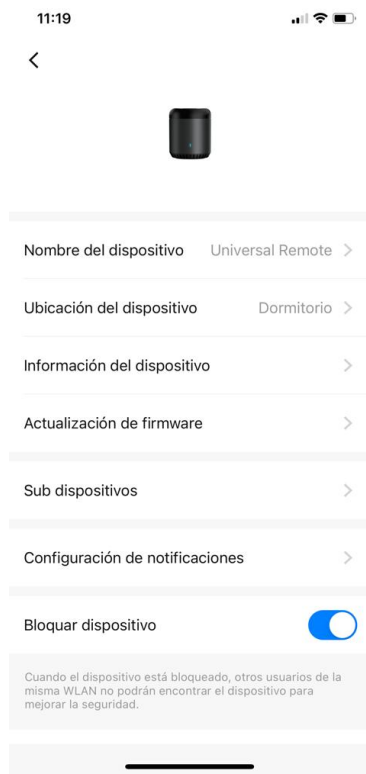
conectado para conectarse él y así poder tener conexión con los demás dispositivos



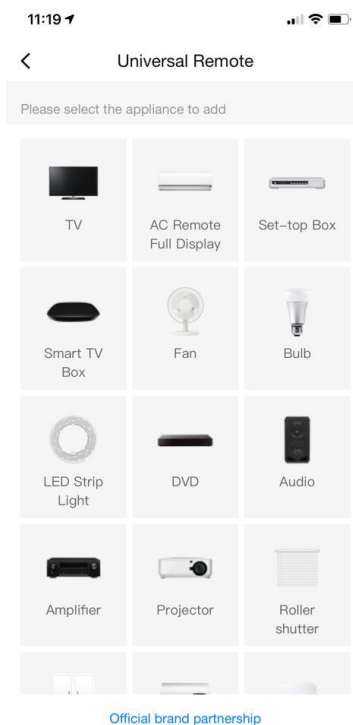
Una vez enlazado nos aparece en la pantalla de inicio:



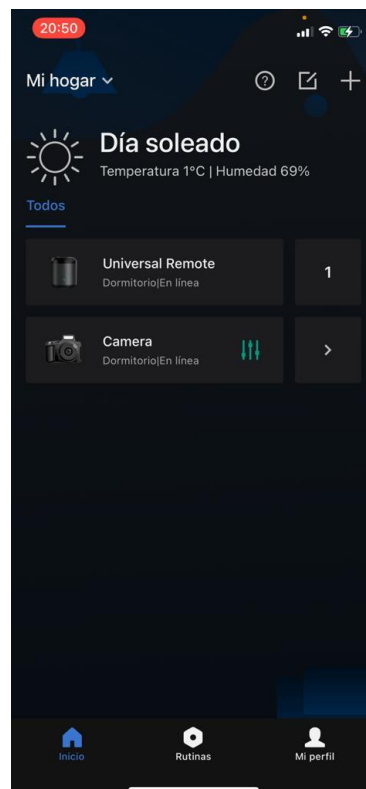
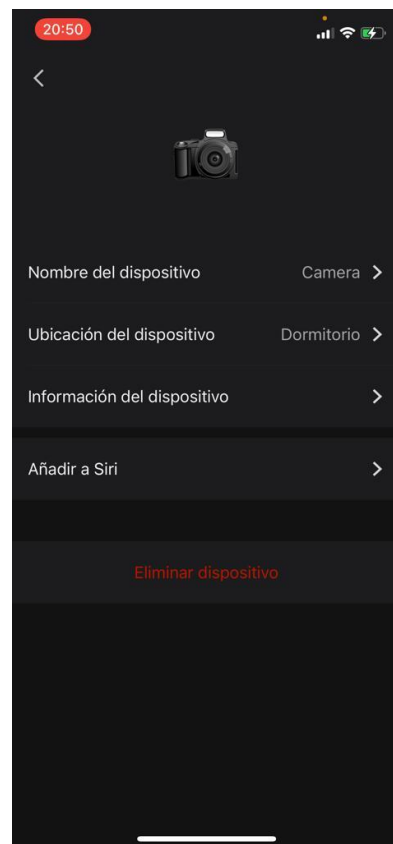
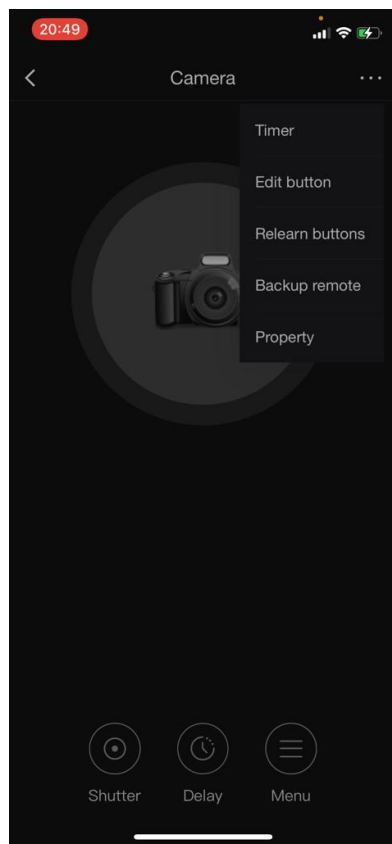
Si pulsamos en el nos muestra toda la información:



Si pulsamos el cero al lado del nombre se abre un menú para añadir un dispositivo

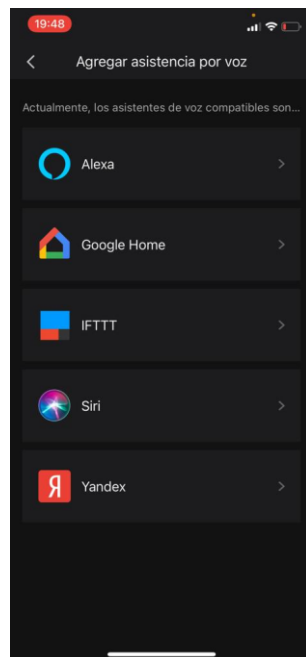


En mi caso probé a sincronizar una cámara de fotos para ver si podía hacer fotos con el RM , aparece una lista de marcas de dispositivos y aplica unos botones generales

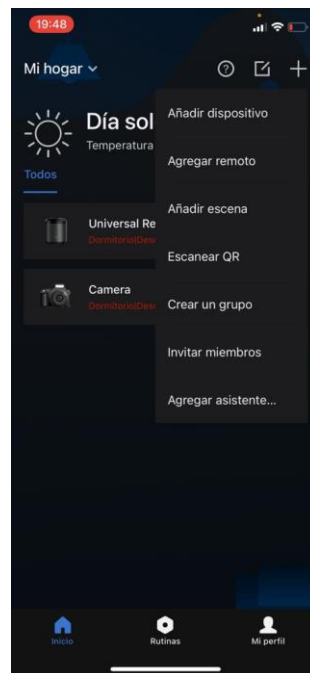


Otras funciones:

Asistencia por voz



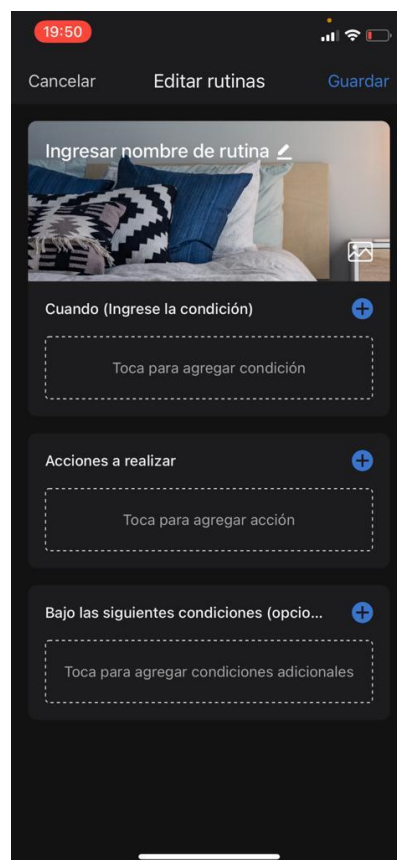
Opciones generales

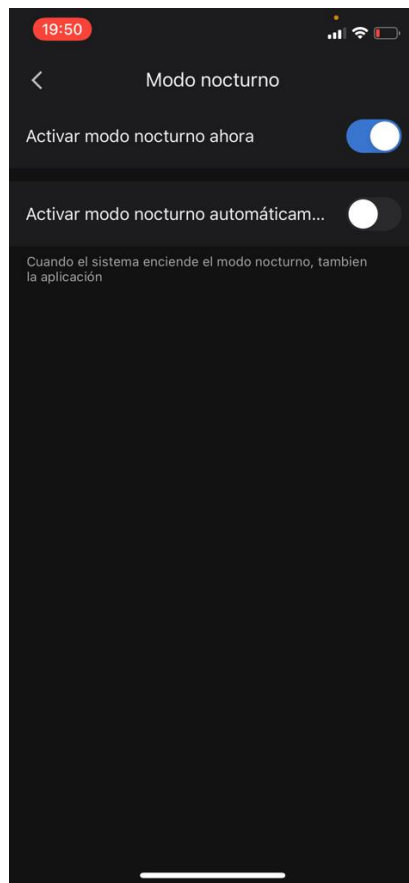


Invitar miembros al dispositivo

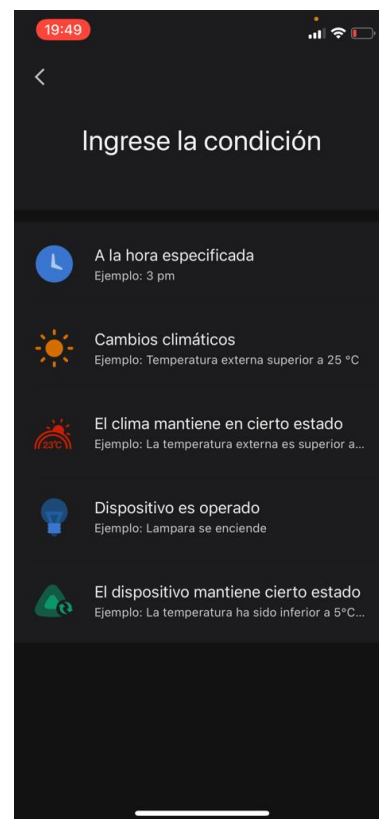


Crear rutinas





Condicion para las rutinas



○ Trabajo sobre el dispositivo

Lo primero que hice fue un Nmap para obtener información del dispositivo

```
kali@kali:~$ sudo nmap 192.168.0.35 -sV -v -p- -O
[sudo] password for kali:
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-02 05:26 EST
NSE: Loaded 45 scripts for scanning.
Initiating ARP Ping Scan at 05:26
Scanning 192.168.0.35 [1 port]
Completed ARP Ping Scan at 05:26, 0.27s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 05:26
Completed Parallel DNS resolution of 1 host. at 05:26, 13.01s elapsed
Initiating SYN Stealth Scan at 05:26
Scanning 192.168.0.35 [65535 ports]
Increasing send delay for 192.168.0.35 from 0 to 5 due to 1886 out of 6285 dropped probes since last increase.
SYN Stealth Scan Timing: About 14.86% done; ETC: 05:29 (0:02:58 remaining)
SYN Stealth Scan Timing: About 22.50% done; ETC: 05:30 (0:03:30 remaining)
SYN Stealth Scan Timing: About 41.17% done; ETC: 05:31 (0:03:14 remaining)
SYN Stealth Scan Timing: About 48.82% done; ETC: 05:31 (0:02:54 remaining)
SYN Stealth Scan Timing: About 56.50% done; ETC: 05:32 (0:02:31 remaining)
SYN Stealth Scan Timing: About 64.26% done; ETC: 05:32 (0:02:06 remaining)
SYN Stealth Scan Timing: About 71.88% done; ETC: 05:32 (0:01:40 remaining)
SYN Stealth Scan Timing: About 79.26% done; ETC: 05:32 (0:01:15 remaining)
SYN Stealth Scan Timing: About 86.74% done; ETC: 05:32 (0:00:48 remaining)
Completed SYN Stealth Scan at 05:32, 371.93s elapsed (65535 total ports)
Initiating Service scan at 05:32
Initiating OS detection (try #1) against 192.168.0.35
NSE: Script scanning 192.168.0.35.
Initiating NSE at 05:32
Completed NSE at 05:32, 0.00s elapsed
Initiating NSE at 05:32
Completed NSE at 05:32, 0.00s elapsed
Nmap scan report for 192.168.0.35
Host is up (0.017s latency).
All 65535 scanned ports on 192.168.0.35 are closed
MAC Address: C8:F7:42:83:98:60 (HangZhou Gubei Electronics Technology)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: specialized|general purpose
Running: DTE embedded, lwIP 1.4.X, Enlogic embedded, Ocean Signal embedded, Philips embedded
OS CPE: cpe:/a:lwip_project:lwip cpe:/h:philips:hue_bridge cpe:/a:lwip_project:lwip:1.4
OS details: DTE Energy Bridge (lwIP stack), Enlogic PDU (FreeRTOS/lwIP), Ocean Signal E101V emergency beacon (FreeRTOS/lwIP), Philips Hue Bridge (lwIP stack v1.4.0)
Network Distance: 1 hop

Read data files from: /usr/bin/..share/nmap
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 386.32 seconds
Raw packets sent: 67545 (2.973MB) | Rcvd: 65601 (2.624MB)
```


En este primer escaneo no obtuve ninguna información porque fue un escaneo TCP y no tenía ningún puerto con este protocolo así que probé con **UDP**

```
kali@kali:~$ sudo nmap 192.168.0.35 -sU -p- -v -T4 -sV
Starting Nmap 7.80 ( https://nmap.org ) at 2021-01-02 06:49 EST
NSE: Loaded 45 scripts for scanning.
Initiating ARP Ping Scan at 06:49
Scanning 192.168.0.35 [1 port]
Completed ARP Ping Scan at 06:49, 0.17s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 06:49
Completed Parallel DNS resolution of 1 host. at 06:49, 13.02s elapsed
Initiating UDP Scan at 06:49
Scanning 192.168.0.35 [65535 ports]
UDP Scan Timing: About 30.66% done; ETC: 06:51 (0:01:10 remaining)
UDP Scan Timing: About 62.18% done; ETC: 06:51 (0:00:37 remaining)
Completed UDP Scan at 06:51, 95.97s elapsed (65535 total ports)
Initiating Service scan at 06:51
Scanning 2 services on 192.168.0.35
Discovered open port 16385/udp on 192.168.0.35
Discovered open|filtered port 16385/udp on 192.168.0.35 is actually open
Discovered open port 80/udp on 192.168.0.35
Discovered open|filtered port 80/udp on 192.168.0.35 is actually open
Service scan Timing: About 50.00% done; ETC: 06:53 (0:00:57 remaining)
Completed Service scan at 06:52, 97.75s elapsed (2 services on 1 host)
NSE: Script scanning 192.168.0.35.
Initiating NSE at 06:52
Completed NSE at 06:52, 0.12s elapsed
Initiating NSE at 06:52
Completed NSE at 06:52, 0.05s elapsed
Nmap scan report for 192.168.0.35
Host is up (0.077s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE VERSION
80/udp    open  snmp    SNMPv3 server
16385/udp open  unknown
```

Bien tenemos dos puertos UDP abiertos , el puerto 16385 es un puerto que usa Apple para comunicarse con sus dispositivos Apple FaceTime, Apple Game Center (RTP/RTCP)

A partir de aquí intente conectarme a ambos puertos con netcat , busque algunos exploits para esa versión de SNMPv3 de ejecución de código pero ninguno daba resultado y entonces fue cuando me para a analizar las librerías existentes de Python que encontré en la investigación previa.

```
kali@kali:~$ git clone https://github.com/mjg59/python-broadlink
Cloning into 'python-broadlink'...
remote: Enumerating objects: 975, done.
remote: Total 975 (delta 0), reused 0 (delta 0), pack-reused 975
Receiving objects: 100% (975/975), 288.47 KiB | 1.15 MiB/s, done.
Resolving deltas: 100% (514/514), done.
kali@kali:~$ cd python-broadlink/
kali@kali:~/python-broadlink$ ls
broadlink cli LICENSE protocol.md README.md requirements.txt setup.py
kali@kali:~/python-broadlink$ cd broadlink/
kali@kali:~/python-broadlink/broadlink$ ls
alarm.py climate.py cover.py device.py exceptions.py helpers.py __init__.py light.py remote.py sensor.py switch.py
kali@kali:~/python-broadlink/broadlink$ cd ..
kali@kali:~/python-broadlink$ pip install -r requirements.txt
Collecting cryptography==2.6.1
  Downloading cryptography-2.6.1-cp34-abi3-manylinux1_x86_64.whl (2.3 MB)
Requirement already satisfied: six>=1.4.1 in /usr/lib/python3/dist-packages (from cryptography==2.6.1->r requirements.txt (line 1)) (1.15.0)
Collecting cffi>=1.11.3, >=1.8
  Downloading cffi-1.14.4-cp38-cp38-manylinux1_x86_64.whl (411 kB)
Requirement already satisfied: asn1crypto>=0.21.0 in /usr/lib/python3/dist-packages (from cryptography==2.6.1->r requirements.txt (line 1)) (0.24.0)
Collecting pycparser
  Downloading pycparser-2.20-py2.py3-none-any.whl (112 kB)
Installing collected packages: pycparser, cffi, cryptography
Successfully installed cffi-1.14.4 cryptography-2.6.1 pycparser-2.20
```


Una vez instalé la librería probé a ver que podía hacer con el dispositivo en distintas configuraciones, esta primera prueba es con el dispositivo ya vinculado al móvil y a la red. Probé varias de las funciones que contenía la librería pero sin mucho éxito

```
kali@kali:~/python-broadlink$ python3
Python 3.8.4 (default, Jul 13 2020, 21:16:07)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import broadlink
>>> broadlink.discover()
[<rm4: Broadlink RM mini 3 (0x5f36) at 192.168.0.35:80 | c8:f7:42:83:98:60 | Universal Remote | Unlocked>]
>>> devices[0].auth()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'devices' is not defined
>>> devices=broadlink.discover()
>>> devices[0].auth()
True
>>> devices[0].enter_learning
<bound method rm.enter_learning of <rm4: Broadlink RM mini 3 (0x5f36) at 192.168.0.35:80 | c8:f7:42:83:98:60 | Universal Remote | Unlocked>
>>> devices[0].enter_learning()
```

Investigue un poco la función para mandar las “órdenes” en este caso `send_packet()`, para ver si yo podría mandar código propio con esta función y ser ejecutado, pero la función esta hardcodeada en código máquina, esperando unos valores concretos, aunque sí conseguí cambiar el nombre del dispositivo.

```
>>> devices[0].get_fwversion()
44057
>>> devices[0].get_type()
'RM4'
>>> devices[0].send_packet()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: send_packet() missing 2 required positional arguments: 'command' and 'payload'
>>> devices[0].set_name()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set_name() missing 1 required positional argument: 'name'
>>> devices[0].set_name('Remote Ramon')
>>>
KeyboardInterrupt
>>> exit()
kali@kali:~/python-broadlink$
```

Esta es la definición de la función

```
def send_packet(self, command: int, payload: bytes) -> bytes:
    """Send a packet to the device."""
    self.count = (self.count + 1) & 0xFFFF
    packet = bytearray(0x38)
    packet[0x00] = 0x5A
    packet[0x01] = 0xA5
    packet[0x02] = 0xAA
    packet[0x03] = 0x55
    packet[0x04] = 0x5A
    packet[0x05] = 0xA5
    packet[0x06] = 0xAA
```

Y este es un uso de ella

```
response = self.send_packet(0x65, payload)
check_error(response[0x22:0x24])
payload = self.decrypt(response[0x38:])
```

Aparte de esto investigue como cifra y descifra sus mensajes el dispositivo

```
def encrypt(self, payload: bytes) -> bytes:
    """Encrypt the payload."""
    encryptor = self.aes.encryptor()
    return encryptor.update(payload) + encryptor.finalize()

def decrypt(self, payload: bytes) -> bytes:
    """Decrypt the payload."""
    decryptor = self.aes.decryptor()
    return decryptor.update(payload) + decryptor.finalize()
```

Esta fue la parte propia del dispositivo y luego investigue el módulo **cli** de la librería

```
kali@kali:~/python-broadlink/cli$ ./broadlink_cli --help
usage: broadlink_cli [-h] [--device DEVICE] [--type TYPE] [--host HOST] [--mac MAC] [--temperature] [--energy] [--check] [--checknl] [--turnon] [--turnoff] [--turnnlon] [--turnnloff] [--switch] [--send] [--sensors] [--learn]
                    [--rfscanlearn] [--learnfile LEARNFILE] [--durations] [--convert] [--joinwifi JOINWIFI JOINWIFI]
                    [data [data ...]]

positional arguments:
  data                  Data to send or convert

optional arguments:
  -h, --help            show this help message and exit
  --device DEVICE        device definition as 'type host mac'
  --type TYPE            type of device
  --host HOST            host address
  --mac MAC              mac address (hex reverse), as used by python-broadlink library
  --temperature          request temperature from device
  --energy               request energy consumption from device
  --check                check current power state
  --checknl              check current nightlight state
  --turnon               turn on device
  --turnoff              turn off device
  --turnnlon             turn on nightlight on the device
  --turnnloff            turn off nightlight on the device
  --switch               switch state from on to off and off to on
  --send                 send command
  --sensors               check all sensors
  --learn                learn command
  --rfscanlearn           rf scan learning
  --learnfile LEARNFILE save learned command to a specified file
  --durations             use durations in micro seconds instead of the Broadlink format
  --convert               convert input data to durations
  --joinwifi JOINWIFI JOINWIFI
                        Args are SSID PASSPHRASE to configure Broadlink device with
```

En este módulo descubrí que no es necesario conectarte mediante la App al dispositivo , si pulsas el botón de reset hasta que la luz parpadea rápidamente y luego lo vuelves a pulsar el RM emite su propia wifi a la que puedes conectarte para encontrar el dispositivo con el módulo cli

```
kali@kali:~/python-broadlink/cli$ ./broadlink_discovery
Discovering ...
#####
RM4
# broadlink_cli --type 0x5f36 --host 192.168.0.35 --mac c8f742839860
Device file data (to be used with --device @filename in broadlink_cli) :
0x5f36 192.168.0.35 c8f742839860
```

Si nos conectamos de esta manera y hacemos las pruebas anteriores nos damos cuenta de que el nombre desaparece y aparecen unas letras en chino , pero además si queremos conectarlo a nuestra red wifi tenemos que usar la función de **broadlink,setup(wifi,pass,parametro)**

interactuar con el RM pero lo intente a la hora de autenticarme con el dispositivo y tampoco capture esos paquetes.

```
from broadlinkhacktools import PacketDecryptor, PacketPrinter, PersistenceHandler
from broadlinkhacktools.protocol.const import DEFAULT_IV, DEFAULT_KEY
```

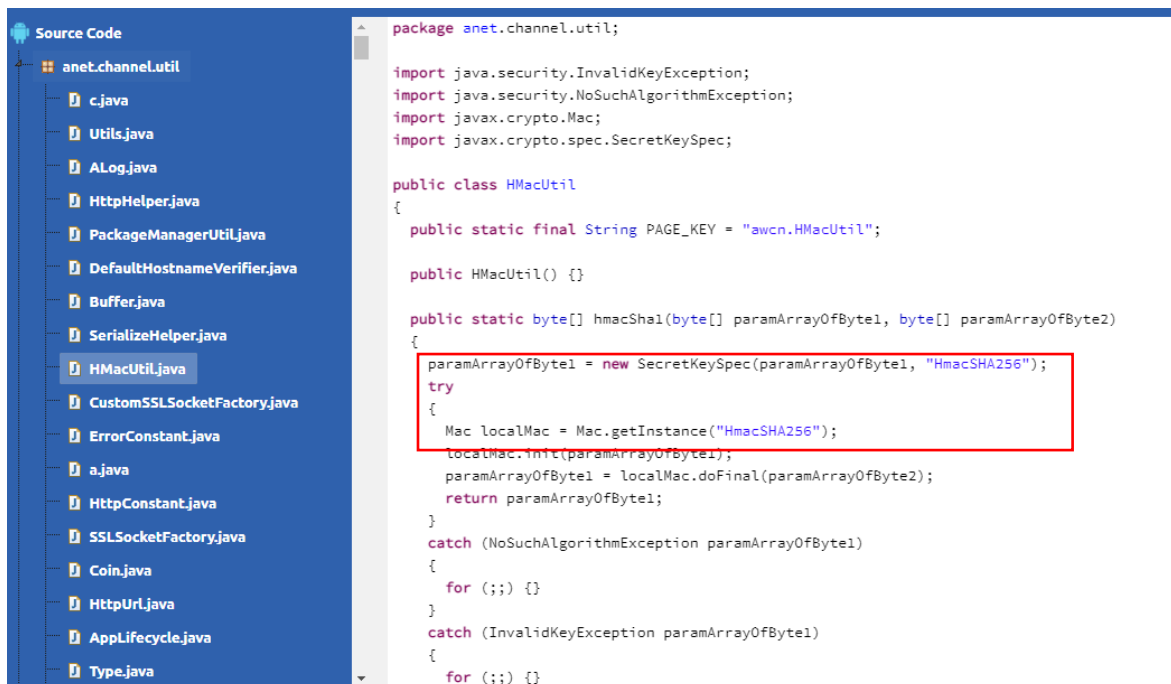
```
# Load packets from binary files.
src_folder = 'some_folder'
packets = PersistenceHandler.load_packets(src_folder)

# Decrypt packets using default key.
decryptor = PacketDecryptor(DEFAULT_KEY, DEFAULT_IV)
decryptor.decrypt(packets)

# Print packets to a file.
printer = PacketPrinter()
with open('packets.txt', 'w+') as file:
    for packet in packets:
        printer.print(packet, file=file)
```

○ Código fuente

Por último descompile la aplicación del móvil para Android para ver si existía alguna dirección en el código o cualquier parte del código que pudiera darme información sensible.



```
package anet.channel.util;

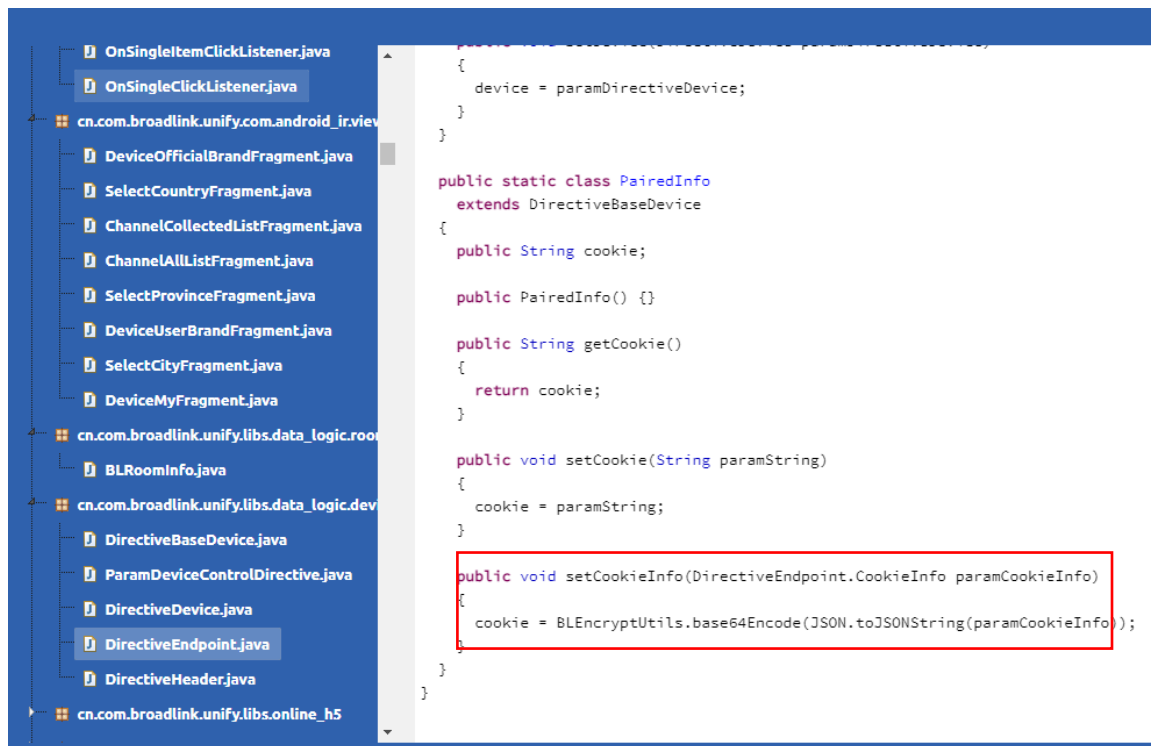
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public class HMacUtil
{
    public static final String PAGE_KEY = "awcn.HMacUtil";

    public HMacUtil() {}

    public static byte[] hmacSha1(byte[] paramArrayOfByte1, byte[] paramArrayOfByte2)
    {
        paramArrayOfByte1 = new SecretKeySpec(paramArrayOfByte1, "HmacSHA256");
        try
        {
            Mac localMac = Mac.getInstance("HmacSHA256");
            localMac.init(paramArrayOfByte1);
            paramArrayOfByte1 = localMac.doFinal(paramArrayOfByte2);
            return paramArrayOfByte1;
        }
        catch (NoSuchAlgorithmException paramArrayOfByte1)
        {
            for (;;) {}
        }
        catch (InvalidKeyException paramArrayOfByte1)
        {
            for (;;) {}
        }
    }
}
```

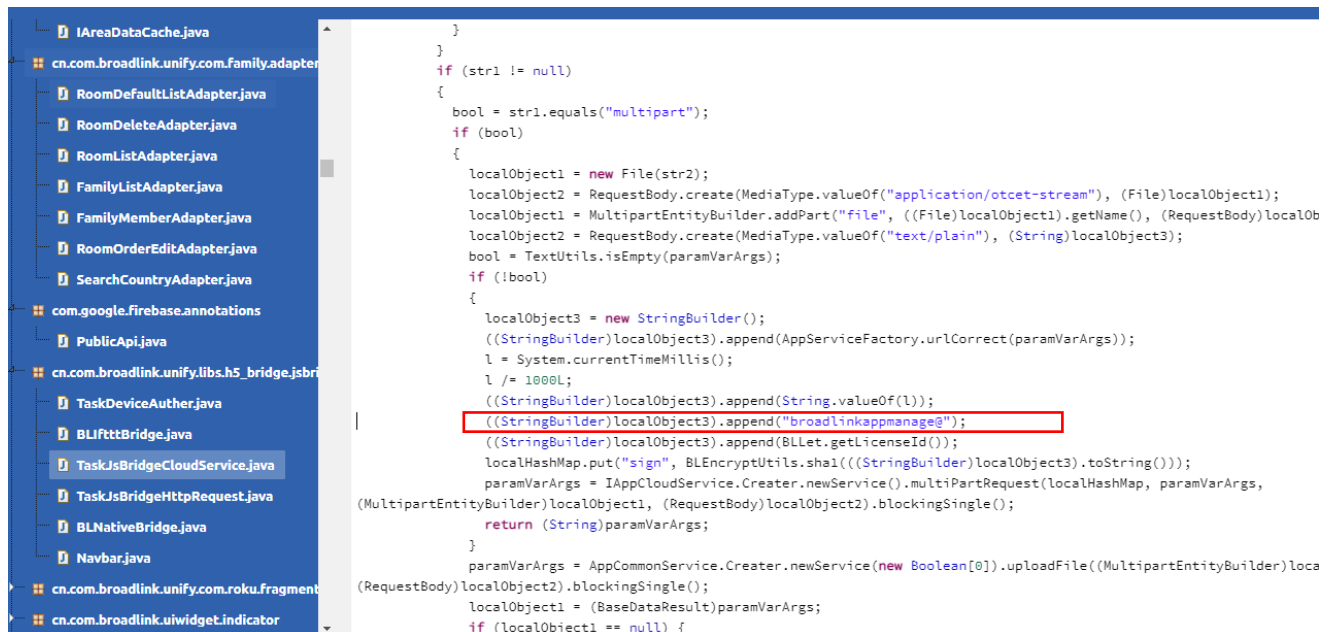

Tanto en la siguiente imagen como en la anterior se pueden apreciar dos métodos de cifrado SHA256 y base64 para algunas de sus conexiones



En esta imagen vienen predefinidas muchas direcciones , pero no tiene pinta de que debieran ser secretas ya que son enlaces a la tienda de la app o a la ayuda de Google




Aquí que es la parte de la nube , parece que tiene ese “usuario” para conectarse



```
    }  
    }  
    if (str1 != null)  
    {  
        bool = str1.equals("multipart");  
        if (bool)  
        {  
            localObject1 = new File(str2);  
            localObject2 = RequestBody.create(MediaType.valueOf("application/octet-stream"), (File) localObject1);  
            localObject1 = MultipartEntityBuilder.addPart("file", ((File) localObject1).getName(), (RequestBody) localObject2);  
            localObject2 = RequestBody.create(MediaType.valueOf("text/plain"), (String) localObject3);  
            bool = TextUtils.isEmpty(paramVarArgs);  
            if (!bool)  
            {  
                localObject3 = new StringBuilder();  
                ((StringBuilder) localObject3).append(AppServiceFactory.urlCorrect(paramVarArgs));  
                l = System.currentTimeMillis();  
                l /= 1000L;  
                ((StringBuilder) localObject3).append(String.valueOf(l));  
                ((StringBuilder) localObject3).append("broadlinkappmanage@");  
                ((StringBuilder) localObject3).append(BLLet.getLicenseId());  
                localHashMap.put("sign", BLEncryptUtils.shal1(((StringBuilder) localObject3).toString()));  
                paramVarArgs = IAppCloudService.Creator.newService().multipartRequest(localHashMap, paramVarArgs,  
                    (MultipartEntityBuilder) localObject1, (RequestBody) localObject2).blockingSingle();  
                return (String) paramVarArgs;  
            }  
            paramVarArgs = AppCommonService.Creator.newService(new Boolean[0]).uploadFile((MultipartEntityBuilder) localObject1,  
                (RequestBody) localObject2).blockingSingle();  
            localObject1 = (BaseDataResult) paramVarArgs;  
            if (localObject1 == null) {  
                }  
            }  
        }  
    }  
}
```

Aquí tenemos el token para la nube



```
import k.MediaType;  
import k.MultipartEntityBuilder;  
import k.RequestBody;  
import k.ResponseBody;  
import org.apache.cordova.CallbackContext;  
import org.json.JSONException;  
import org.json.JSONObject;  
  
public class TaskJsBridgeCloudService  
    extends AsyncTask<Void, Void, String>  
    {  
        public static final String FAMILY_TOKEN_KEY = "xgx3d+fe3478$ukx";  
        public static final String SERVICE_DATA = "dataservice";  
        public static final String SERVICE_IR = "irservice";  
        public static final String SERVICE_PRODUCT = "productservice";  
        public CallbackContext callbackContext;  
        public String cmdJsonStr;  
  
        public TaskJsBridgeCloudService(String paramString, CallbackContext paramCallbackContext)  
        {  
            cmdJsonStr = paramString;  
            callbackContext = paramCallbackContext;  
        }  
  
        private Map generateHead(String paramString1, String paramString2)  
        {  
            BaseHttpHeader localBaseHttpHeader = AppServiceFactory.getBaseHttpHeader();  
            paramString1 = h.tuxmobil.fahrplan.congress.StringBuilder.append(paramString1, "xgx3d+fe3478$ukx", paramString2);  
            paramString1.append(LocalBaseHttpHeader.getUserId());  
            paramString1 = paramString1.toString();  
        }  
    }  
}
```


Y en esta parte donde parece que es la recolección de datos el programa para des ofuscar el código produce esta salida, no se si es n error o que el código en esta parte esta protegido de alguna manera, pero este mensaje me lo he encontrado varias veces en otros archivos que tenían pinta de importantes



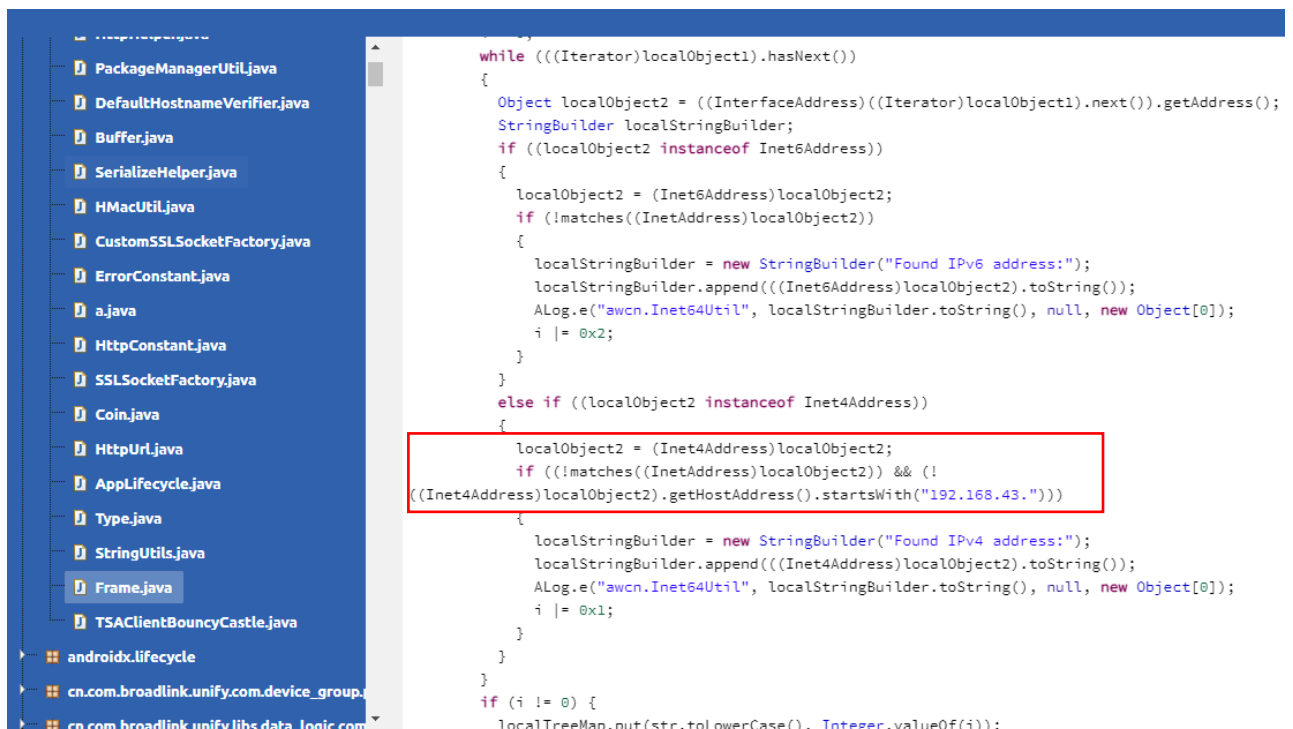
```
import cn.com.broadlink.sdkplug.sdkplug;
import java.util.Map;

public class BLDataPicker
{
    public static final String ID_DATA_UPLOAD = "10003";
    public static final String ID_H5_UPLOAD = "10004";
    public static final String KEY_CATEGORY_ID = "categoryId";
    public static final String KEY_EVENT_ID = "eventId";
    public static final String KEY_PLATFORM = "platform";
    public static final String PLATFORM_ANDROID = "android";
    public static final String TAG_DATA_UPLOAD = "app data";
    public static final String TAG_H5_UPLOAD = "h5 data";

    public BLDataPicker() {}

    public static void appEvent(int paramInt1, int paramInt2, Map paramMap)
    {
        throw new RuntimeException("d2j fail translate: java.lang.RuntimeException: fail exe a4 = a3\n\tat com.googlecode.dex2jar.ir.ts.an.BaseAnalyze.exec(BaseAnalyze.java:92)\n\tat com.googlecode.dex2jar.ir.ts.an.BaseAnalyze.exec(BaseAnalyze.java:1)\n\tat com.googlecode.dex2jar.ir.ts.Cf com.googlecode.dex2jar.ir.ts.an.BaseAnalyze.analyze0(BaseAnalyze.java:75)\n\tat com.googlecode.dex2jar.ir.ts.an.BaseAnalyze.analyze(BaseAnalyze.java:69)\n\tat com.googlecode.dex2jar.ir.ts.UnSSATransformer.transform(UnSSATransformer.java:274)\n\tat com.googlecode.d2j.dex.Dex2jar$2.optimize(Dex2jar.java:163)\n\tat com.googlecode.d2j.dex.Dex2Asm.convertCode(Dex2Asm.java:414)\n\tat com.googlecode.d2j.dex.ExDex2Asm.convertCode(ExDex2Asm.java:42)\n\tat com.googlecode.d2j.dex.Dex2jar$2.convertCode(Dex2jar.java:128)\n\tat com.googlecode.d2j.dex.Dex2Asm.convertMethod(Dex2Asm.java:509)\n\tat com.googlecode.d2j.dex.Dex2Asm.convertClass(Dex2Asm.java:406)\n\tat com.googlecode.d2j.dex.Dex2Asm.convertDex(Dex2Asm.java:423)\n\tat
```

Aquí por último esta parte nos indica que por defecto la ip va a empezar con :



```
while (((Iterator)localObject1).hasNext())
{
    Object localObject2 = ((InterfaceAddress)((Iterator)localObject1).next()).getAddress();
    StringBuilder localStringBuilder;
    if ((localObject2 instanceof Inet6Address))
    {
        localObject2 = (Inet6Address)localObject2;
        if (!matches((InetAddress)localObject2))
        {
            localStringBuilder = new StringBuilder("Found IPv6 address:");
            localStringBuilder.append(((Inet6Address)localObject2).toString());
            ALog.e("awcn.Inet64Util", localStringBuilder.toString(), null, new Object[0]);
            i |= 0x2;
        }
    }
    else if ((localObject2 instanceof Inet4Address))
    {
        localObject2 = (Inet4Address)localObject2;
        if (!matches((InetAddress)localObject2)) && (!(
            ((Inet4Address)localObject2).getHostAddress().startsWith("192.168.43.")))
        {
            localStringBuilder = new StringBuilder("Found IPv4 address:");
            localStringBuilder.append(((Inet4Address)localObject2).toString());
            ALog.e("awcn.Inet64Util", localStringBuilder.toString(), null, new Object[0]);
            i |= 0x1;
        }
    }
}
if (i != 0) {
    localTreeMap.put(str.toLowerCase(), Integer.valueOf(i));
}
```

- Problemas que se han presentado durante la investigación

-No tener dispositivos inteligentes ni un Alexa o Google home para conectarlo al RM y ver los paquetes que información manda cuando se conecta a ellos, cuando pido algo desde Alexa hacia el RM, etc. y así haber podido utilizar propiamente la librería

broadlinkhacktools.

- Conclusiones de la investigación, indicando el impacto y posible trabajo futuro

-En mi opinión este dispositivo esta bastante securizado no tanto por la seguridad en sí, si no, más por la simpleza de sus funciones.

-El dispositivo no emite señal para conectarse a el salvo cuando no ha sido conectado a ningún dispositivo o cuando el usuario a sabiendas marca la opción de compartir dispositivo para que otros usuarios puedan localizarlo, así que es bastante difícil tomar el control del dispositivo de forma remota.

-La aplicación es bastante grande y por lo que he podido observar también es bastante segura , tiene todas las direcciones ofuscadas o si aparecen a simple vista es porque no deben ser secretas.

-El dispositivo solo reproduce señales IR , se podría guardar la señal de apagado de varias televisiones y apagarlas todas como ataque por ejemplo.

- El posible trabajo futuro seria realizar un análisis de los paquetes para ver qué información manda y recibe el RM con otros dispositivos inteligentes, para ver si transmite la Ip de alguno y probar a cambiar la Ip de esa orden por la de otro dispositivo y ver si acepta ese paquete o que medidas de seguridad implementa a ese nivel.