Pwning Wi-Fi lightbulbs

LimitedResults
25th of May 2019
BSIDES Stuttgart

BootRom

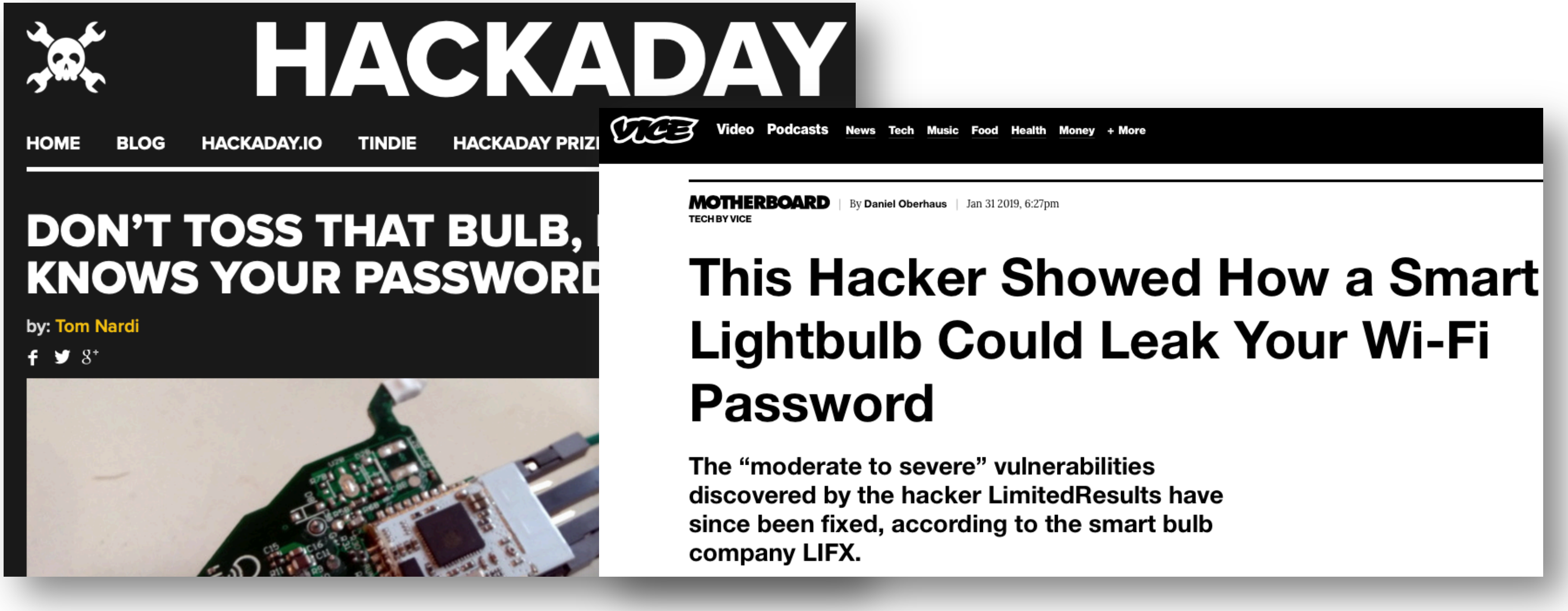
INTRODUCTION

$whoami

- I am **Limited**
  - By my time, my budget, my skills too…
  - Offensive side
- **Results are here**
  – www.LimitedResults.com started in Oct. 18
  – Random hacks
- I like to
  – Attack real products
  – Focus on hardware
- No Affiliation
- Raw slides maker ☺

About this talk

- Last January...media coverage



DON'T TOSS THAT BULB, I
KNOWS YOUR PASSWORD

by: Tom Nardi

MOTHERBOARD TECH BY VICE | By Daniel Oberhaus | Jan 31 2019, 6:27pm

This Hacker Showed How a Smart Lightbulb Could Leak Your Wi-Fi Password

The "moderate to severe" vulnerabilities discovered by the hacker LimitedResults have since been fixed, according to the smart bulb company LIFX.

¯_(ツ)_/¯

The Plan

- **Introduction**
  - Already Done!
- **The Security in IoT**
  - Security context
  - Vulnerabilities
  - Hardware hacking
- **The Lightbulbs**
  - Lightbulbs anatomy
  - Different lightbulbs?
  - Lightbulbs ecosystem
- **Security analysis**
  - Assets inside
  - Threats modeling
  - Hacker point-of-view
- **The Hacks**
  - Xiaomi Yeelight
  - LIFX Mini
  - WIZ connected
  - Tuya light
- **Discussions**
  - Synthesis
  - Limited Impact
  - Back to basics
  - My opinion
- **Conclusion**
  - conclusion

Bootloader

THE SECURITY IN IOT

Security in IoT?

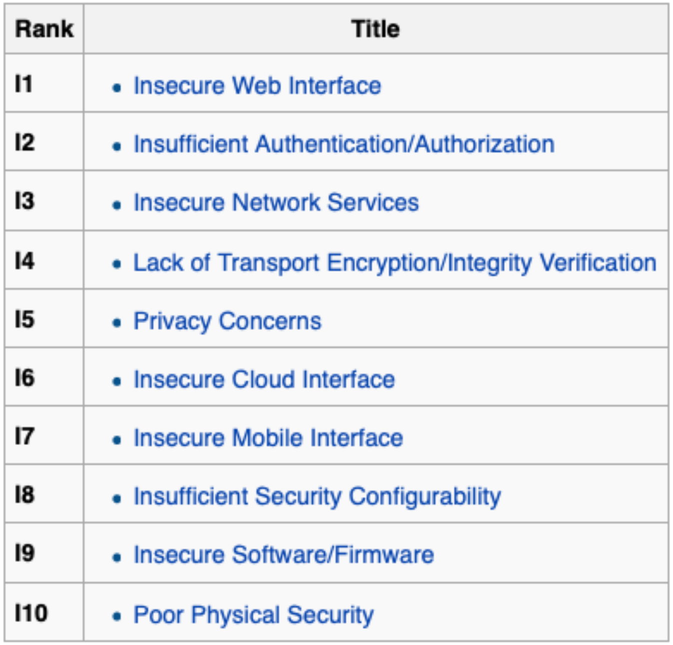
- What/where are the rules?
  - Guidelines?
  - Standards?
- Security
  - Not the priority of customers
  - Not the priority of vendors
- ONLY new features & costs are important
  - Select cheapest hardware
  - Reuse of code (as it is)
  - Wild outsourcing
  - Marketing budget
  - Go to market first
- **Fertile Ground for hackers**

NO RULES

Same vulns == Same problems

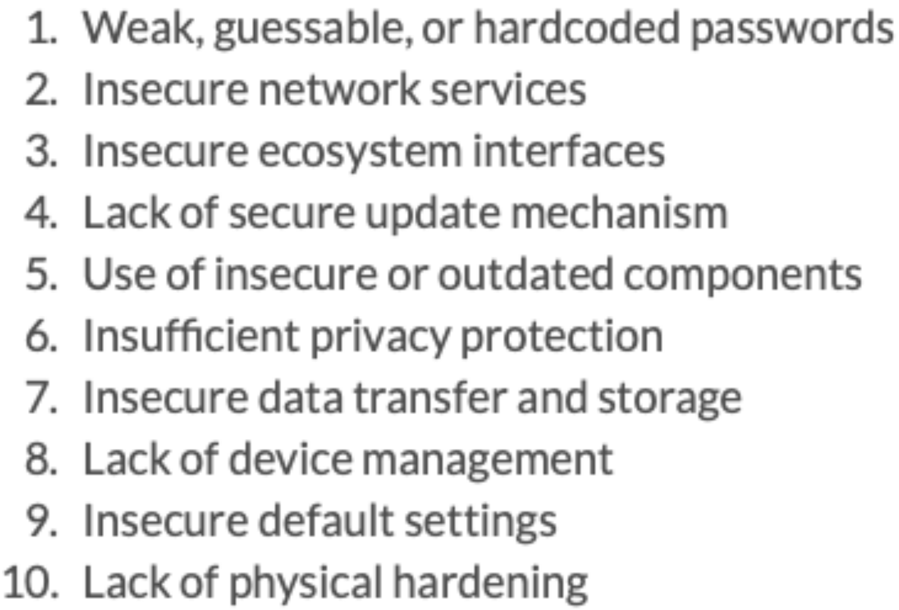
- Top 10 IoT Vulns are the same since 5 years
  - https://www.owasp.org/index.php/Top_IoT_Vulnerabilities

2014



Rank	Title
I1	• Insecure Web Interface
I2	• Insufficient Authentication/Authorization
I3	• Insecure Network Services
I4	• Lack of Transport Encryption/Integrity Verification
I5	• Privacy Concerns
I6	• Insecure Cloud Interface
I7	• Insecure Mobile Interface
I8	• Insufficient Security Configurability
I9	• Insecure Software/Firmware
I10	• Poor Physical Security

2018



1. Weak, guessable, or hardcoded passwords
2. Insecure network services
3. Insecure ecosystem interfaces
4. Lack of secure update mechanism
5. Use of insecure or outdated components
6. Insufficient privacy protection
7. Insecure data transfer and storage
8. Lack of device management
9. Insecure default settings
10. Lack of physical hardening

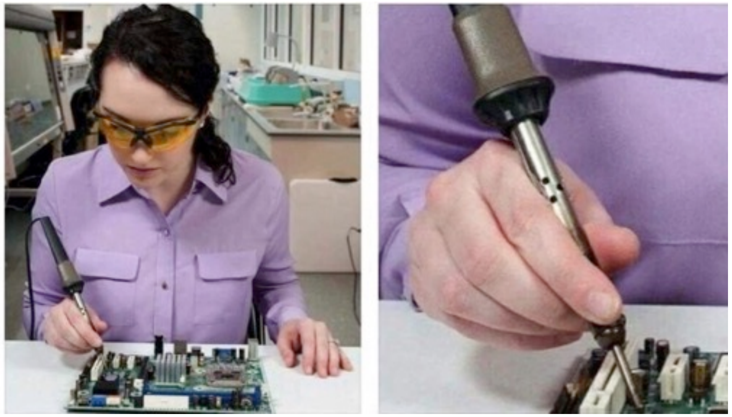
  - Terrible statement but it is the reality
  - Don't worry, that will continue…

In the Embedded Hacker's Mind

- The Strategy
  - Find the target
    - Valuable/bankable
    - Not too much secure
    - Not too much people on it
  - Reverse the target
    - Hard/Soft Reverse
    - FW extraction
  - Find the vulnerability/ies
    - Static/dynamic approach?
    - Pure Soft, or more hardware?
  - Exploit
    - PoCs, CVE, disclosure process….
    - Or Weaponization
  - Profit
    - Bug bounty
    - Ransom, resell..oops

Hardware Hacking

- Mistaken beliefs
  - You need Physical access!
    - Yes but depends on the attack scenario…
    - And Reverse begins by physical access
  - It’s expensive!
    - I would say 200$ to start
    - Is it such a big security barrier?
- True facts about HW Vulns
  - Difficult to patch
  - More products impacted
  - Sometimes really trivial
- So buy an iron solder!
  - Be careful, it is hot



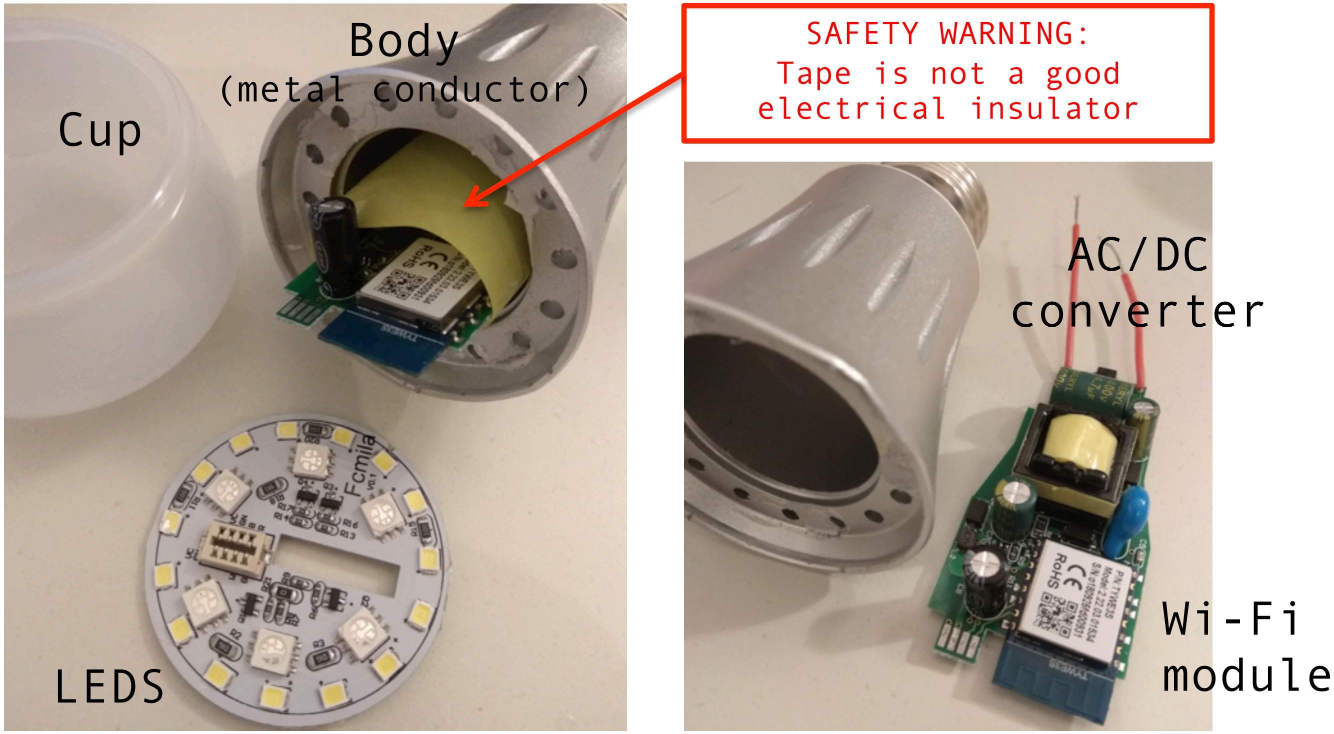
Just some HW tricks

- Hardware is cheap today(aliexpress, ebay…)
  - A lot of equipment… but also a lot of targets!
  - Quality is not always as expected. Be careful
- Some random tips
  - Find his own methodology
  - Practice
  - Keep practicing
  - Open source is the key
    - Flashrom, Gdb, openOCD, Binwalk, GHIDRA!
    - A lot of github repos!
    - More and more HW projects too!
    - FPGAs are nice
- Hardware hacking is not expensive and effective.

Booting…

THE LIGHTBULBS

Lightbulb Anatomy



SAFETY WARNING:
Tape is not a good electrical insulator

Teardown of a (random) bulb, 10 euros on Aliexpress

- Focus on the Wi-Fi module
  - The brain of the device

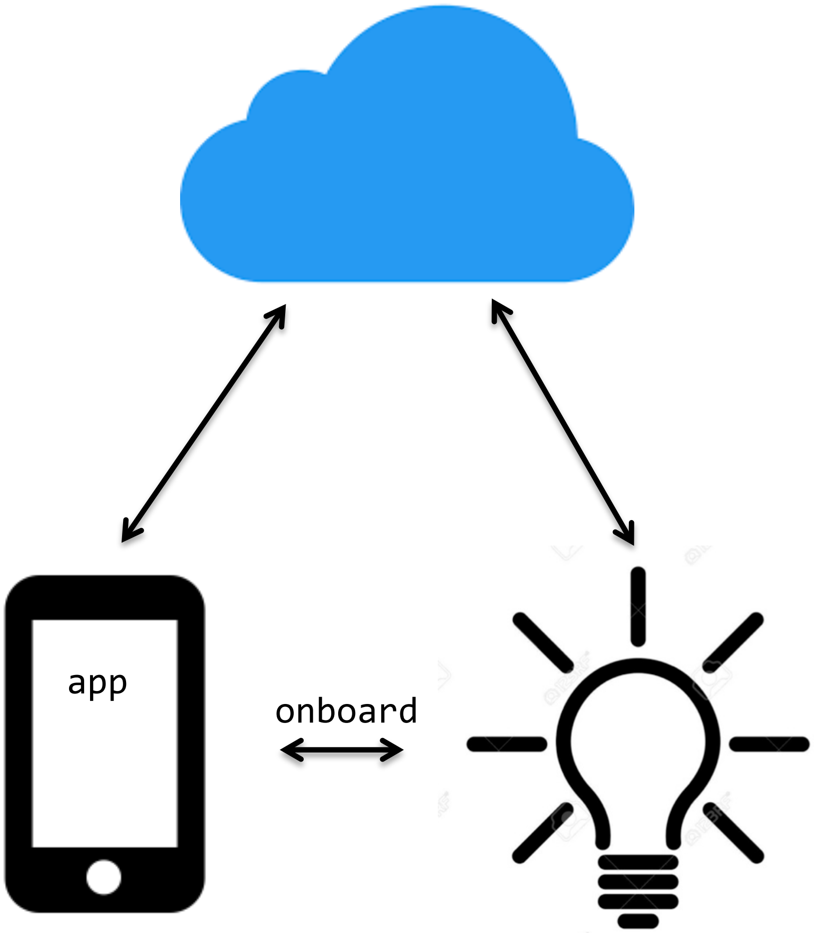
Different lightbulbs?

- BLE, Wi-FI or Hub (zigbee) Lightbulbs ????
- Same design, only the chip is different
- BLE lightbulbs have been hacked A LOT:
  - CSR dongle (5e)
  - your android phone (HCI snoop log in /sdcard/
  - Wireshark to analyze the BLE frames
  - BtLEJuice to act as MITM and replay packets
  - Hcitool, gatool… very simple to send packets
  - *Incomplete list, sorry*

- The talk focus on Wi-Fi light bulbs. Why?
  - Less explored
  - BLE devices don't have Cloud
  - BLE devices don't stand on Wi-Fi network

Wi-Fi Lightbulb ecosystem

- The Actors
  - The device (HW +FW)
  - The cloud
  - The app
- User WiFi is required
  - Use of SSL
  - Use of WPA2
- Onboarding protocol
  - Phone <-> Device
  - First time you connect
  - Password sharing
  - Proprietary protocols
    - Like Smartconfig
    - Use of BLE sometimes…



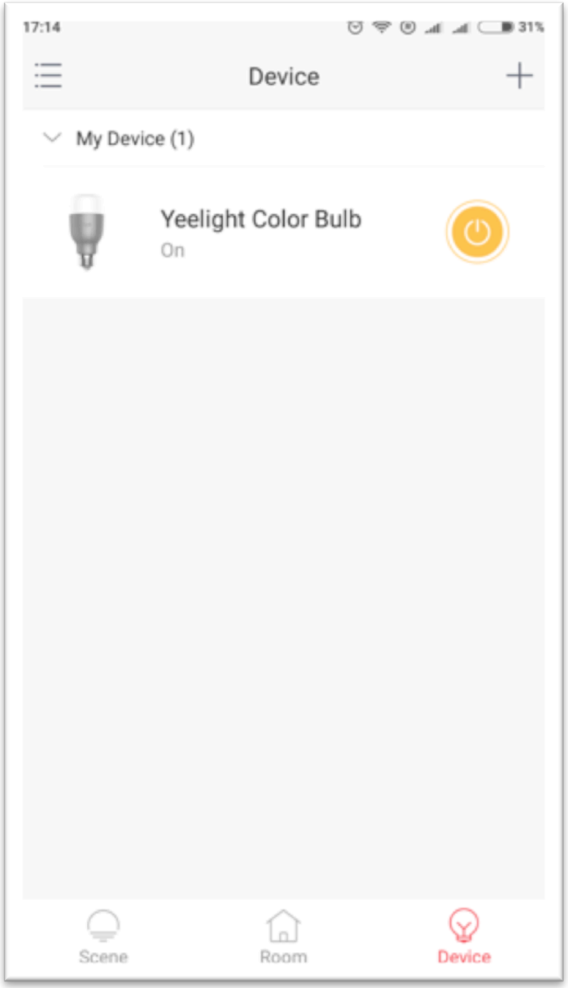
Sorry for this diagram :-/

Still booting…

SECURITY ANALYSIS

Why Hacking Wi-Fi bulbs?

- More interesting than the BLE sister
- Wi-Fi bulbs are:
  - Popular
  - Cheap (20-40)
  - Such an innocent but…sneaky device!
- Easy setup
- Easy to understand for people
  - I mean… It’s a bulb!
  - You can control it via an app
  - Good to ‘educate’ people



17:14 31%

Device

My Device (1)

Yeelight Color Bulb
On

Assets

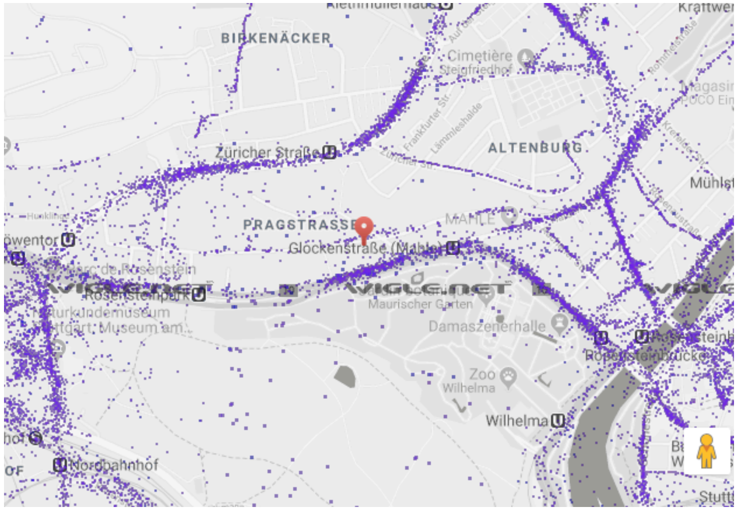
- What are the critical assets ~~to protect~~ to hack?
  - User account
    - Your Lifx account, Your Xiaomi account…
  - Authentication key, device ID to the Cloud
    - Used for Onboarding, MQTT protocol…
  - Wi-Fi credentials
    - SSID and WPA2 key
  - Company IPs
    - Hardware and software reverse, protect cloning?
  - User Data
    - Private DataBase, GDPR?

Threats modeling

- Main threats
  - Control other people's lamps
    - Access to Users accounts or to Cloud authentication keys
  - Retrieve Data/Cloud database
    - Big data leak is never good…
  - Access the User Network
    - Wi-Fi credentials
- Wi-Fi Light bulbs have a design weakness
  - The Wi-Fi credentials have to be inside the end-node device
- The vendor threat model doesn't take in consideration the ENTIRE product life cycle.
  - Development > Production > On the Field > **Garbage**
  - Physical access rated as '**Out of Scope**'.

Attacker point of view

- I decide to focus on the device(Hw+Fw)
- New threat is attacking from/ Into the Garbage
  - Physical access? Not a problem here
    - Contact inside waste recycling companies
    - Buy second hand devices
    - Just steal devices
  - Imagine how much devices you can get...
  - https://wigle.net
    - Perfect to localize networks
- **Let's go hack these devices!**



OS is running

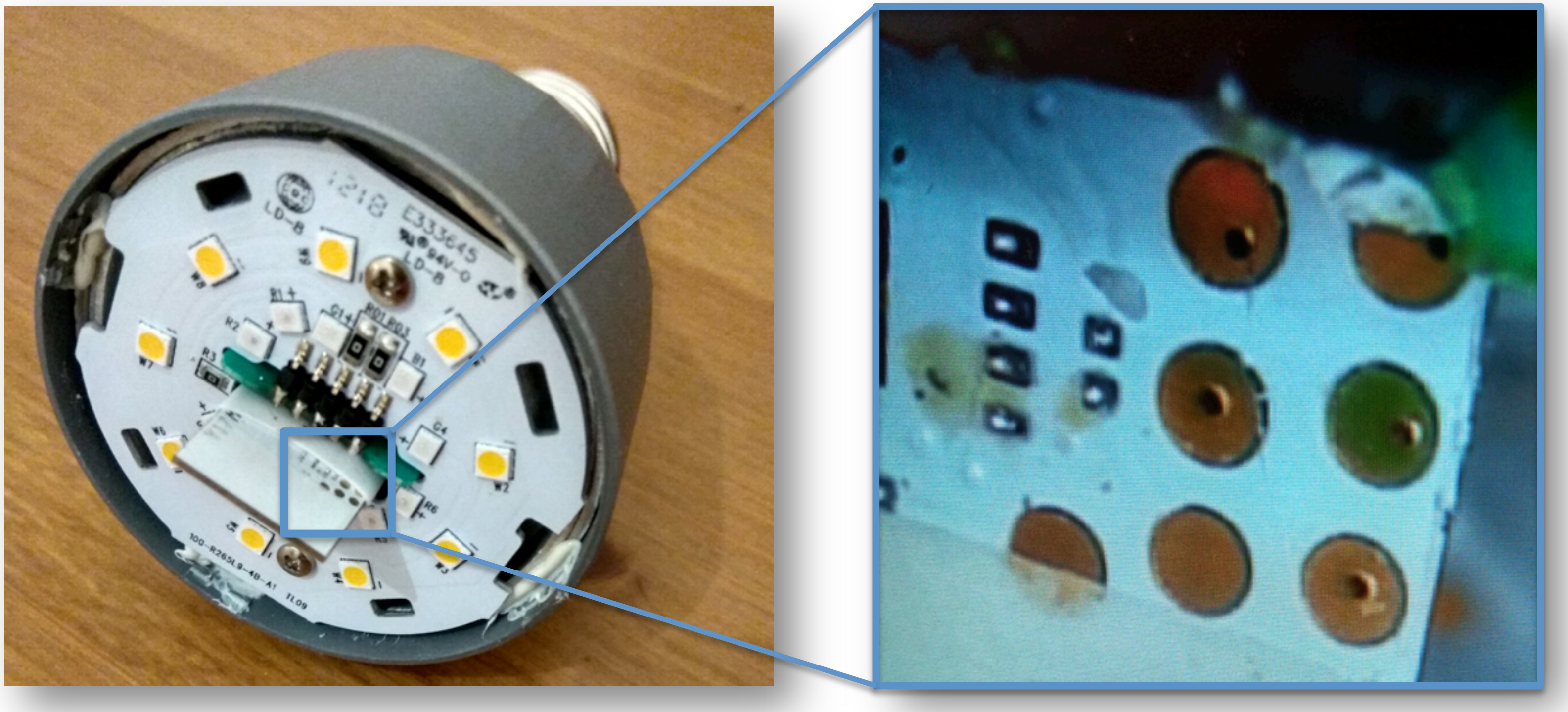
THE HACKS

Pwn n*1: Xiaomi

- The *‘super’* IoT company
  - They sell everything
    - Mobile phones, toothbrush, e-bikes…
  - They have a big Cloud
    - Yes, really big…
- A golden mine of devices…
- And Vulns
  - No Bug Bounty for European guys :-/
- Now:
  - Focus on the Xiaomi Yeelight
    - 20 euros on Amazon

The Xiaomi Yeelight

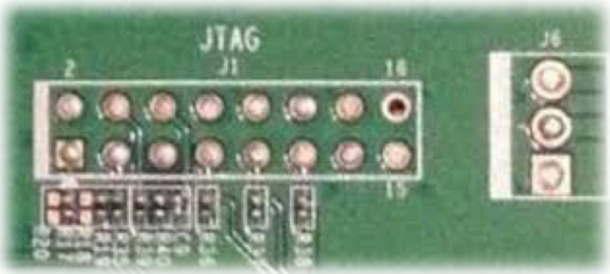
- After the device is configured, the cup is removed:



- Flying probes used during the production
  - Five marked tests points, coincidence?

5 pins? JTAG ☺

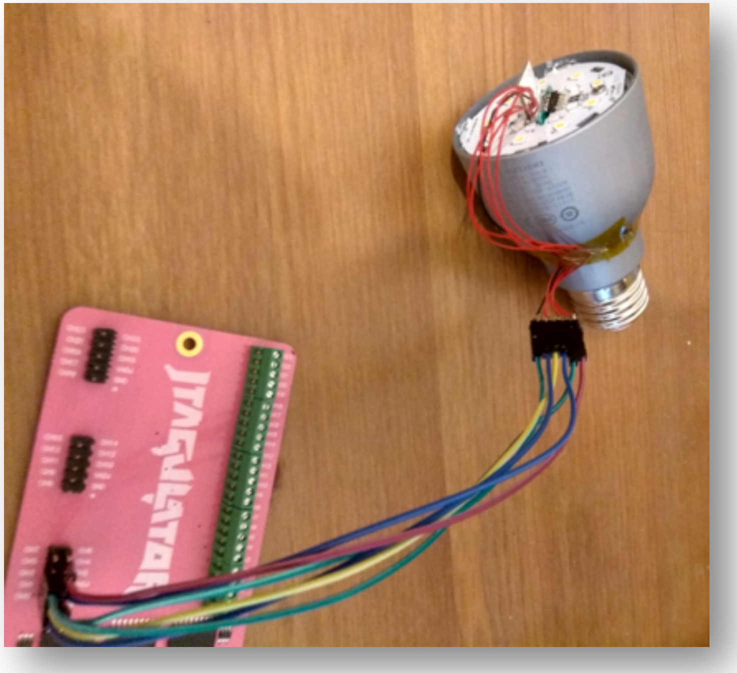
- JTAG = Join Test Action Group
  - Standardized debug interface
  - Verifying design, flash firmware and test during production
  - 5 signals => TDI,TDO,TCK, TMS, TRST (optional) + GND!
- This interface are generally easily guessed

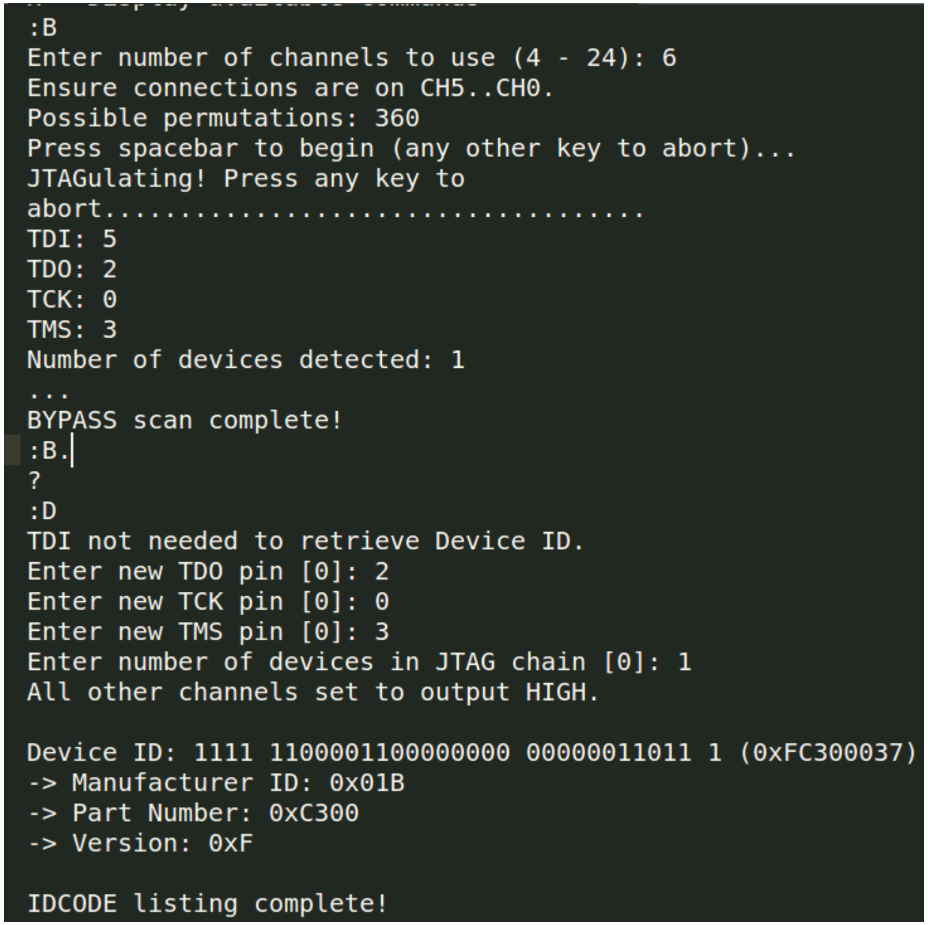


- SWD = Serial Wire Debug
  - Debug interface with only two signals (SWD and SWCLK)
  - ARM devices only (like STM32)
  - https://static.docs.arm.com/ihi0031/c/IHI0031C_debug_interface_as.pdf

How to setup JTAG

- Identify the pins
  - JTAGulator
    - Marvell 88mw300
- Successful ID!



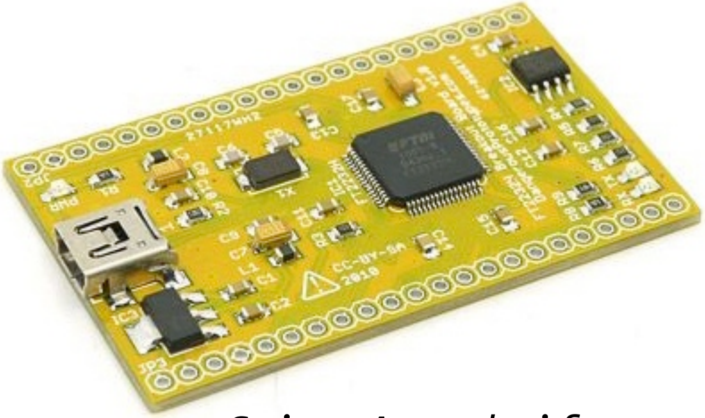


```
:B
Enter number of channels to use (4 - 24): 6
Ensure connections are on CH5..CH0.
Possible permutations: 360
Press spacebar to begin (any other key to abort)...
JTAGulating! Press any key to
abort..................................
TDI: 5
TDO: 2
TCK: 0
TMS: 3
Number of devices detected: 1
...
BYPASS scan complete!
:B.
?
:D
TDI not needed to retrieve Device ID.
Enter new TDO pin [0]: 2
Enter new TCK pin [0]: 0
Enter new TMS pin [0]: 3
Enter number of devices in JTAG chain [0]: 1
All other channels set to output HIGH.

Device ID: 1111 1100001100000000 00000011011 1 (0xFC300037)
-> Manufacturer ID: 0x01B
-> Part Number: 0xC300
-> Version: 0xF

IDCODE listing complete!
```

Establish JTAG connection

- The five JTAG signals are now identified.
- Connect your Debug Probe

  - Rich guys => Segger EDU (60$)
  - **Limited guys => FT2232H Board (20$)**

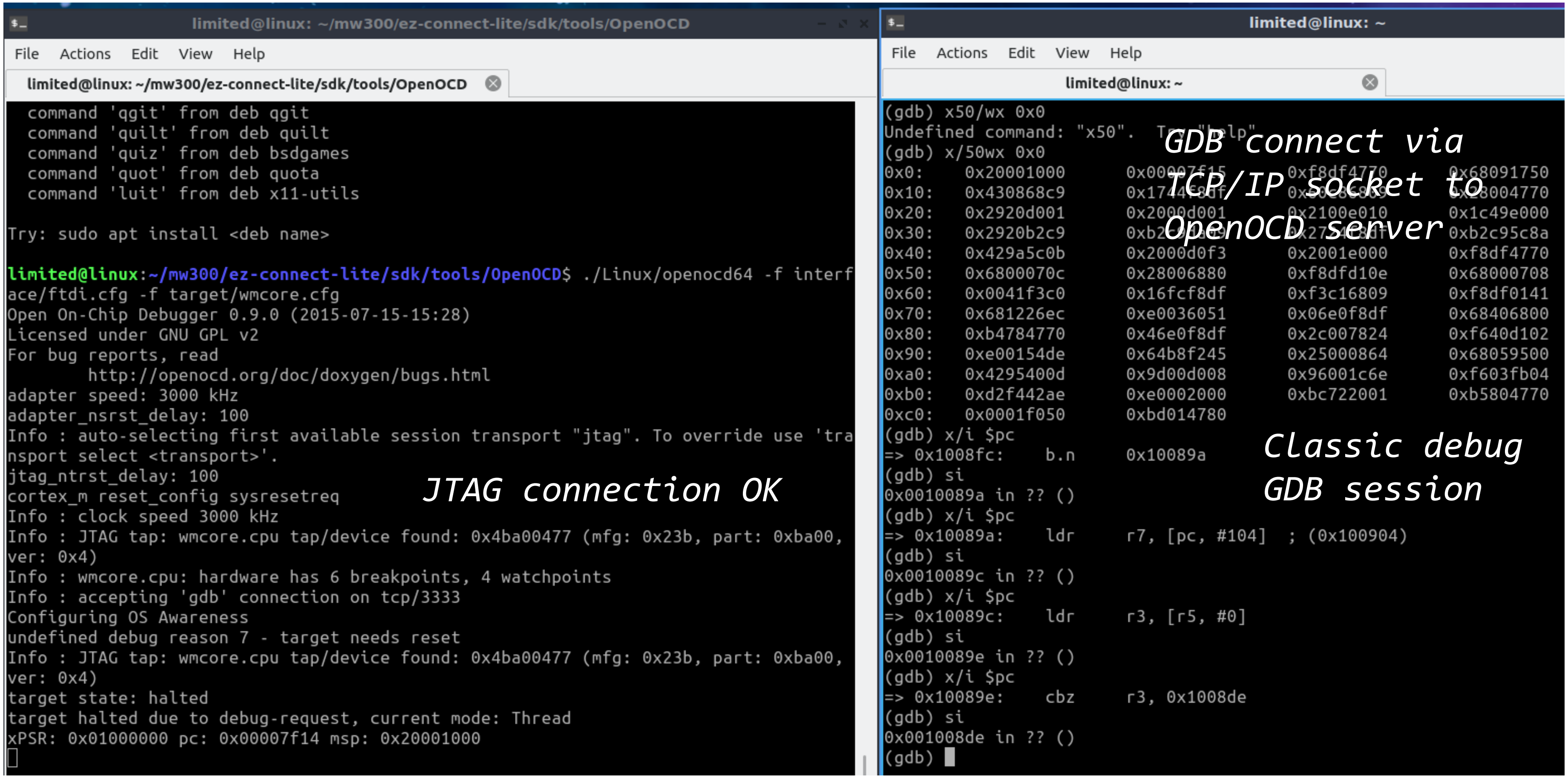


Swiss Army knife

- Install OpenOCD
  - Gnu-mcu-eclipse
- Grab the mw300 config
  - On the website
- ./openocd -f mw300.cfg

Profit the JTAG power

- OpenOCD + config + gdb = Full Debug



limited@linux: ~/mw300/ez-connect-lite/sdk/tools/OpenOCD
File Actions Edit View Help
limited@linux: ~/mw300/ez-connect-lite/sdk/tools/OpenOCD

```
  command 'qgit' from deb qgit
  command 'quilt' from deb quilt
  command 'quiz' from deb bsdgames
  command 'quot' from deb quota
  command 'luit' from deb x11-utils

Try: sudo apt install <deb name>

limited@linux:~/mw300/ez-connect-lite/sdk/tools/OpenOCD$ ./Linux/openocd64 -f interf
ace/ftdi.cfg -f target/wmcore.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'tra
nsport select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00,
ver: 0x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : accepting 'gdb' connection on tcp/3333
Configuring OS Awareness
undefined debug reason 7 - target needs reset
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00,
ver: 0x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
```

JTAG connection OK

```
(gdb) x50/wx 0x0
Undefined command: "x50".  Try "help".
(gdb) x/50wx 0x0
0x0:    0x20001000      0x00007f15      0xf8df4770      0x68091750
0x10:   0x430868c9      0x1744f8df      [illegible]     [illegible]
0x20:   0x2920d001      0x2000d001      0x2100e010      0x1c49e000
0x30:   0x2920b2c9      [illegible]     [illegible]     0xb2c95c8a
0x40:   0x429a5c0b      0x2000d0f3      0x2001e000      0xf8df4770
0x50:   0x6800070c      0x28006880      0xf8dfd10e      0x68000708
0x60:   0x0041f3c0      0x16fcf8df      0xf3c16809      0xf8df0141
0x70:   0x681226ec      0xe0036051      0x06e0f8df      0x68406800
0x80:   0xb4784770      0x46e0f8df      0x2c007824      0xf640d102
0x90:   0xe00154de      0x64b8f245      0x25000864      0x68059500
0xa0:   0x4295400d      0x9d00d008      0x96001c6e      0xf603fb04
0xb0:   0xd2f442ae      0xe0002000      0xbc722001      0xb5804770
0xc0:   0x0001f050      0xbd014780
(gdb) x/i $pc
=> 0x1008fc:    b.n     0x10089a
(gdb) si
0x0010089a in ?? ()
(gdb) x/i $pc
=> 0x10089a:    ldr     r7, [pc, #104]  ; (0x100904)
(gdb) si
0x0010089c in ?? ()
(gdb) x/i $pc
=> 0x10089c:    ldr     r3, [r5, #0]
(gdb) si
0x0010089e in ?? ()
(gdb) x/i $pc
=> 0x10089e:    cbz     r3, 0x1008de
(gdb) si
0x001008de in ?? ()
(gdb) █
```

GDB connect via TCP/IP socket to OpenOCD server

Classic debug GDB session

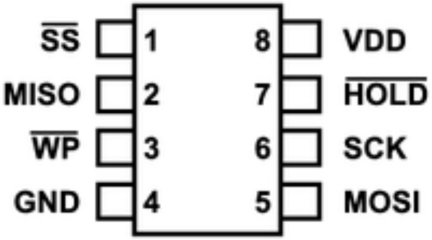
The Results

- JTAG not disabled
  - Firmware extracted
    - **Wi-Fi credentials in Plaintext**
  - Full control of the Marvell mw300
    - R/W
    - Code exec
  - Possibility to flash the device to insert persistent backdoor
    - Supply chain attack
- Low cost attack
  - Less than 150$, one hour, no skills
  - Device still OK, can be sold/reused

Cheaper attack? The SPI way

- Imagine
  - No JTAG probe?
  - or even JTAG disabled?
- But we need the firmware!

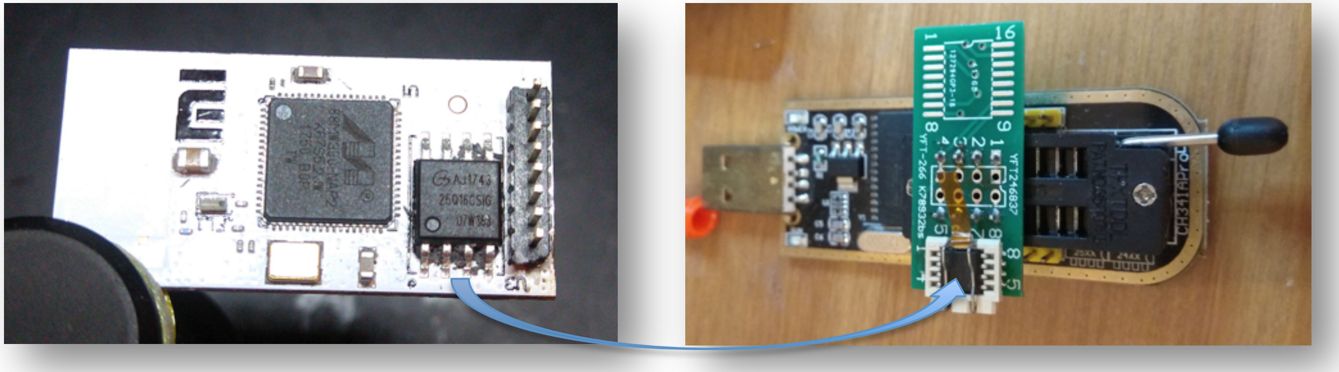
- Most of cheap IoT Devices use SPI Flash ICs as storage
  - Macronix (MXIC)
  - GigaDevice (GD)
  - And ISSI,SPANSION (now Cypress)



- SPI is serial protocol
  - Easy to play with SPI, specs online, FT2232H support
  - Can be decoded on the fly easily (Sigrok or Saleae)

The Cheapest way!

- Let’s dump the SPI flash
  - CH341a usb programmer
  - Flashrom tool (a must-have)

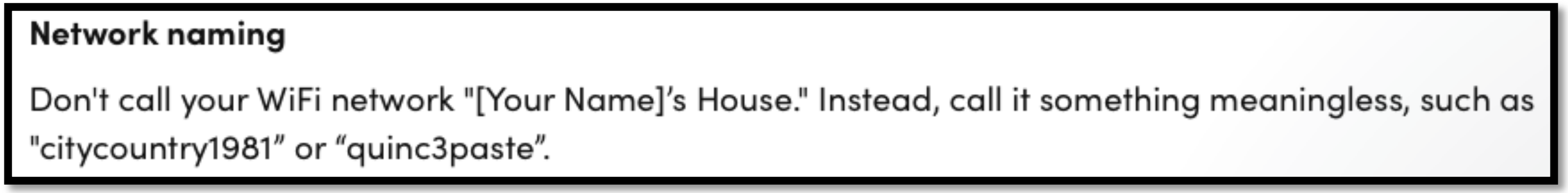


Unsolder the SPI flash to read it directly

- Cheapest attack
  - Less than 10$, one hour, no skills
  - You got the full firmware
    - Wi-Fi credentials in clear, FW ready for ghidra

Pwn n*2: LIFX

- IoT Company (Melbourne)
  - Been hacked in the past…
  - Now they have a real security policy
    - https://www.lifx.com/pages/keeping-your-devices-and-yourself-secure



Network naming

Don't call your WiFi network "[Your Name]'s House." Instead, call it something meaningless, such as "citycountry1981" or "quinc3paste".

High level Security

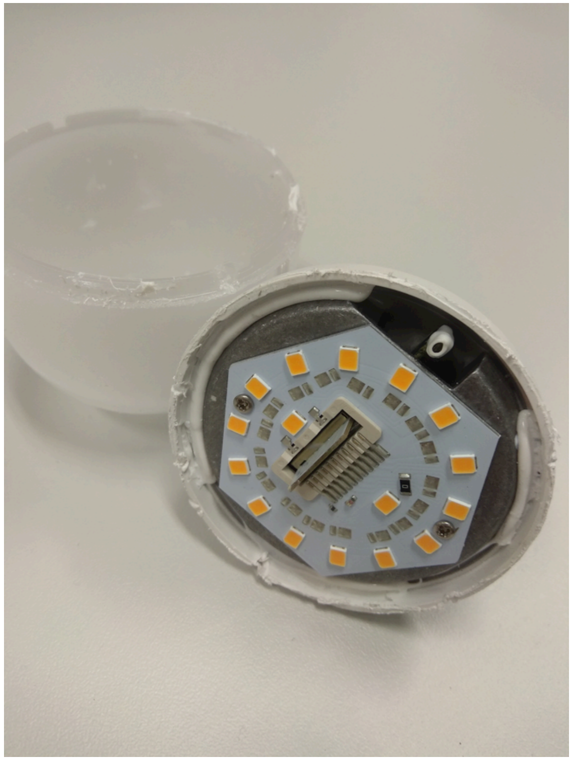
- Focus on the LIFX Mini
  - 30$ on Amazon
    - Discount! 15$ now

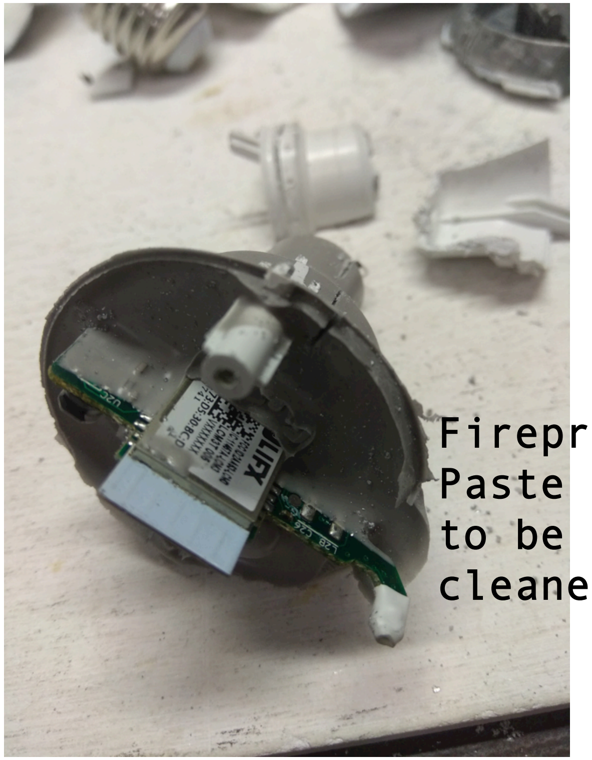
Like a butcher

- WARNING
  - Advanced tools are necessary for this hack

The LIFX teardown

- Teardown
  - Bulb is removed
  - Access to the electronic module

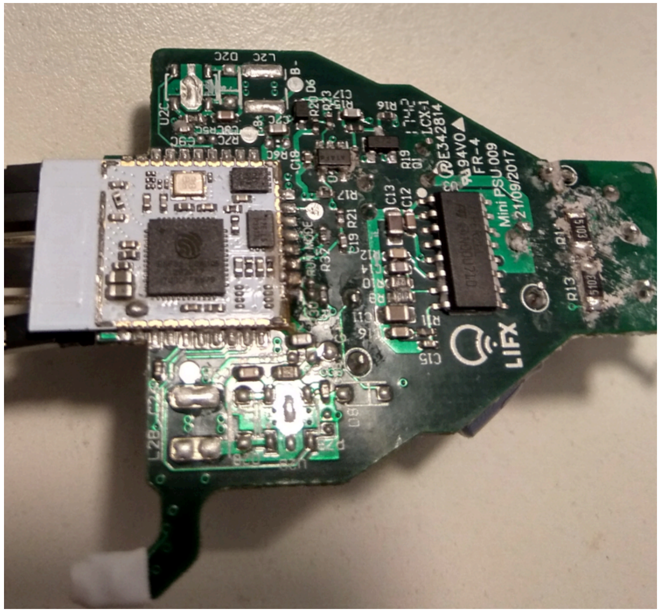


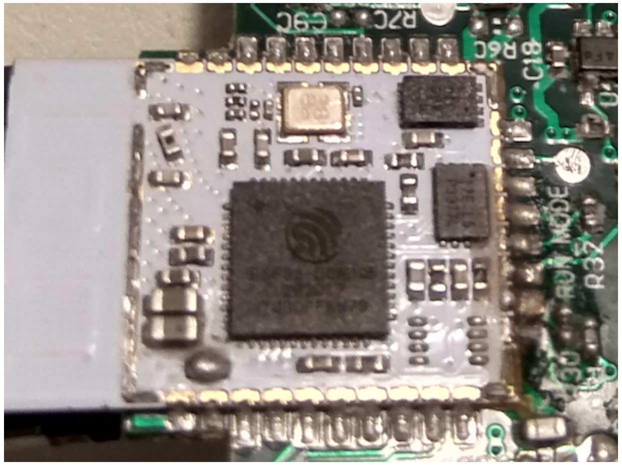


Fireproof Paste has to be cleaned

PCB reverse

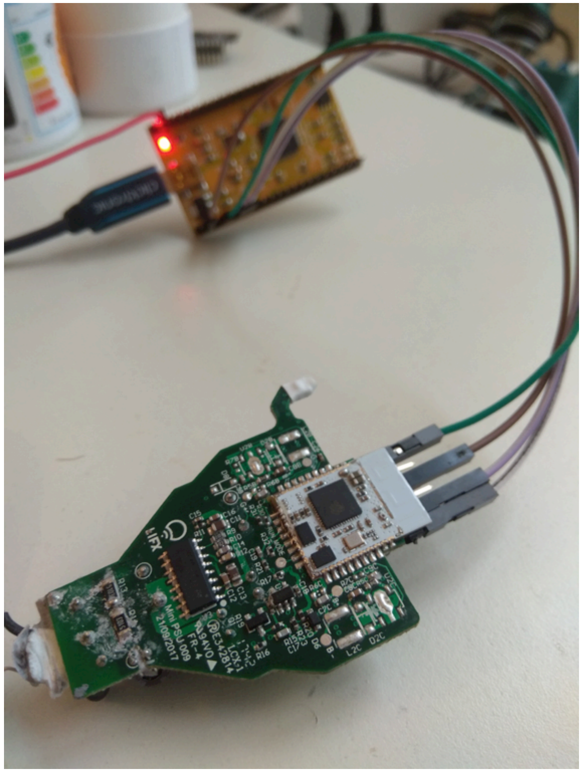
- The shield is removed
- ESP32 inside
  - Wi-Fi SoC
  - Xtensa CPU
  - SDK on github (esp-idf)
  - Datasheet online
- About security
  - OTP Efuses
  - AES accelerator
  - Secure boot
  - Firmware encryption
- Not bad!



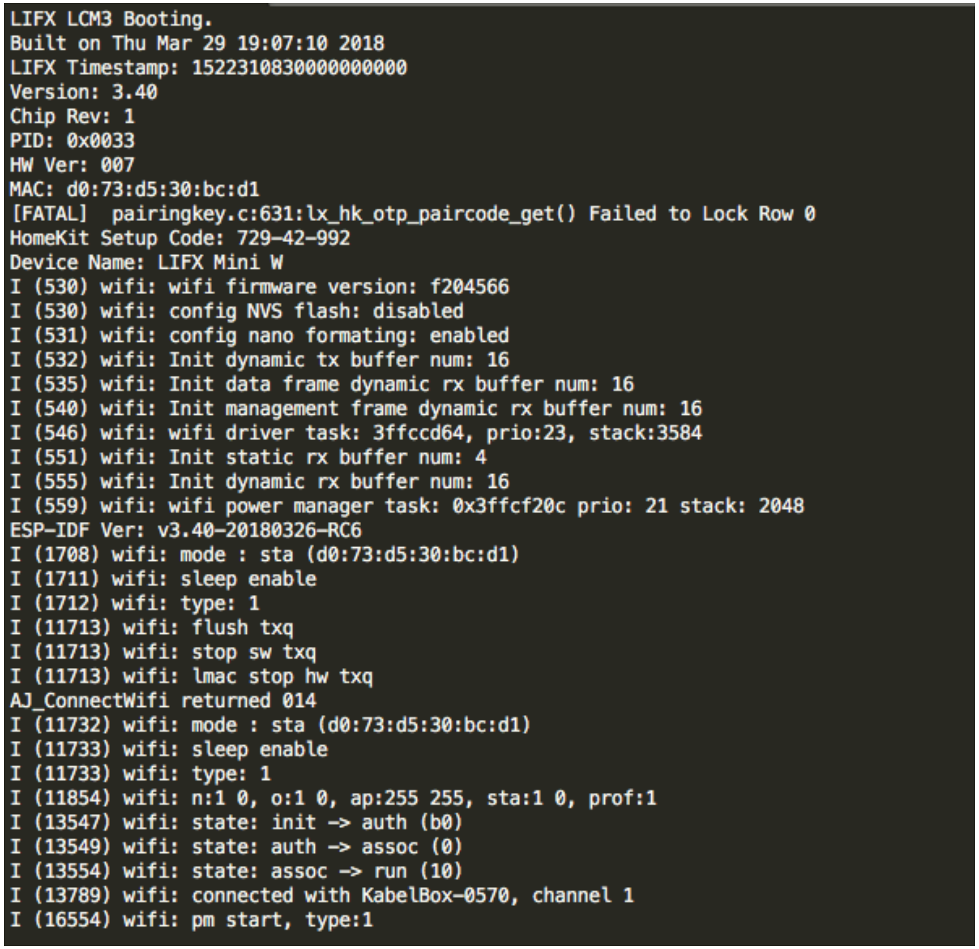


The setup

- Is it alive?
- FT2232H + 4 wires
  - 3,3V, GND, TX, RX (UART)



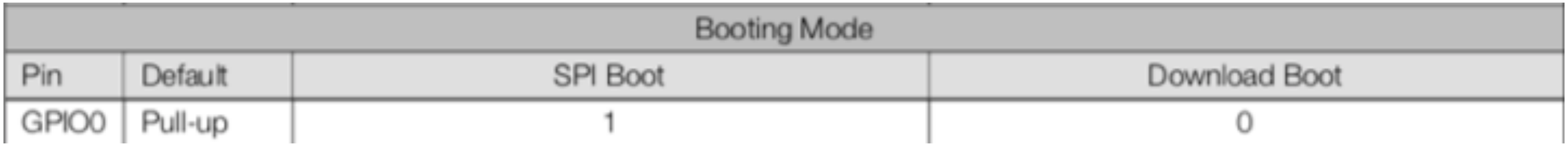
- Uart Log



```
LIFX LCM3 Booting.
Built on Thu Mar 29 19:07:10 2018
LIFX Timestamp: 1522310830000000000
Version: 3.40
Chip Rev: 1
PID: 0x0033
HW Ver: 007
MAC: d0:73:d5:30:bc:d1
[FATAL]  pairingkey.c:631:lx_hk_otp_paircode_get() Failed to Lock Row 0
HomeKit Setup Code: 729-42-992
Device Name: LIFX Mini W
I (530) wifi: wifi firmware version: f204566
I (530) wifi: config NVS flash: disabled
I (531) wifi: config nano formating: enabled
I (532) wifi: Init dynamic tx buffer num: 16
I (535) wifi: Init data frame dynamic rx buffer num: 16
I (540) wifi: Init management frame dynamic rx buffer num: 16
I (546) wifi: wifi driver task: 3ffccd64, prio:23, stack:3584
I (551) wifi: Init static rx buffer num: 4
I (555) wifi: Init dynamic rx buffer num: 16
I (559) wifi: wifi power manager task: 0x3ffcf20c prio: 21 stack: 2048
ESP-IDF Ver: v3.40-20180326-RC6
I (1708) wifi: mode : sta (d0:73:d5:30:bc:d1)
I (1711) wifi: sleep enable
I (1712) wifi: type: 1
I (11713) wifi: flush txq
I (11713) wifi: stop sw txq
I (11713) wifi: lmac stop hw txq
AJ_ConnectWifi returned 014
I (11732) wifi: mode : sta (d0:73:d5:30:bc:d1)
I (11733) wifi: sleep enable
I (11733) wifi: type: 1
I (11854) wifi: n:1 0, o:1 0, ap:255 255, sta:1 0, prof:1
I (13547) wifi: state: init -> auth (b0)
I (13549) wifi: state: auth -> assoc (0)
I (13554) wifi: state: assoc -> run (10)
I (13789) wifi: connected with KabelBox-0570, channel 1
I (16554) wifi: pm start, type:1
```

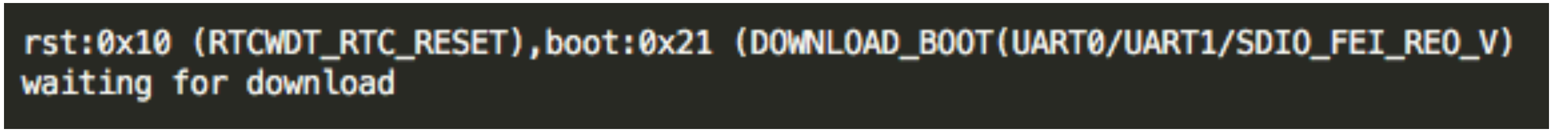
Get the firmware

- Reading the datasheet:



		Booting Mode	
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0

  – Io0 grounded + PowerON to access download boot



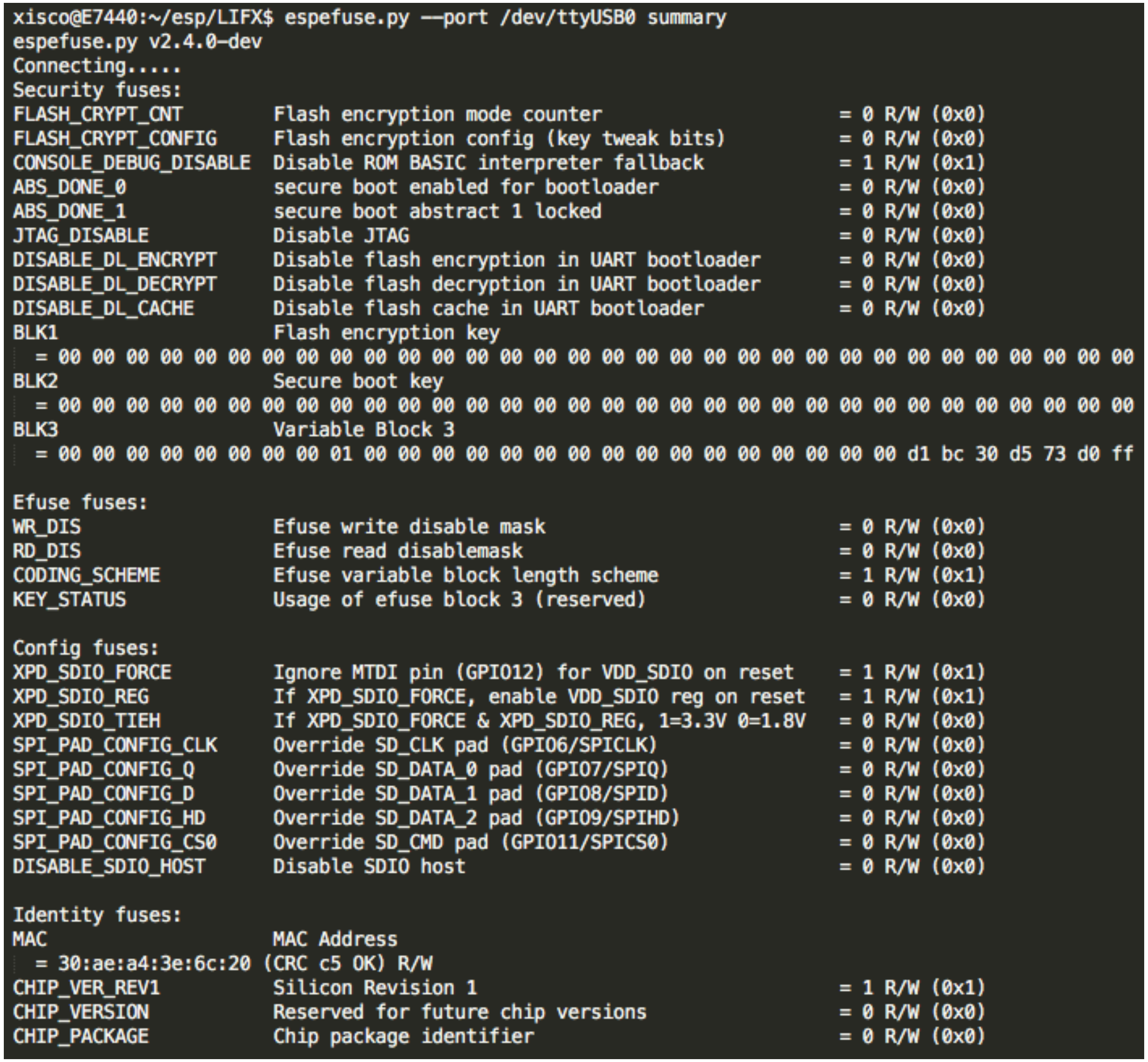
```
rst:0x10 (RTCWDT_RTC_RESET),boot:0x21 (DOWNLOAD_BOOT(UART0/UART1/SDIO_FEI_REO_V)
waiting for download
```

UART output when the ESP32 is on Download Boot

- Results:
  – Extract the firmware
    - *esptool.py -p /dev/ttyUSB0 -b 460800 read_flash 0 0x200000 flash.bin*
  – Reverse and Profit!
    - Wi-Fi credentials in plaintext(again)

More Vulns? yes

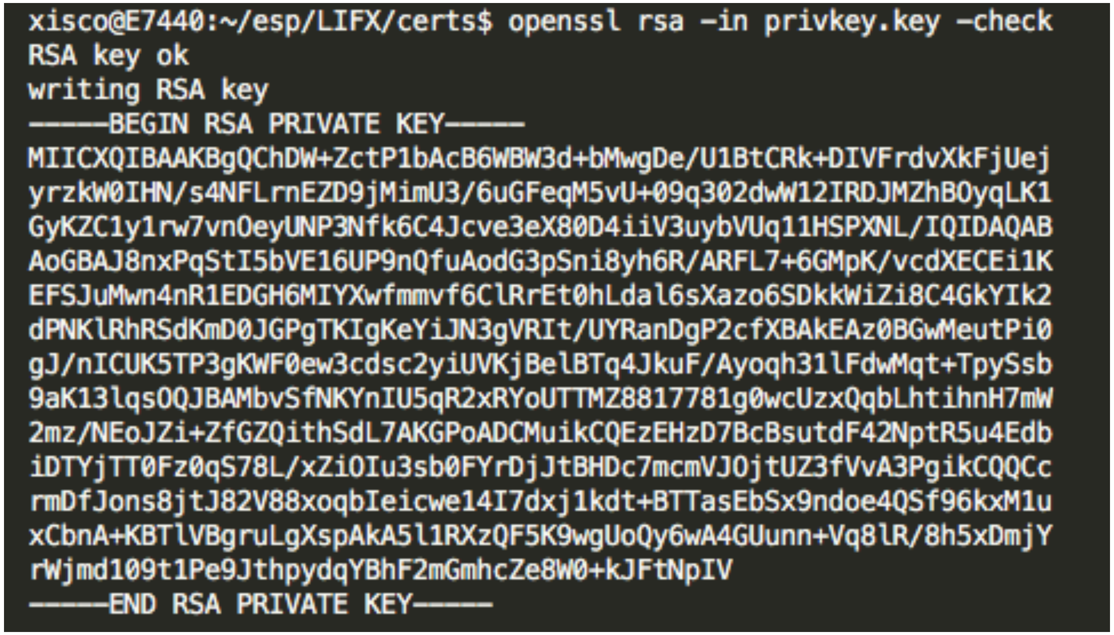
- Security configuration totally blank
  - Just by dumping the E-fuses
- Results:
  - No sec boot
  - No fw encryption
  - No JTAG disabled
- It is a dev board!



```
xisco@E7440:~/esp/LIFX$ espefuse.py --port /dev/ttyUSB0 summary
espefuse.py v2.4.0-dev
Connecting.....
Security fuses:
FLASH_CRYPT_CNT        Flash encryption mode counter                     = 0 R/W (0x0)
FLASH_CRYPT_CONFIG     Flash encryption config (key tweak bits)          = 0 R/W (0x0)
CONSOLE_DEBUG_DISABLE  Disable ROM BASIC interpreter fallback            = 1 R/W (0x1)
ABS_DONE_0             secure boot enabled for bootloader                = 0 R/W (0x0)
ABS_DONE_1             secure boot abstract 1 locked                     = 0 R/W (0x0)
JTAG_DISABLE           Disable JTAG                                      = 0 R/W (0x0)
DISABLE_DL_ENCRYPT     Disable flash encryption in UART bootloader       = 0 R/W (0x0)
DISABLE_DL_DECRYPT     Disable flash decryption in UART bootloader       = 0 R/W (0x0)
DISABLE_DL_CACHE       Disable flash cache in UART bootloader            = 0 R/W (0x0)
BLK1                   Flash encryption key
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BLK2                   Secure boot key
  = 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
BLK3                   Variable Block 3
  = 00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 d1 bc 30 d5 73 d0 ff

Efuse fuses:
WR_DIS                 Efuse write disable mask                          = 0 R/W (0x0)
RD_DIS                 Efuse read disablemask                            = 0 R/W (0x0)
CODING_SCHEME          Efuse variable block length scheme                = 1 R/W (0x1)
KEY_STATUS             Usage of efuse block 3 (reserved)                 = 0 R/W (0x0)

Config fuses:
XPD_SDIO_FORCE         Ignore MTDI pin (GPIO12) for VDD_SDIO on reset    = 1 R/W (0x1)
XPD_SDIO_REG           If XPD_SDIO_FORCE, enable VDD_SDIO reg on reset   = 1 R/W (0x1)
XPD_SDIO_TIEH          If XPD_SDIO_FORCE & XPD_SDIO_REG, 1=3.3V 0=1.8V   = 0 R/W (0x0)
SPI_PAD_CONFIG_CLK     Override SD_CLK pad (GPIO6/SPICLK)                = 0 R/W (0x0)
SPI_PAD_CONFIG_Q       Override SD_DATA_0 pad (GPIO7/SPIQ)               = 0 R/W (0x0)
SPI_PAD_CONFIG_D       Override SD_DATA_1 pad (GPIO8/SPID)               = 0 R/W (0x0)
SPI_PAD_CONFIG_HD      Override SD_DATA_2 pad (GPIO9/SPIHD)              = 0 R/W (0x0)
SPI_PAD_CONFIG_CS0     Override SD_CMD pad (GPIO11/SPICS0)               = 0 R/W (0x0)
DISABLE_SDIO_HOST      Disable SDIO host                                 = 0 R/W (0x0)

Identity fuses:
MAC                    MAC Address
  = 30:ae:a4:3e:6c:20 (CRC c5 OK) R/W
CHIP_VER_REV1          Silicon Revision 1                                = 1 R/W (0x1)
CHIP_VERSION           Reserved for future chip versions                 = 0 R/W (0x0)
CHIP_PACKAGE           Chip package identifier                           = 0 R/W (0x0)
```

More Vulns? Yes of course

- RSA Private Key in plaintext
  - Used for device onboarding



```
xisco@E7440:~/esp/LIFX/certs$ openssl rsa -in privkey.key -check
RSA key ok
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQChDW+ZctP1bAcB6WBW3d+bMwgDe/U1BtCRk+DIVFrdvXkFjUej
yrzkW0IHN/s4NFLrnEZD9jMimU3/6uGFeqM5vU+09q302dwW12IRDJMZhBOyqLK1
GyKZC1y1rw7vnOeyUNP3Nfk6C4Jcve3eX80D4iiV3uybVUq11HSPXNL/IQIDAQAB
AoGBAJ8nxPqStI5bVE16UP9nQfuAodG3pSni8yh6R/ARFL7+6GMpK/vcdXECEi1K
EFSJuMwn4nR1EDGH6MIYXwfmmvf6ClRrEt0hLdal6sXazo6SDkkWiZi8C4GkYIk2
dPNKlRhRSdKmD0JGPgTKIgKeYiJN3gVRIt/UYRanDgP2cfXBAkEAz0BGwMeutPi0
gJ/nICUK5TP3gKWF0ew3cdsc2yiUVKjBelBTq4JkuF/Ayoqh31lFdwMqt+TpySsb
9aK13lqsOQJBAMbvSfNKYnIU5qR2xRYoUTTMZ8817781g0wcUzxQqbLhtihnH7mW
2mz/NEoJZi+ZfGZQithSdL7AKGPoADCMuikCQEzEHzD7BcBsutdF42NptR5u4Edb
iDTYjTT0Fz0qS78L/xZiOIu3sb0FYrDjJtBHDc7mcmVJOjtUZ3fVvA3PgikCQQCc
rmDfJons8jtJ82V88xoqbIeicwe14I7dxj1kdt+BTTasEbSx9ndoe4QSf96kxM1u
xCbnA+KBTlVBgruLgXspAkA5l1RXzQF5K9wgUoQy6wA4GUunn+Vq8lR/8h5xDmjY
rWjmd109t1Pe9JthpydqYBhF2mGmhcZe8W0+kJFtNpIV
-----END RSA PRIVATE KEY-----
```

- Still the same key the last time I check…

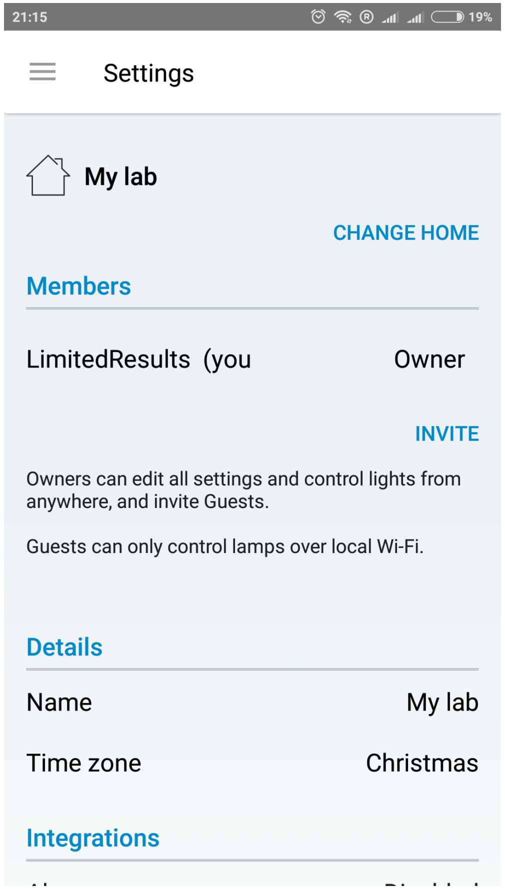
The Results

- Main issues
  - Wi-Fi credentials in Plaintext
  - Unsecure configuration
  - Access to the RSA private key for onboarding
- **The REAL issues**
  - **Serial Bootloader cannot be disabled in ESP32**
    - **Offer an easy access to dump the FW (always)**
  - **ESP32 has interesting but not used security features**
    - **Secure boot and Flash encryption**
- Low cost attack
  - 25$, 30min, no skills
  - Device is destroyed. OK, who cares?

Pwn n*3: WIZ

- Wiz = IoT branch of TAO light (Shanghai)

- Focus on Wiz light
  - 20e on Amazon



Settings

My lab

CHANGE HOME

Members

LimitedResults (you — Owner

INVITE

Owners can edit all settings and control lights from anywhere, and invite Guests.

Guests can only control lamps over local Wi-Fi.

Details

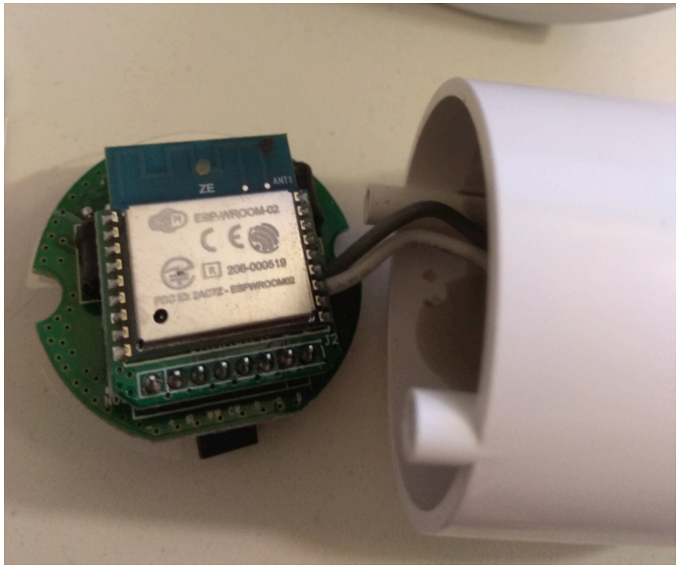
Name — My lab

Time zone — Christmas

Integrations

Easy Teardown

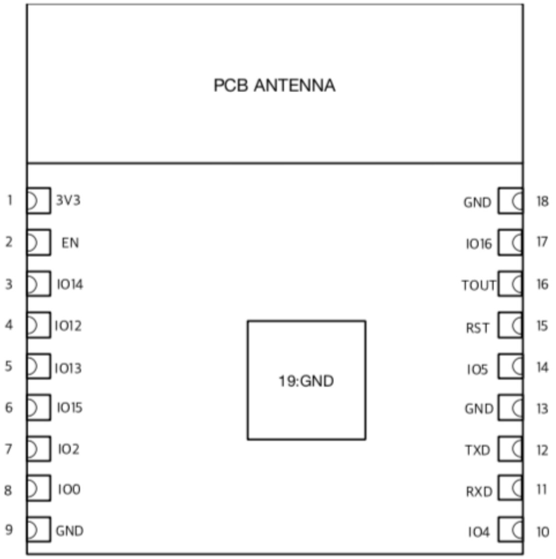
- Two minutes to access the module

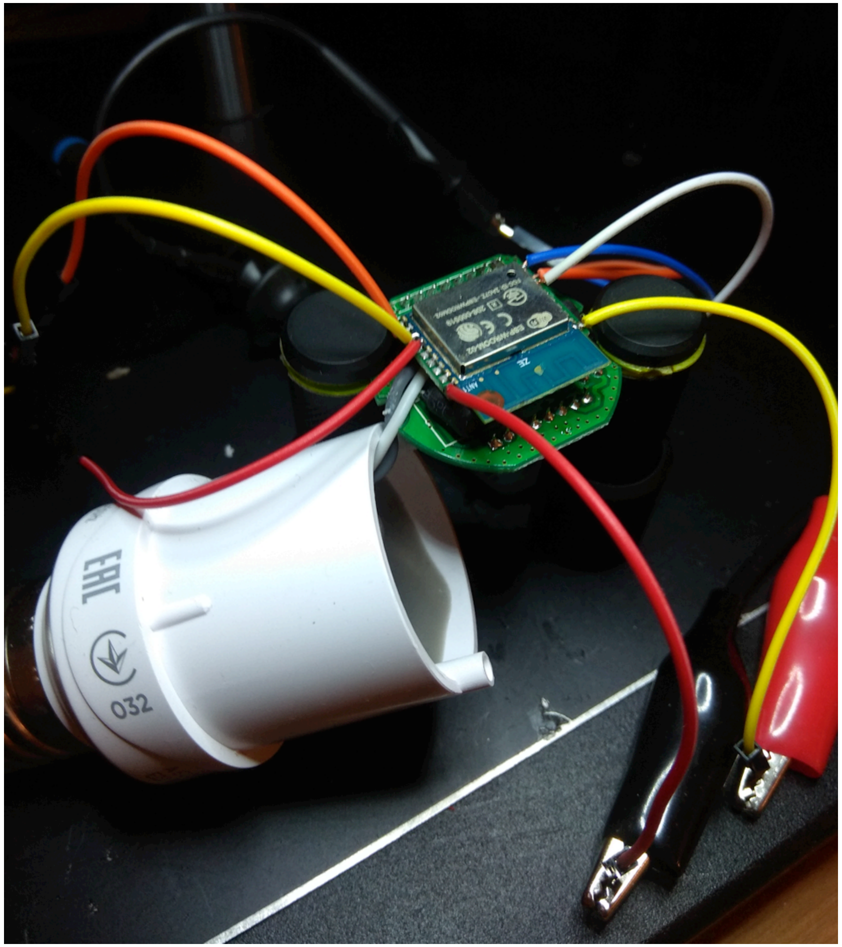


- The module is ESP-WROOM-02(based on ESP8266)
  - ESP8266 has no security features
  - SDK doesn’t support encryption/obfuscation of sensitive data

Dirty Setup

- Just look the Pinout module to access UART

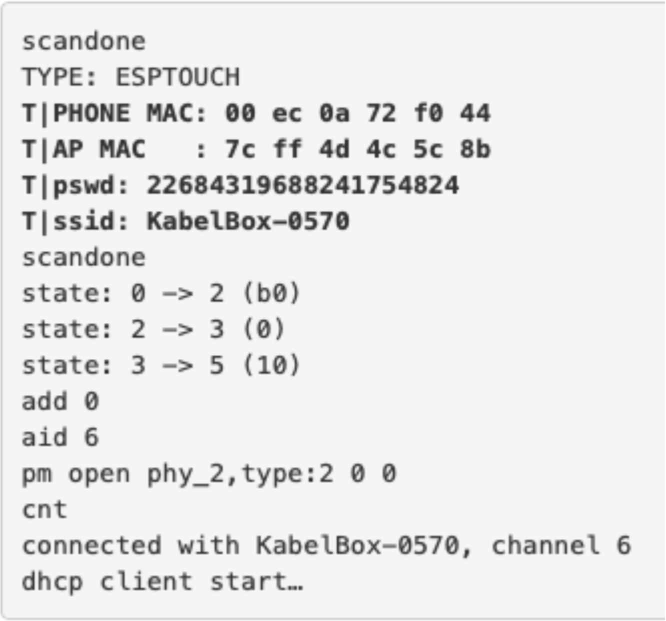




- No damage at all
  - Supply the device using 3.3V DC
  - Plug a Uart to USB cable
  - Put the chip in download mode
  - Enjoy

The Wiz Vulns

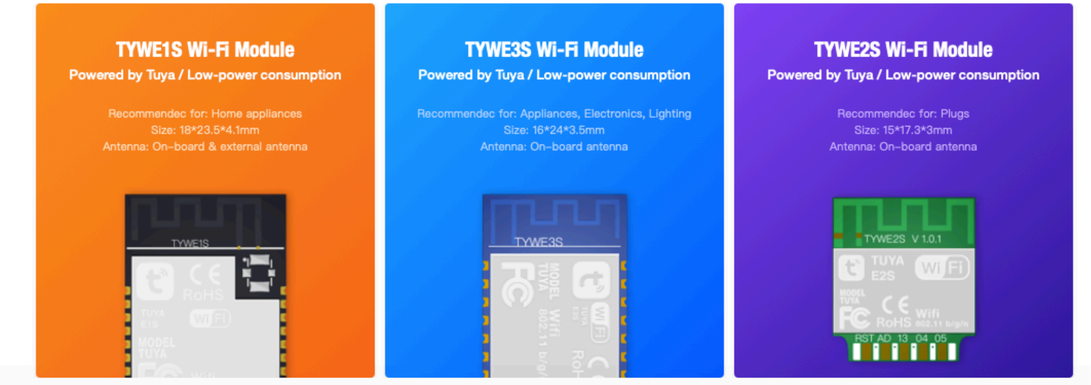
- Same VULNS than previously
  - Not a big surprise here
  - And they don’t want to patch :-/
- Extra vuln during normal mode:
  - The Wi-Fi credentials displayed to UART!
- Hard to exploit, I am agree
  - But that gives an idea how the device was design in a security point of view…
  - They do not even clean the debug printf()
  - Come on… What can we do???



```
scandone
TYPE: ESPTOUCH
T|PHONE MAC: 00 ec 0a 72 f0 44
T|AP MAC    : 7c ff 4d 4c 5c 8b
T|pswd: 22684319688241754824
T|ssid: KabelBox-0570
scandone
state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)
add 0
aid 6
pm open phy_2,type:2 0 0
cnt
connected with KabelBox-0570, channel 6
dhcp client start…
```

Pwn n*4: Tuya Lighbulbs

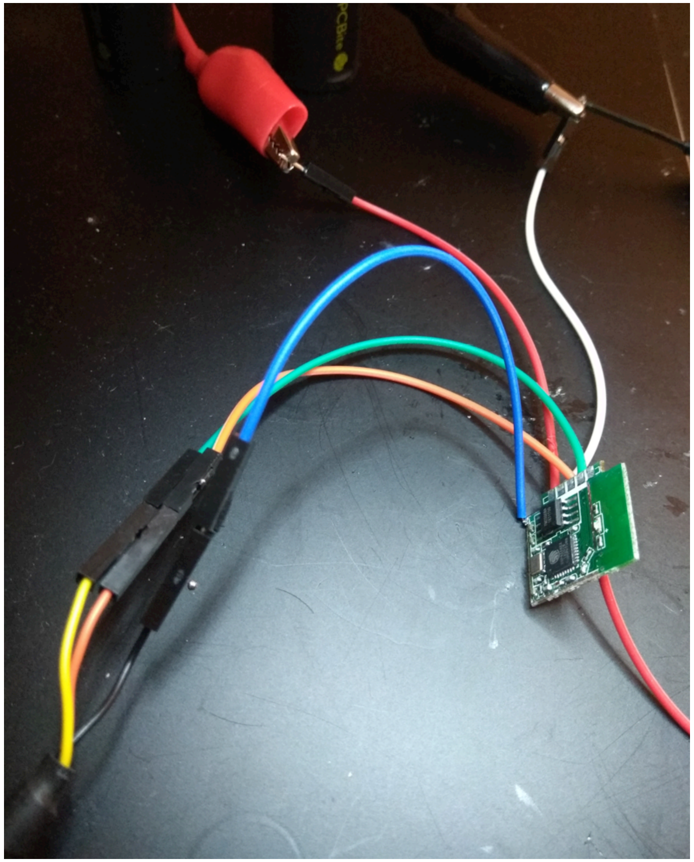
- Tuya Smart is a complete IoT ecosystem company
  - https://en.tuya.com/solutions-details/xlfa1
- They provide HW design, app development, connectivity, Cloud services…

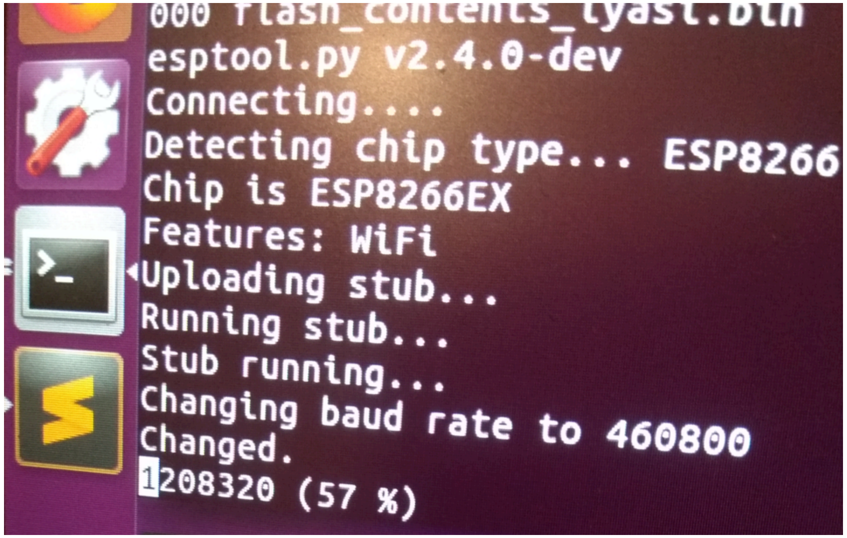

TYWE1S Wi-Fi Module
Powered by Tuya / Low-power consumption
Recommendec for: Home appliances
Size: 18*23.5*4.1mm
Antenna: On-board & external antenna
TYWE3S Wi-Fi Module
Powered by Tuya / Low-power consumption
Recommendec for: Appliances, Electronics, Lighting
Size: 16*24*3.5mm
Antenna: On-board antenna
TYWE2S Wi-Fi Module
Powered by Tuya / Low-power consumption
Recommendec for: Plugs
Size: 15*17.3*3mm
Antenna: On-board antenna

- Focus on Lyasi Device
  - Maybe 10 euros?

5 min-hack

- Also based on ESP8266





```
000 flash_contents_tyast.bin
esptool.py v2.4.0-dev
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
1208320 (57 %)
```

- Dirty setup again…
- Dump, reverse (a bit)
- Remount, Enjoy!

Tuya Results

- A complete MQTT client is running into the bulb
  - DeviceID and LocalKey are hardcoded
- Easy to control the bulb directly
  - PoC using a Python script
  - Tuyapi on github
    - https://github.com/codetheweb/tuyapi
  - Even if the app is running, MQTT messages can be sent to the bulb over Wi-FI
  - (and of course Wi-Fi credentials are in plaintext)

- *See www.limitedresults.com for PoC video*

Power Off

DISCUSSIONS

Quick Synthesis

- These investigations across different vendors show the same vulnerabilities
  - Bad security design
  - Bad security configuration
  - Lack of confidentiality
  - The ‘Clear user Data’ Feature not efficient
- Open the door to supply chain attacks
  - Wonderful spying capabilities
  - Don’t be surprised if this is exploited on the field..

(Limited) Impact

- For example, Resp. disclosure with LIFX was ~~difficult~~ limited
  - https://www.lifx.com/pages/privacy-security
- Three months to work on mitigations:
  - Encryption of the sensitive data!
  - New security settings!
  - RSA private key encrypted too!
- But…Is it SECURE now?
  - The FW v3.42 has been dumped and reversed
    - LIFX custom Encryption is ~~bad…~~ broken
    - Full of mistakes, badly designed
    - Ugly patch, sorry
- **Resp. disclosure again? LoL**

Back to Basics

- Basic rules (To apply to all IoT devices)
  - Network Segregation
    - Create an AP dedicated to IoT devices
  - Renew Passwords & apply Updates
    - No comment (I am lazy to do that…)
  - Think about the data you share
    - A bulb knows when you are at home, when you go to sleep…Pretty scary, isn’t it?
- The Medias, Companies, Schools should educate how to deal with connected objects
- Warning Labels on package/website such as:
  - *“No security inside”*
  - *“This product will share/store your private data”*
  - **Vendors refused**

IMHO

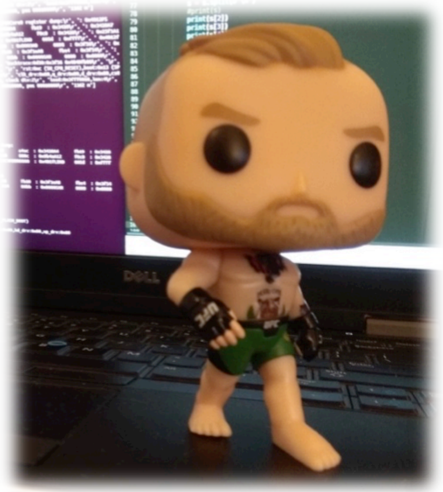
- (Most of) IoT vendors do not care about security
  - Priority to dev. costs & time to market
- Bug reports are often “complicated” or even impossible
  - No security contact
  - Security researchers considered as troublemakers
  - **Responsible disclosure just used as ‘damage control’**
    - **Never be a fan of Resp.disclo by the way…**
  - Need to use medias as leverage
- IoT vendors should learn from mobile phone industry
  - Bug bounties, mutual respect, continuous efforts to fix bugs…

Bricked

CONCLUSION

Conclusion

- Finally more about presenting some HW tricks to really pwn bulbs
- IoT ecosystem needs
  - A FULL secure Product Life Cycle
  - Regulations/Sanctions for unsecure vendors
  - Security ratings, certifications
  - Stop to consider security guys/girls as a Threat
  - Hire security engineers
  - Develop a TRUE security policy
- The customers have to be informed/ educated
  - Then they can make their choice
  - Who really needs connected light bulbs? :-/



Thank you! Questions?

- www.limitedresults.com
- @LimitedResults