



Security in IoT Ecosystem

Module 7

Smart Socket Pentest Part IV

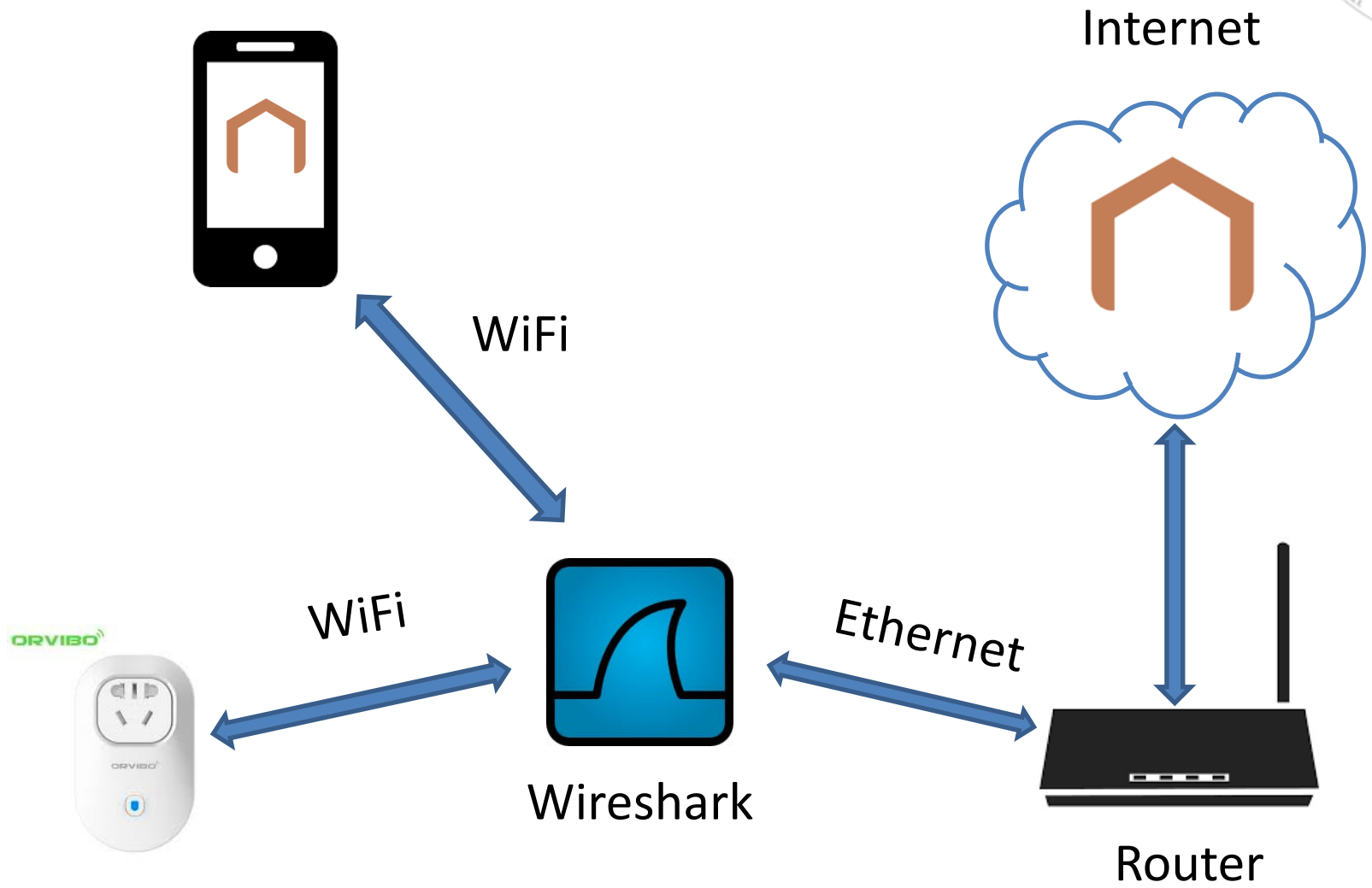
Prof.: Joaquín Recas

Smart Socket Initial Setup



1. Initial Setup
2. Capture Traffic
3. Traffic analysis
4. Apk Analysis
5. Decrypt traffic
6. Take control of the device (MiM)
7. Pentest Project tasks

Study communications:



capture2.pcapng

ip.addr == 18.197.83.128

No.	Time	Source	Destination	Protocol	Length	Info
84	14.537238127	192.168.1.91	18.197.83.128	TCP	60	58383 → 10001 [SYN] Seq=0 Win=5840 Len=0 MSS=1440
85	14.578162605	18.197.83.128	192.168.1.91	TCP	58	10001 → 58383 [SYN, ACK] Seq=0 Ack=1 Win=26883 Len=0 M
88	14.600741931	192.168.1.91	18.197.83.128	TCP	60	58383 → 10001 [ACK] Seq=1 Ack=1 Win=5840 Len=0
92	15.608756126	192.168.1.91	18.197.83.128	TCP	240	58383 → 10001 [PSH, ACK] Seq=1 Ack=1 Win=5840 Len=186
93	15.647162899	18.197.83.128	192.168.1.91	TCP	54	10001 → 58383 [ACK] Seq=1 Ack=187 Win=27872 Len=0
94	15.647189972	18.197.83.128	192.168.1.91	TCP	160	10001 → 58383 [PSH, ACK] Seq=1 Ack=187 Win=27872 Len=1
95	15.742301397	192.168.1.91	18.197.83.128	TCP	60	58383 → 10001 [ACK] Seq=187 Ack=107 Win=5734 Len=0
96	15.757494085	192.168.1.91	18.197.83.128	TCP	256	58383 → 10001 [PSH, ACK] Seq=187 Ack=107 Win=5734 Len=
97	15.807547076	18.197.83.128	192.168.1.91	TCP	128	10001 → 58383 [PSH, ACK] Seq=107 Ack=389 Win=28944 Len=
98	15.947027090	192.168.1.91	18.197.83.128	TCP	60	58383 → 10001 [ACK] Seq=389 Ack=181 Win=5660 Len=0
99	16.013377910	192.168.1.91	18.197.83.128	TCP	240	58383 → 10001 [PSH, ACK] Seq=389 Ack=181 Win=5660 Len=
100	16.060055042	18.197.83.128	192.168.1.91	TCP	272	10001 → 58383 [PSH, ACK] Seq=181 Ack=575 Win=30016 Len=
101	16.186322718	192.168.1.91	18.197.83.128	TCP	60	58383 → 10001 [ACK] Seq=575 Ack=399 Win=5442 Len=0
121	18.861219916	192.168.1.91	18.197.83.128	TCP	192	58383 → 10001 [PSH, ACK] Seq=575 Ack=399 Win=5442 Len=
122	18.898070681	18.197.83.128	192.168.1.91	TCP		

▶ Frame 92: 240 bytes on wire (1920 bits), 240 bytes captured (1920 b
 ▶ Ethernet II, Src: Imaginat_89:16:5c (00:19:f5:89:16:5c), Dst: Crisp
 ▶ Internet Protocol Version 4, Src: 192.168.1.91, Dst: 18.197.83.128
 ▶ Transmission Control Protocol, Src Port: 58383, Dst Port: 10001, Se
 ▼ Data (186 bytes)
 Data: 686400ba706b53545ec900000000000000000000000000000000...
 [Length: 186]

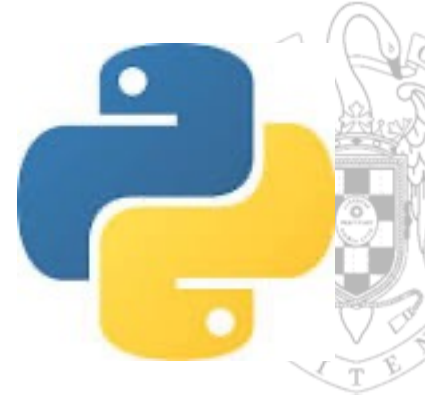
```

0000 00 00 2b 3c 0c 6c 00 19 f5 89 16 5c 08 00 45 00  ...+<.l...
0010 00 e2 00 0a 00 00 fe 06 93 c3 c0 a8 01 5b 12 c5  ...o...h...pkST^
0020 53 80 e4 0f 27 11 00 00 19 6e f9 d1 a5 8c 50 18  S...h...P...
0030 16 d0 6f 9b 00 00 68 64 00 ba 70 6b 53 54 5e c9  ...o...hd...pkST^
0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...o...hd...pkST^
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ...o...hd...pkST^
0060 2e 8d fc 89 e7 bc 96 40 cb 68 a1 b7 08 aa 0f 65  ...o...hd...pkST^
0070 6a 19 6f 73 a5 54 e2 ee 46 e0 a5 0b 64 d8 86 cd  j...os...T...F...d...
0080 5b 1a 87 c4 1f 66 71 37 62 68 ec 5b 66 0d b2 c5  [...]fq7 bh.[f...
0090 88 6a 05 67 13 e0 d9 bc 4a 0c 2c 50 58 df 65 2d  .j.g... J.,PX.e-
00a0 d9 b1 5b d1 7b 27 05 cd 39 fc fd 62 ae 76 c6 e2  .[.{'... 9..b.v...
00b0 61 33 54 79 10 ab 16 98 fb 16 59 d7 0f a1 2b 97  a3Ty...Y...+...
00c0 ff 0b 7c 9f c7 e0 72 d7 fb 81 ac 8b f7 9a 52 bb  .|.r...R...
00d0 88 f0 2f d8 4d 7d 6a 77 4c b5 b8 6d d6 a5 f9 ba  ./.M}jw L.m...
00e0 59 bb 84 e8 7f 2f 0a e4 d1 c3 e9 1c a3 ae 37 93  Y.../...7...
  
```

Encrypted data
 Info from APK:

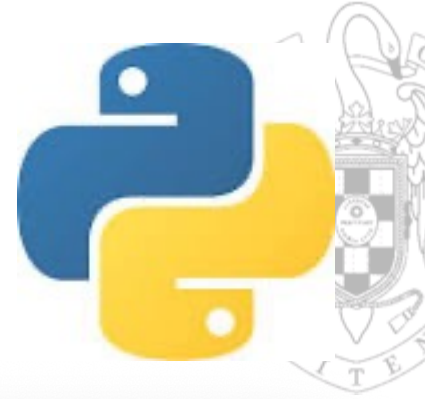
- method AES-ECB?
- Key `khggd54865SNJHGF`?

Python Crypto



- Install crypto library:
`$ sudo pip install pycrypto`
- Select packages that go to/from Orvibo server and have payload
- Copy the payload from WireShark
- Decrypt using AES-ECB + key

Python Crypto



```
decrypt_AES_ECB.py UNREGISTERED
decrypt_AES_ECB.py
1  #!/usr/bin/python3
2
3  from Crypto.Cipher import AES
4  from binascii import unhexlify
5
6  aeskey="KEY"
7
8  wiresharkpacket = unhexlify("WireShark_Data_as_HexStream")
9
10 aesobj = AES.new(aeskey, AES.MODE_ECB)
11 print(aesobj.decrypt(wiresharkpacket[42:]))
```

Line 1, Column 1 Tab Size: 4 Python



We decrypt data

```

ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py 44
Trace 44
{
  "cmd": 0,
  "serial": 1,
  "modelId": "98431e8c9e834776b414a40b2d8ddefb",
  "softwareVersion": "v3.0.9",
  "hardwareVersion": "v1.0.0",
  "mac": "c45bbe61687a",
  "language": "chinese"
}
ubuntu@ubuntu2004:~/Security$

```

Does not work with trace 48 🥲

```

ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py 48
Trace 48
Not a json: b'\xe0\x07\x8f3\x8e~(F\xb0\x05\xb5\xa8\x01i4\xa7\xb6\x03|\xff\xac\x
0f\x94\xa1zNq\x84\xa8~\xa8^-\xa5\xc0X\xb4\x99B]\x03\x11\xc4]7\xc5X\xd3,S\x04\x07
\xfe!\xee|\xc8k\xba\x96\xbbF\x97\x11\xd4Y\xf0\x98\xa6\x1bq\xec&\xfe\x8a\xc4\xd3
&7\x9d\xa4\xc3=\xf9\tG\xb4\xcfM\x1e)\xeb\x1f\xc2\x92\xbefQ\xb3bY\eb`\x04@8_\xed
\xee\xd08\x910\x04\x8c\x8d\x02\|\$"\xe1\xed\t9\xe5\x94\xe7\xaas\xfe\xaf3g# V\xfe\
xef\x869\xb8\xe3g\x7f\x95j\xc1\xd4m\xf2\xe0\x1f\xf74\xa2\xd7,\x9a\n\x98\xa6v\x98
=.\xef\xc3\xf7n*\x84\x85\x97\xa3h\xde\xa1\x90\xc7AB\xc7!\x14\xdak\xd9#\xe0\x88'
ubuntu@ubuntu2004:~/Security$

```




Find new AES-ECB key

```
ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py 46
Trace 46
{
  "serial": 1,
  "cmd": 0,
  "key": "97l12BT6E9tL46XU",
  "status": 0
}
```

...and we're back in business!!! 😊

```
ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py -key 97l12BT6E9tL46XU 48
Trace 48
{
  "cmd": 6,
  "serial": 2,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "password": "CAD2438046ABD12E7961C6096E280EC6",
  "localIp": "10.42.0.243",
  "localPort": 8000,
  "rst_reason": "DEFAULT_RST"
}
```

Status info sent to Orvibo



```
ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py -key 97l12BT6E9tL46XU 49 57
Trace 49
{
  "rst_reason": "DEFAULT_RST",
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "password": "CAD2438046ABD12E7961C6096E280EC6",
  "localPort": 8000,
  "serial": 2,
  "localIp": "10.42.0.243",
  "cmd": 6,
  "status": 0
}

Trace 57
{
  "cmd": 42,
  "serial": 3,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "deviceId": "0",
  "statusType": 0,
  "value1": 1,
  "value2": 0,
  "value3": 0,
  "value4": 0,
  "alarmType": 1
}
```



```
ubuntu@ubuntu2004: ~/Security
ubuntu@ubuntu2004:~/Security$ ./decrypt_aes_ecb.py -key 97l12BT6E9tL46XU 85 89
Trace 85
{
  "ver": "3.7.0",
  "value2": 0,
  "value1": 0,
  "value4": 0,
  "fromMq": true,
  "value3": 0,
  "debugInfo": "Android_ZhiJia365_32_3.7.0.308",
  "userName": "cuenta.tfn@gmail.com",
  "deviceId": "e65089fff2144904b2e9462c342c04f2",
  "defaultResponse": 1,
  "respByAcc": false,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "clientId": "7b08582fc32d4b5f92bdd30f233ff334",
  "propertyResponse": 0,
  "serial": 348894220,
  "delayTime": 0,
  "cmd": 15,
  "qualityOfService": 1,
  "order": "on"
}
Trace 89
{
  "cmd": 15,
  "serial": 348894220,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "clientId": "7b08582fc32d4b5f92bdd30f233ff334",
  "status": 0
}
```

Trace 85: 'Order On' sent by the server

Trace 89: S30c status reply



```
ubuntu@ubuntu2004: ~/Security
./decrypt_aes_ecb.py -key 97l12BT6E9tL46XU 112 114
Trace 112
{
  "ver": "3.7.0",
  "value2": 0,
  "value1": 1,
  "value4": 0,
  "fromMq": true,
  "value3": 0,
  "debugInfo": "Android_ZhiJia365_32_3.7.0.308",
  "userName": "cuenta.tfn@gmail.com",
  "deviceId": "e65089fff2144904b2e9462c342c04f2",
  "defaultResponse": 1,
  "respByAcc": false,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "clientId": "7b08582fc32d4b5f92bdd30f233ff334",
  "propertyResponse": 0,
  "serial": 360201722,
  "delayTime": 0,
  "cmd": 15,
  "qualityOfService": 1,
  "order": "off"
}
Trace 114
{
  "cmd": 15,
  "serial": 360201722,
  "uid": "d7444ec7b8524af591eea269ecb100b7",
  "clientId": "7b08582fc32d4b5f92bdd30f233ff334",
  "status": 0
}
```

Trace 112: 'Order Off' sent by the server

Trace 114: S30c status reply

Information on operation



- On and off:
 - Android app request server to turn on/off plug
 - The server communicates with the socket:
 - "order":"on"
 - "order":"off"
 - The socket responds with a status message
- ...in addition, the device periodically sends status information to the server, among other things...

Smart Socket Initial Setup



1. Initial Setup
2. Capture Traffic
3. Traffic analysis
4. Apk Analysis
5. Decrypt traffic
6. Take control of the device (MiM)
7. Pentest Project tasks

Can we take control of the device?

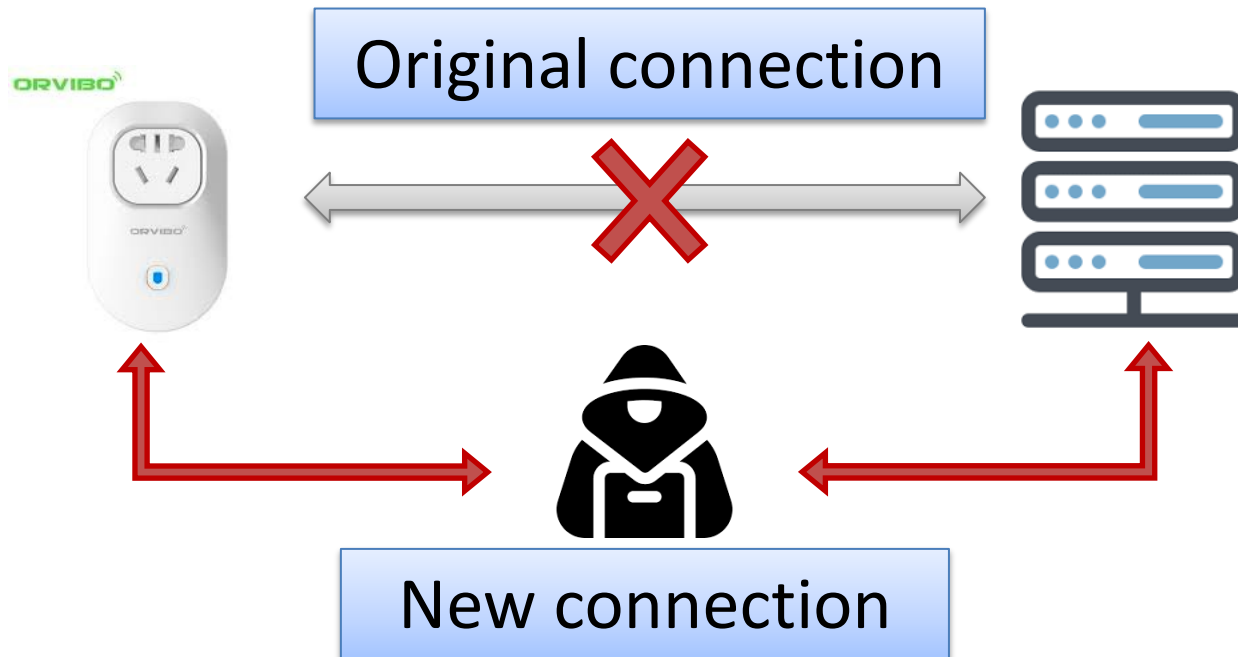


- We know how to encrypt/decrypt messages (symmetric encryption)
- We know the format of the on/off messages
- We know IPs and ports used by device and server
- We can do a MiTM (Man in the Middle) type attack



Man in the Middle

- Messages from the server (and their responses) reach the device by passing through the proxy.



- The proxy can inject new messages to the device, responses to these messages should be discarded.



Proxy *orvibocontrol.py*

1. Unplug the Smarsocket
2. Configure the system to redirect traffic from the device to the server through port 8080 and launch `orvibocontrol.py`:

```
iot@raspy-iot-da: ~  
File Edit View Search Terminal Help  
iot@raspy-iot-da:~$ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp  
-d 18.197.83.128 --dport 10001 -j REDIRECT --to-port 8080  
[sudo] password for iot:  
iot@raspy-iot-da:~$ ./orvibocontrol.py  
[+] Main  
█
```



Proxy *orvibocontrol.py*

3. Plug the Smartsocket

```
iot@raspy-iot-da: ~
File Edit View Search Terminal Help
iot@raspy-iot-da:~$ sudo iptables -t nat -A PREROUTING -i wlan0 -p tcp -d 18.197.83.128 --dport 10001 -j REDIRECT --to-port 8080
[sudo] password for iot:
iot@raspy-iot-da:~$ ./orvibocontrol.py
[+] Main
[+] Client connected
[+] HTTP Server started on 5555

[+] socket -> server
{'hardwareVersion': u'v1.0.0', 'language': u'chinese', 'cmd': 0, 'softwareVersion': u'v3.0.9', 'mac': u'c45bbe61687a', 'serial': 1, 'modelId': u'98431e8c9e834776b414a40b2d8ddefb'}
[+] server -> socket
{'status': 0, 'serial': 1, 'cmd': 0, 'key': u'3MKe0y2YF0hgbJ1Q'}

[+] socket -> server
{'rst_reason': u'DEFAULT_RST', 'localPort': 8000, 'cmd': 6, 'localIp': u'10.42.0.243', 'serial': 2, 'password': u'CAD2438046ABD12E7961C6096E280EC6', 'uid': u'd7444ec7b8524af591eea269ecb100b7'}
[+] server -> socket
{'rst_reason': u'DEFAULT_RST', 'status': 0, 'uid': u'd7444ec7b8524af591eea269ecb100b7', 'cmd': 6, 'localIp': u'10.42.0.243', 'serial': 2, 'password': u'CAD2438046ABD12E7961C6096E280EC6', 'localPort': 8000}
```

Proxy *orvibocontrol.py*



4. Open Firefox in the Raspi on 127.0.0.1:5555



Smart Socket Initial Setup



1. Initial Setup
2. Capture Traffic
3. Traffic analysis
4. Apk Analysis
5. Decrypt traffic
6. Take control of the device (MiM)
7. Pentest Project tasks

Smart Socket Pentest Tasks



Create a document with the following points, for this use screenshots and comment all the steps:

1. Get the key and encryption method:
 1. Download [Home Mate Apk](#)
 2. Analyze it with [JADX](#) to find the decryptByte function
 3. Unzip the APK and locate the library where the message encryption function is located (rgrep decryptByte)
 4. Disassemble the library file using [Binary Ninja](#) and locate the encryption key and method

Smart Socket Pentest Tasks



2. Decrypt messages:

2. Record an On/off sequence trace with [WireShark](#)
3. Use the cipher key of Step 1 and decryptaes.py script to decrypt the first SmartPlug messages
4. Find the new cipher key
5. Decipher an on/off messages and the response from the SmartPlug

Smart Socket Pentest Tasks



3. Analyze and comment `orvibocontrol.py` proxy code:
 1. How many threads and queues are created?
 2. How many sockets are used? What's its purpose?
 3. Create a block diagram of the code
4. Comment possible measures to improve security of the Smartsocket

**This practice represents a total of 30% of the grade.
Send me the corresponding memory until June 3 (recas@ucm.es).**