Abusing EV Chargers
through Bluetooth and USB

Recon 2025

Riccardo Mori <rmori@quarkslab.com>
Robin David <rdavid@quarkslab.com>
Alexandre Chazal <achazal@quarkslab.com>

Warning

This pdf is an archived version of the slides.
To see the original slides with videos and animations, go ***here***

Bio

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

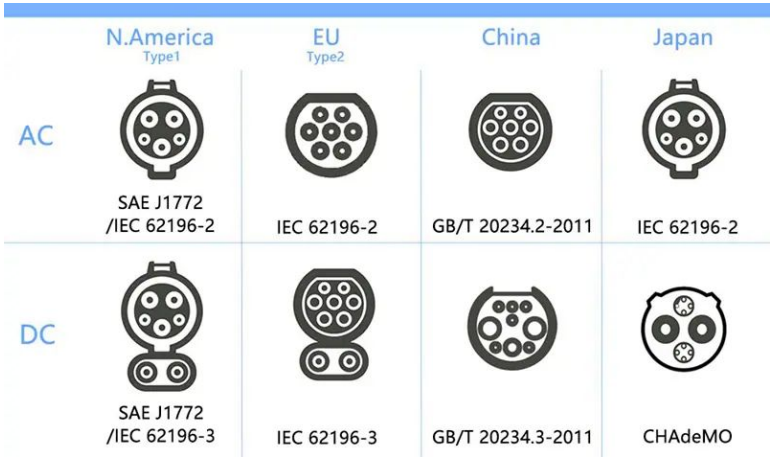
aka EV Charger

Today's focus

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	+++




VicOne
ZERO DAY INITIATIVE
PWN2OWN
AUTOMOTIVE
2025

⇒ **Bug bounty** competition for automotive equipments

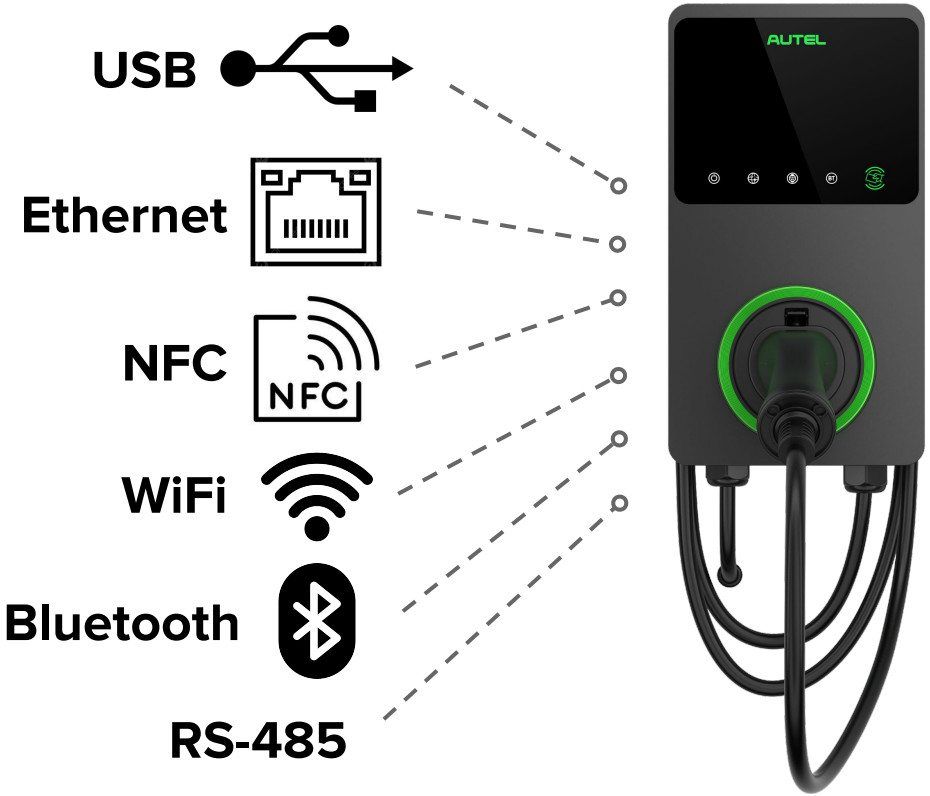
- Exploitation: 3 attempts of 10 minutes
- Need **proof of exploitation**

Target	Cash Prize	Master of Pwn Points
ChargePoint Home Flex (Model CPH50)	$50,000	5
Phoenix Contact CHARX SEC-3150	$50,000	5
WOLFBOX 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



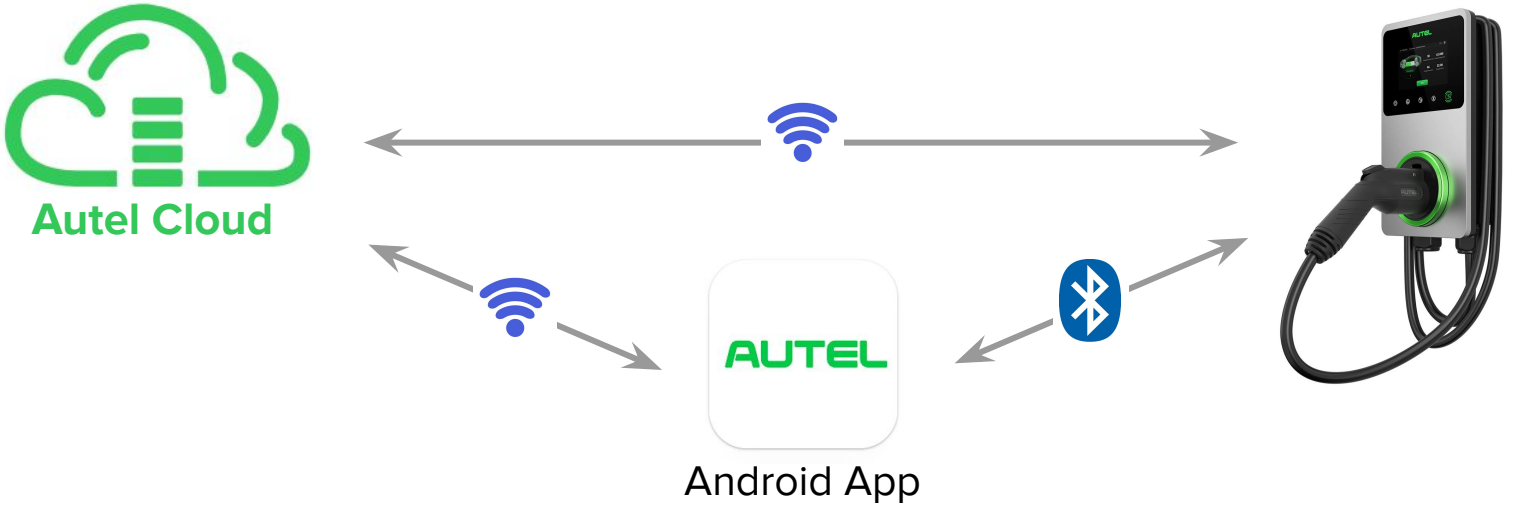
Specs:

- 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

Quarkslab

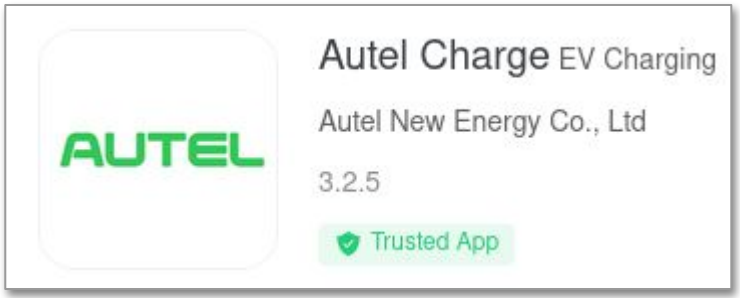
Intercepting Firmware



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)

```
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..."})
```

⇐

#4 Fetching Firmware

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

⇓

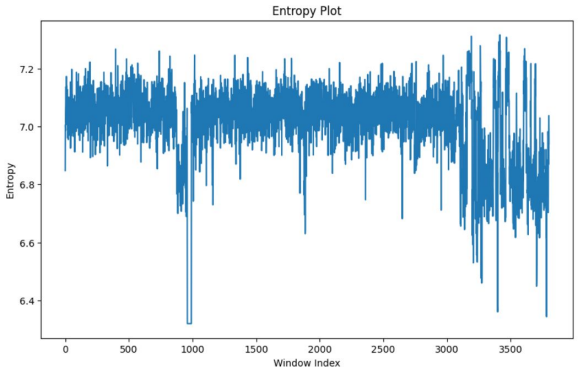
`Firmware_ECC01_V1.51.00.aut`


WE HAVE
THE FIRMWARE
BUT ITS ENCRYPTED ..
imgflip.com

Entropy Checks

Encrypted firmware analysis

Entropy

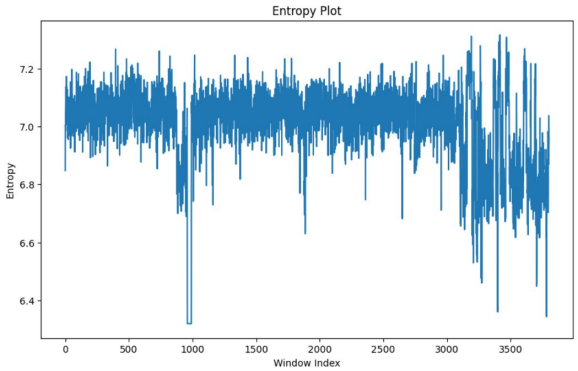


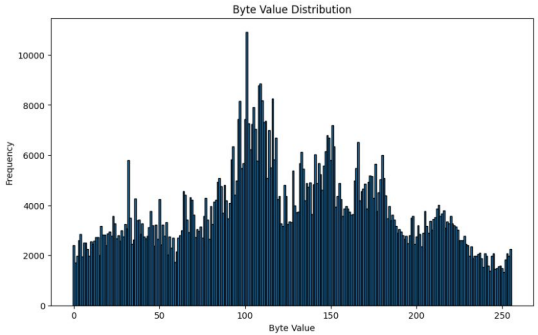
Entropy Checks

Encrypted firmware analysis

Entropy

Byte Distribution





Entropy Checks

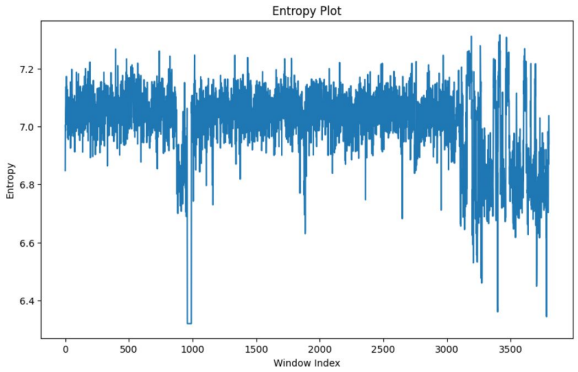
Encrypted firmware analysis

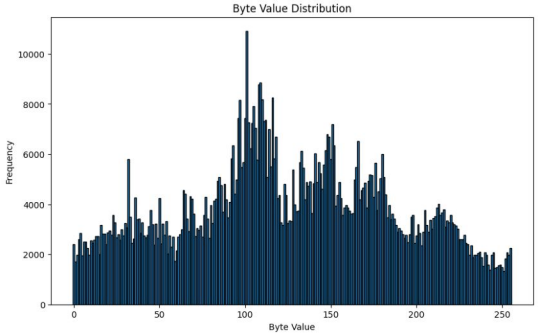
Crypto-Condor

Entropy

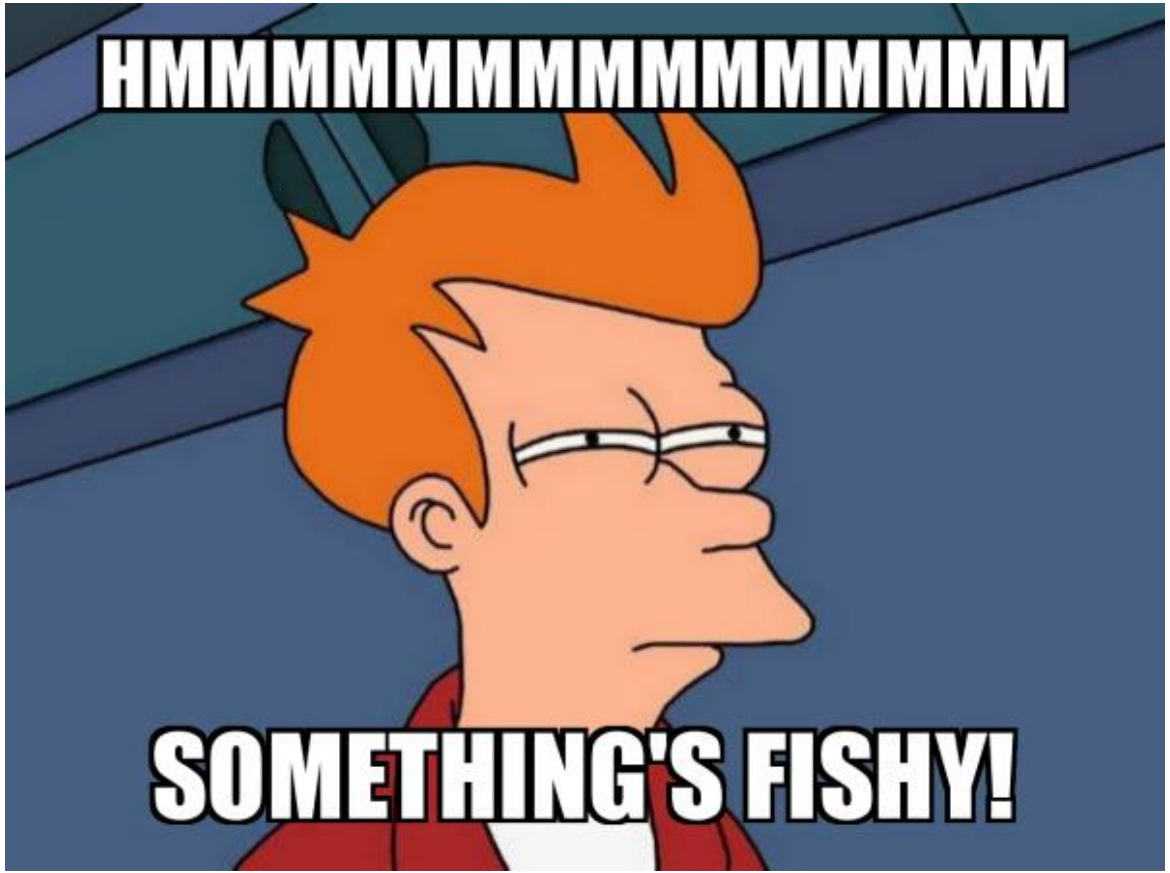
Byte Distribution

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)
```


HMMMMMMMMMMMMMMMM
SOMETHING'S FISHY!

Inferring Key Size: Kasiski/Babbage approach

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

Inferring Key Size: Kasiski/Babbage approach

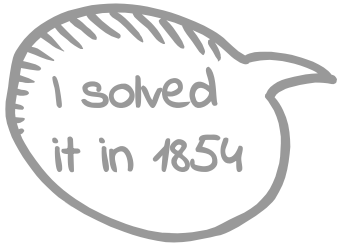
Guess #1: Simple operation (+, -, x, ^) with a repeating key

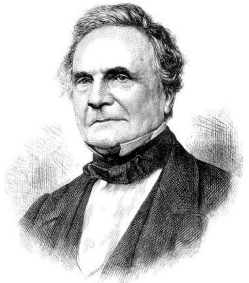
Principle:

⇒ Plaintext ciphered with the same key portion generates the same ciphertext

⇒ Thus theirs distance is a multiple of key length

⇒ The **GCD** of distances should be **multiple** of key length





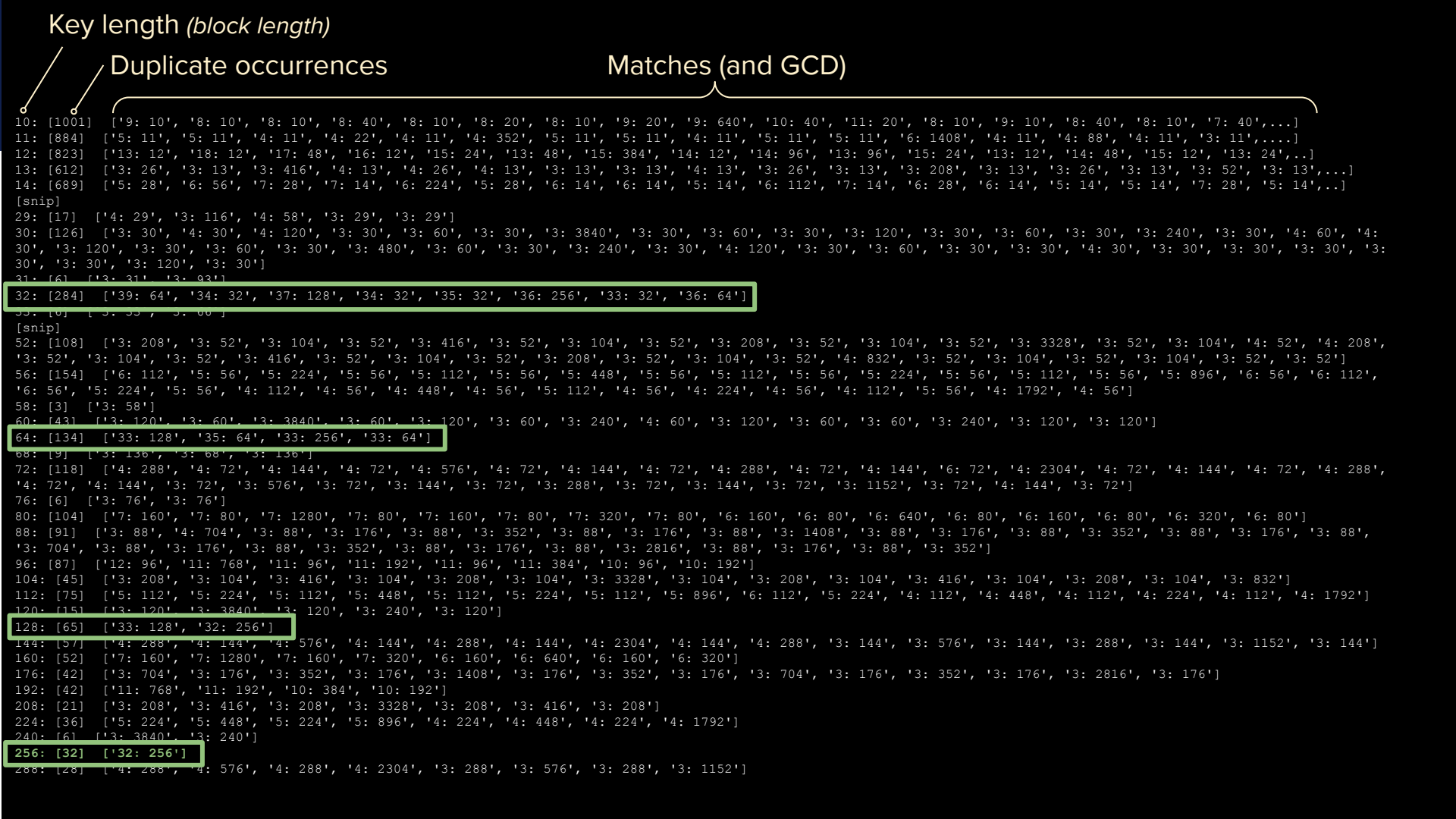
Charles Babbage
(1791-1871)

Friedrich Kasiski
(1805-1881)

Inferring Key Size: Example

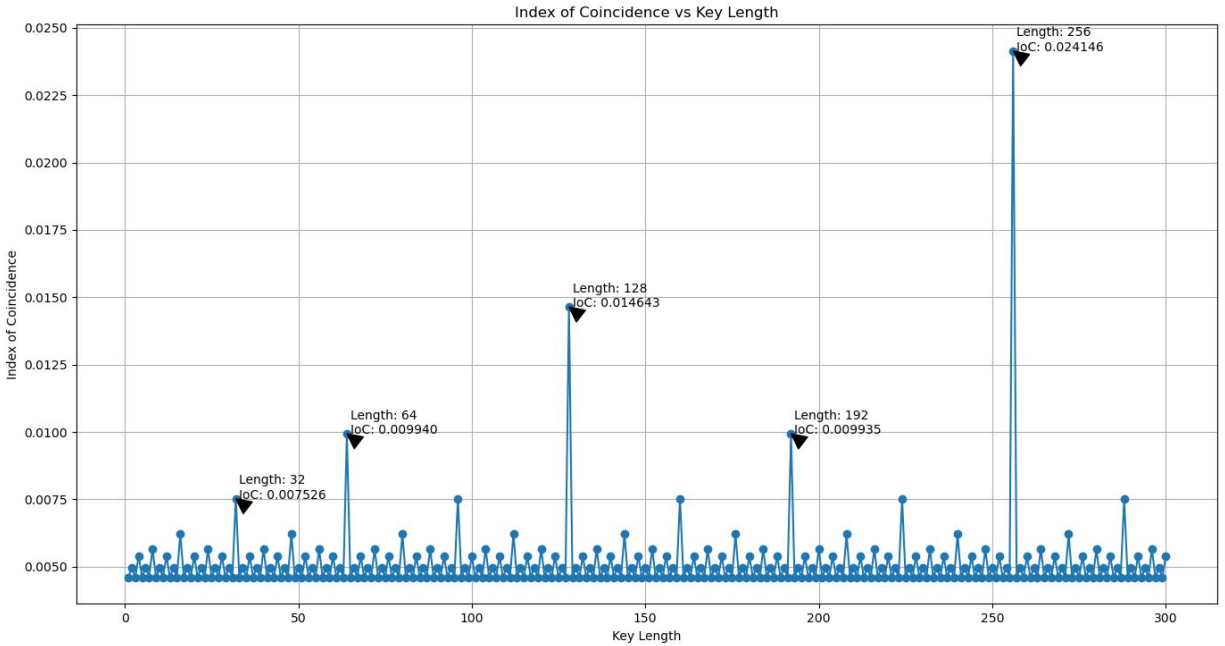
Gcd(60, 40, 10) ⇒ **10** *(key length)*

(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: [6] ['3: 33', '3: 66']
[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: 224', '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 computes most probable key length!



William Frederick Friedman
(1891-1969)

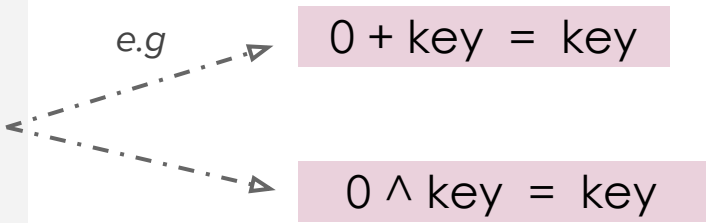
Technique invented in 1920 and published in *"The Index of Coincidence and its Applications in Cryptography"*

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)*



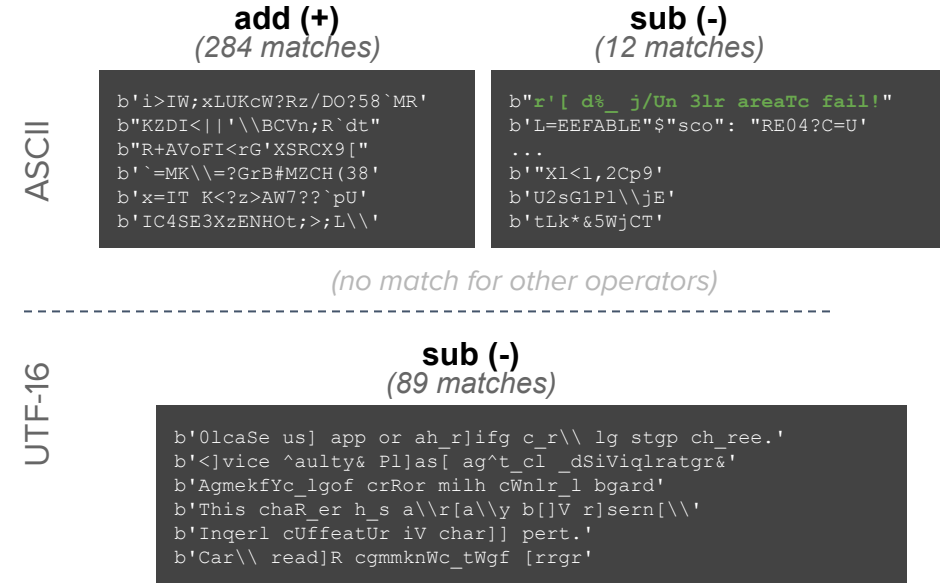
Cryptanalysis: Step #1 (First Attempt)

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



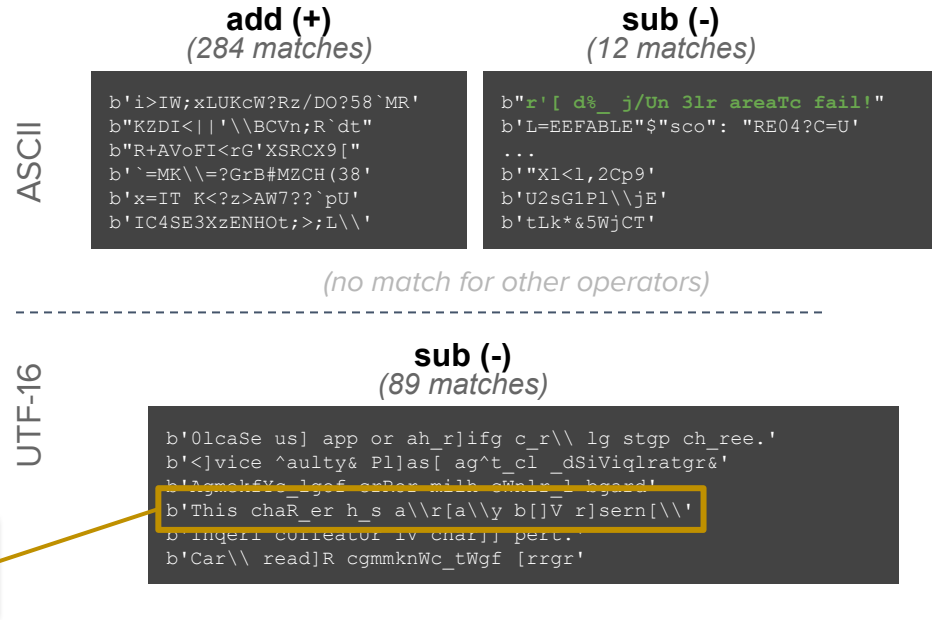
Cryptanalysis: Step #1 (First Attempt)

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



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:

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

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:

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

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
```

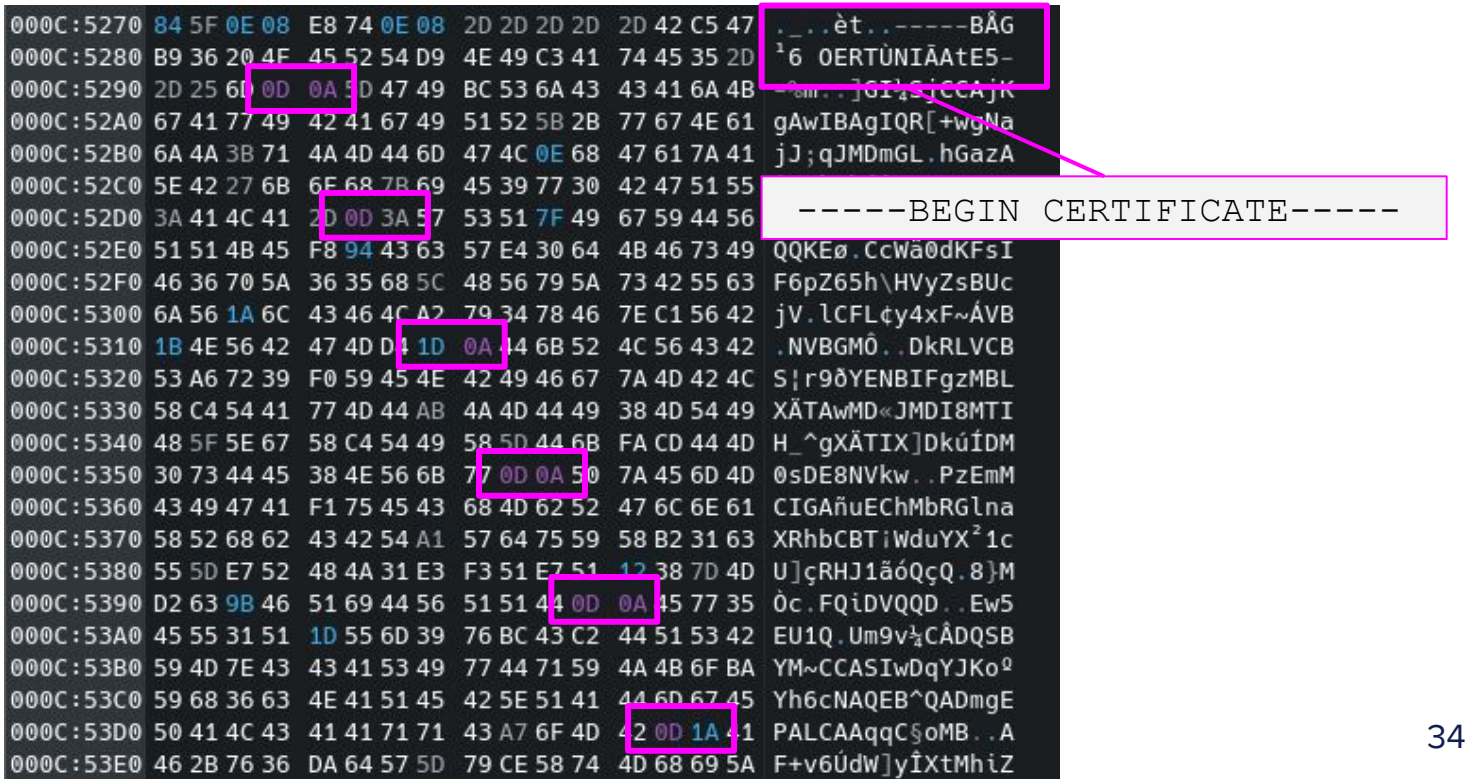
Strings

(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*%l pe2iodDurati[b:=d va\\i'
b'%.2fV, Wu|CR=eR3fA, outOW==>?fA'
b'uaee() case OLD_tCtPAC[AGE`ODD_'
b'T1KKySe97eKnDIr1]vnscFAZMej2Yc'
b'VendI\\Ta_":+&rwb,"strYbg"<."]'
b'CovlonePowernL3: %d;     m_w'
b'Id":REPLACEgCONVECTURID,",i+'
b'1e 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%l,u startSchcdu,e:%ll'
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

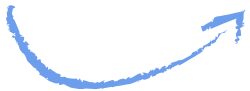


Step #4: Refining plain-ciphered with Certificates

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)
```



```
b'{\t"AuthorizationCacheEnabled":["rw","boolean",&false"],\t"AuthorizeRemoteTxRequests":["rw","boolean","true"],\t"AllowOffljeTxForUnknownId":["rw","boolea
n","false"],\t"ClockAlignedDataInterval":["rw","int","1800"],\t"ConnectionTimeOut":["rw","int","30"],\t"GetConfigurationMaxKeys":["r","int","10"],\t"Heartbe
atInterval":["rw.,"int","14400"],\t"LocalAuthorizeOffline":["rw","boolean","true"],\t"LocalPreAuthorize":["rw","boolean","true"],\t"MeterValuesAlignedData":
]"rw","string","Energy.Active.Import.Register,Voltage,Current.E'
b'port,Power.Offered,Current.Import,Powev.Active.Import"],\t"MeterValuesSampledData":["rw","string","Energy.Active.Import.Register,Voltage,Current.Export,Po
wer.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 nemoteId:%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 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 RemateId:%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
;StopChargingDevOount %d'
b'{"cmd":"RANDOM_DELAY_TIME","seq": "REPLACE_SEQ" ,"data"Z     {"pileNum":"REPLACE_SN", "connectsrId":REPLACE_CONJECTORID,"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,PlugtsCharge:%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 ; outPowerT %d; outCurrR: %d ; outCurrS %d ; outCurrT %d  ?
outVoltR: %d ; kutVoltS %d ; outVoltT %d ; gridVoltR: %d ; gridVoltS %d ; 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 sucess,m_inDlbGroup %d , m_DevRule %d , m_DlbOnOff %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_ojlinePower %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 startPerkodPos:%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'@WWJ 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 recuvrencyStepLen:%d dstEndPeriod:%d recurrencyIndex:%d (*it) startPeriod:%d duration:%d
periodDuration:%d validDuration:%d'
b' <%s:%u>: retF%d dst:%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 pkwer:%d; cur power:%lld ;gridIsSinglePhase:%d;
```

Step #5: Refining with Hex

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**

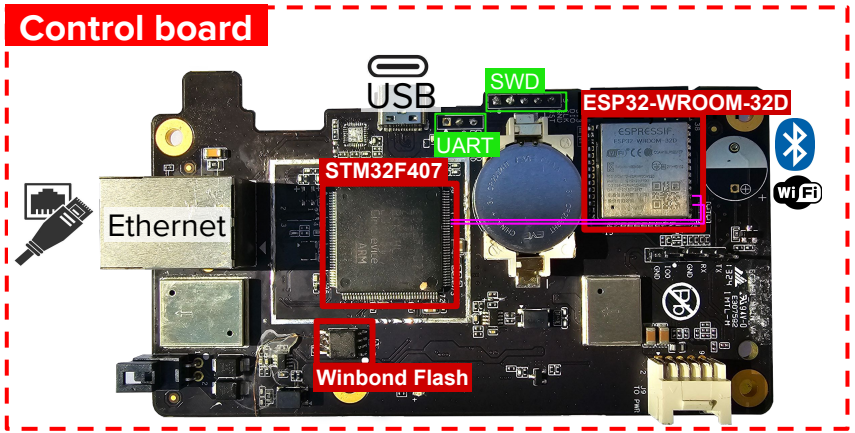
⇒ Finally found the two key parts making the whole firmware valid (with ADD | XOR)

(Turns out the real algorithm was a bit different with a key and a constant [↗])

Vulnerability Research

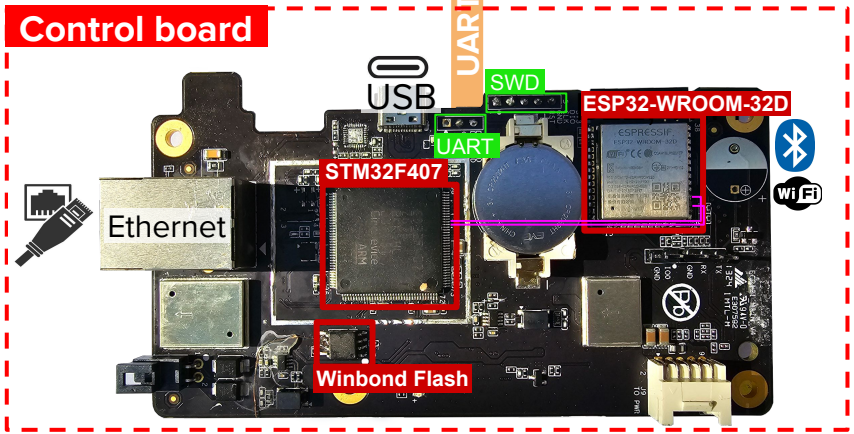
Quarkslab

Hardware Overview



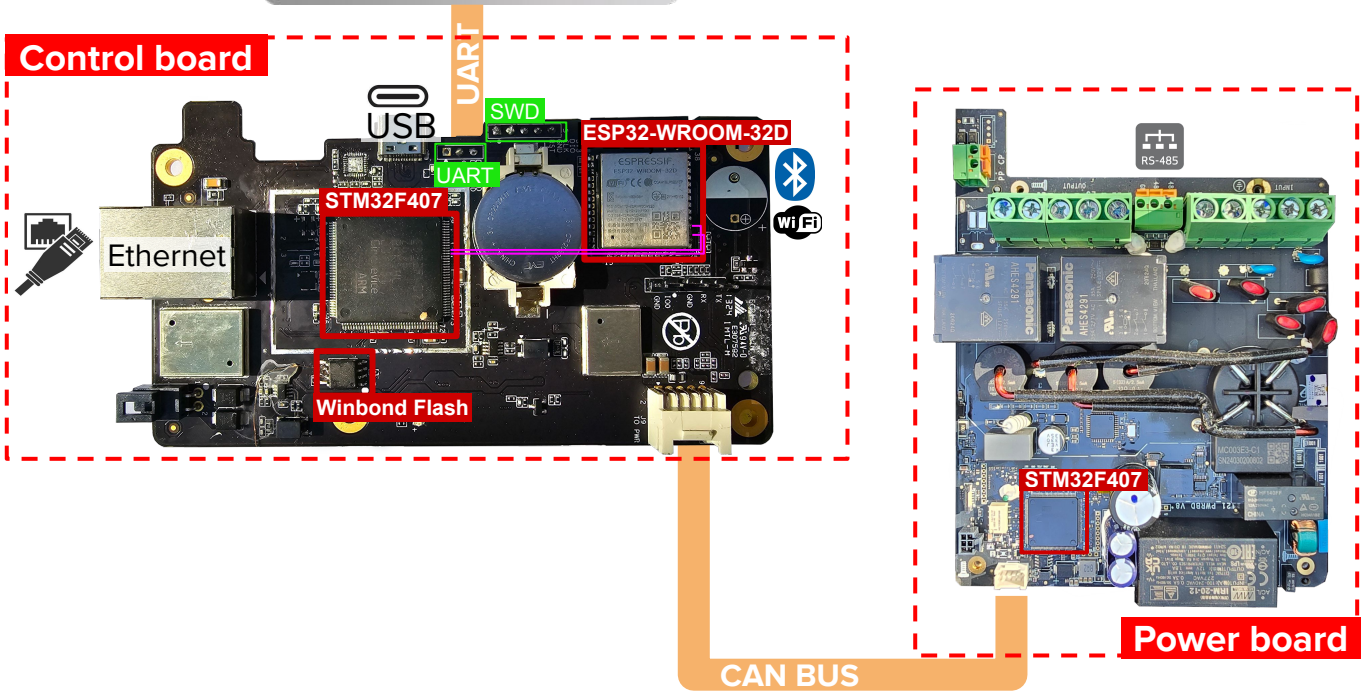
Hardware Overview

Levetop
LT7689



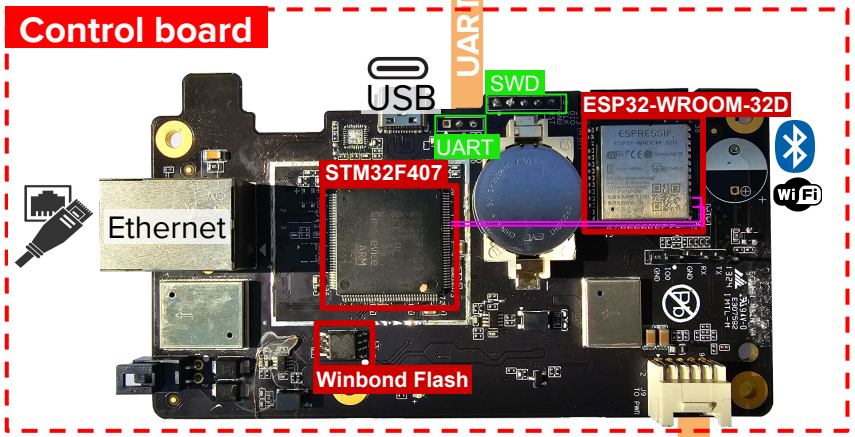
Hardware Overview

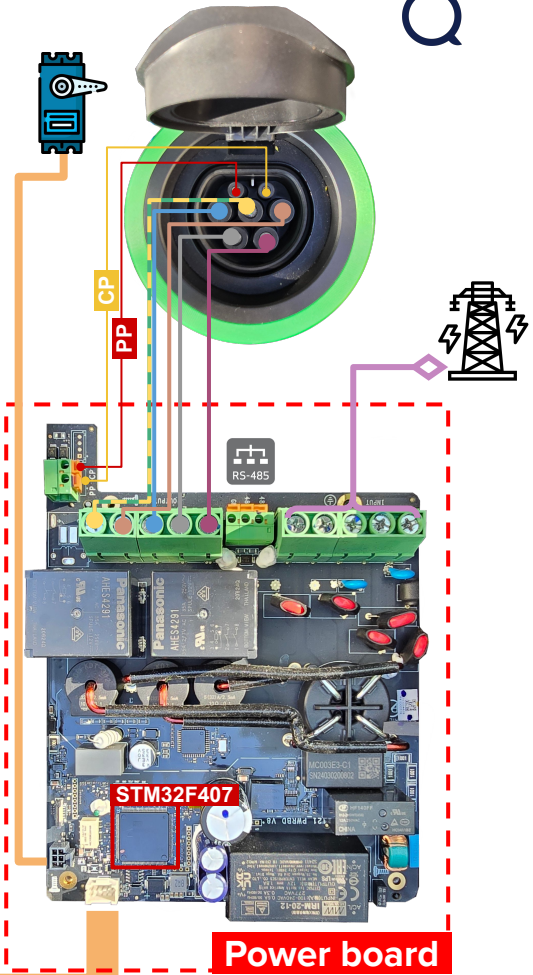
Levetop
LT7689



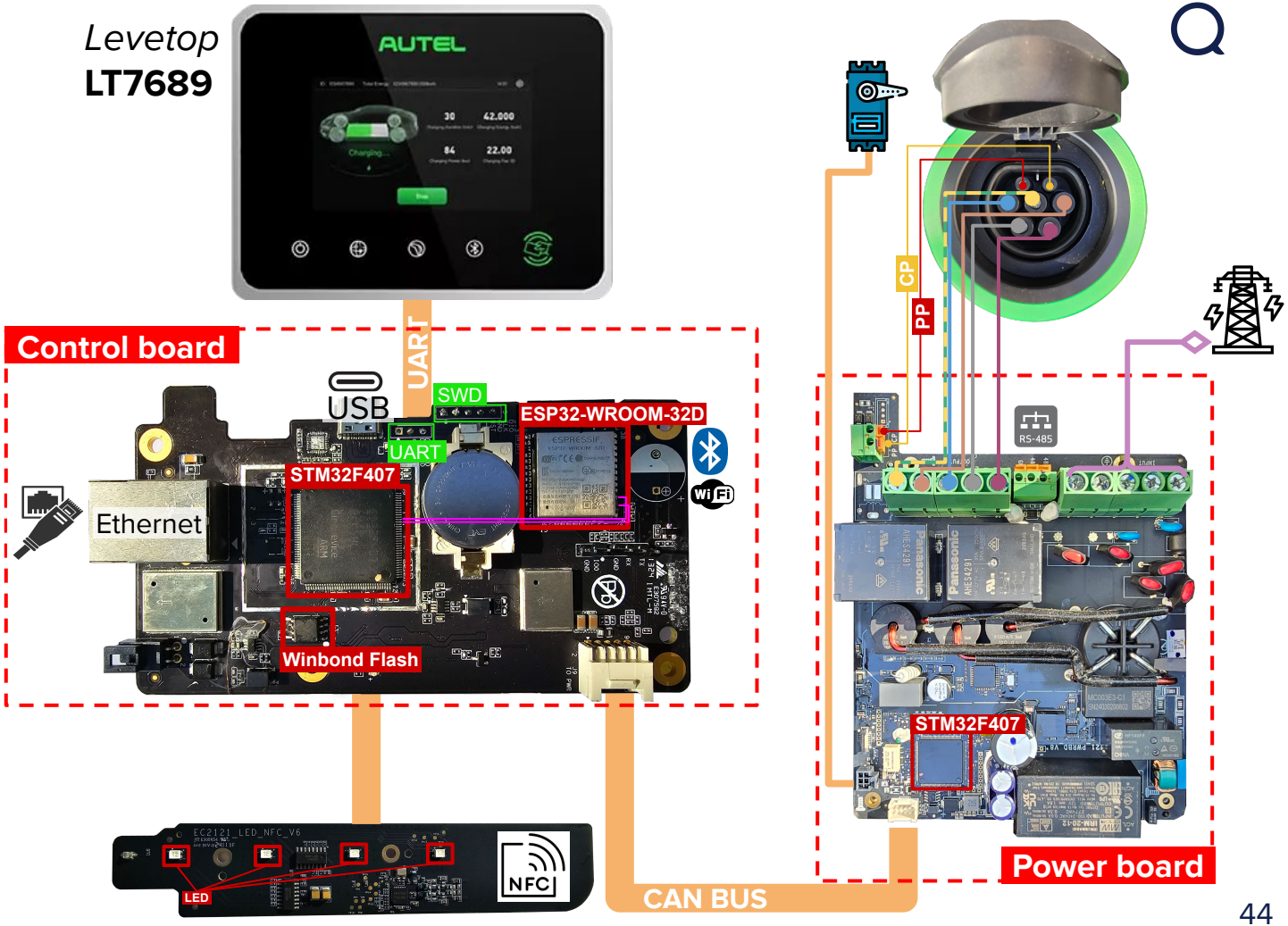
Hardware Overview

Levetop
LT7689





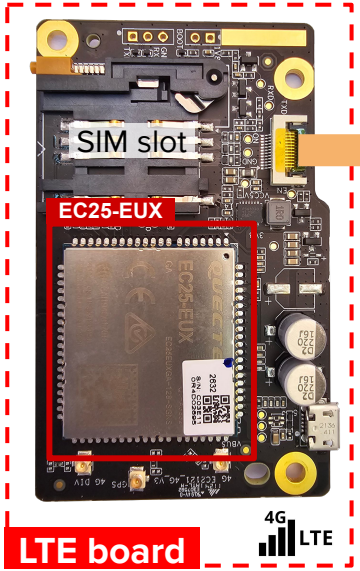
Hardware Overview



Hardware Overview

Levetop
LT7689

UART



LTE board

UART

Control board

USB

SWD

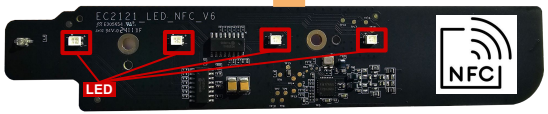
UART

ESP32-WROOM-32D

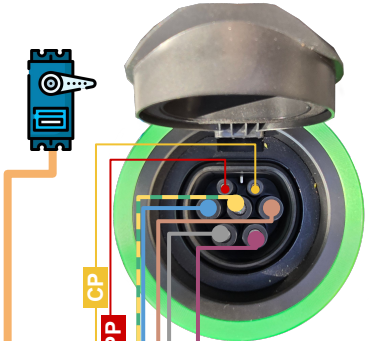
STM32F407

Ethernet

Winbond Flash

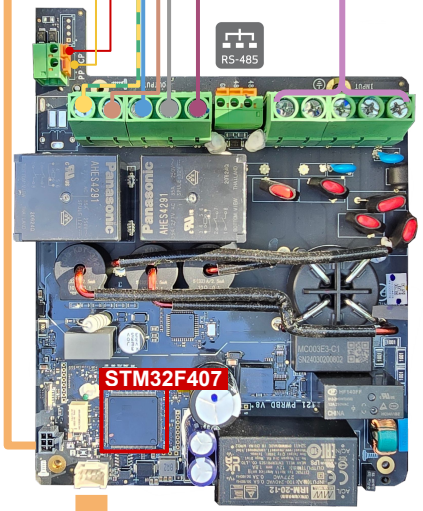


CAN BUS



CP

PP



Power board

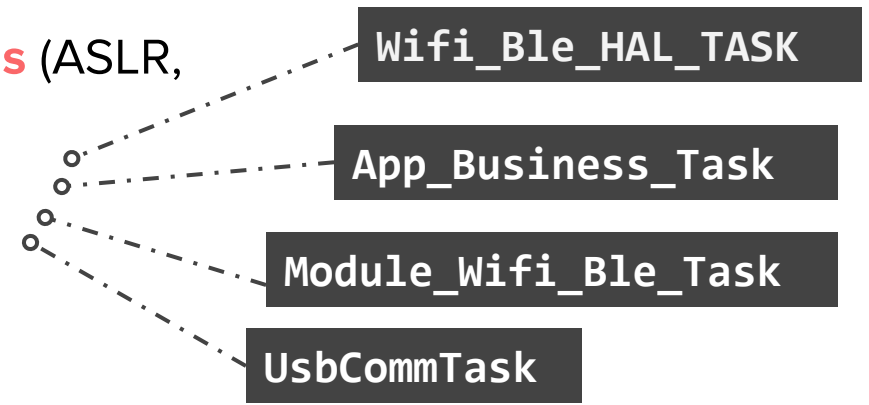
Firmware analysis

Recon:

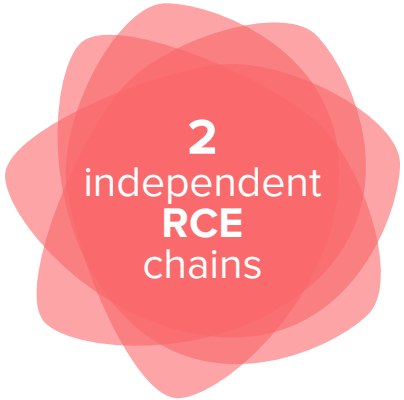
- Loading address `0x08010000`
  (bootloader not part of FOTA?, secureboot?)

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

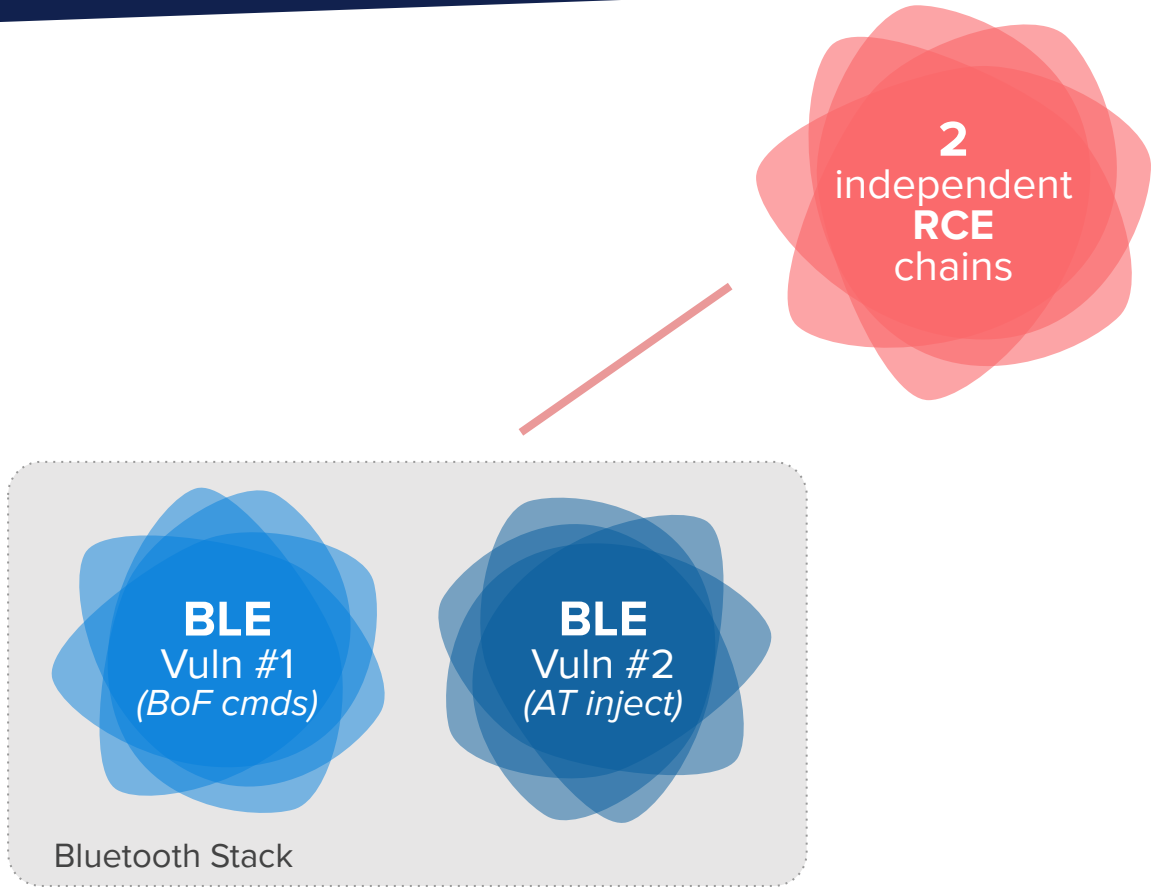
Used internal tool for function similarity ⇒ *Recognized all known FreeRTOS functions*



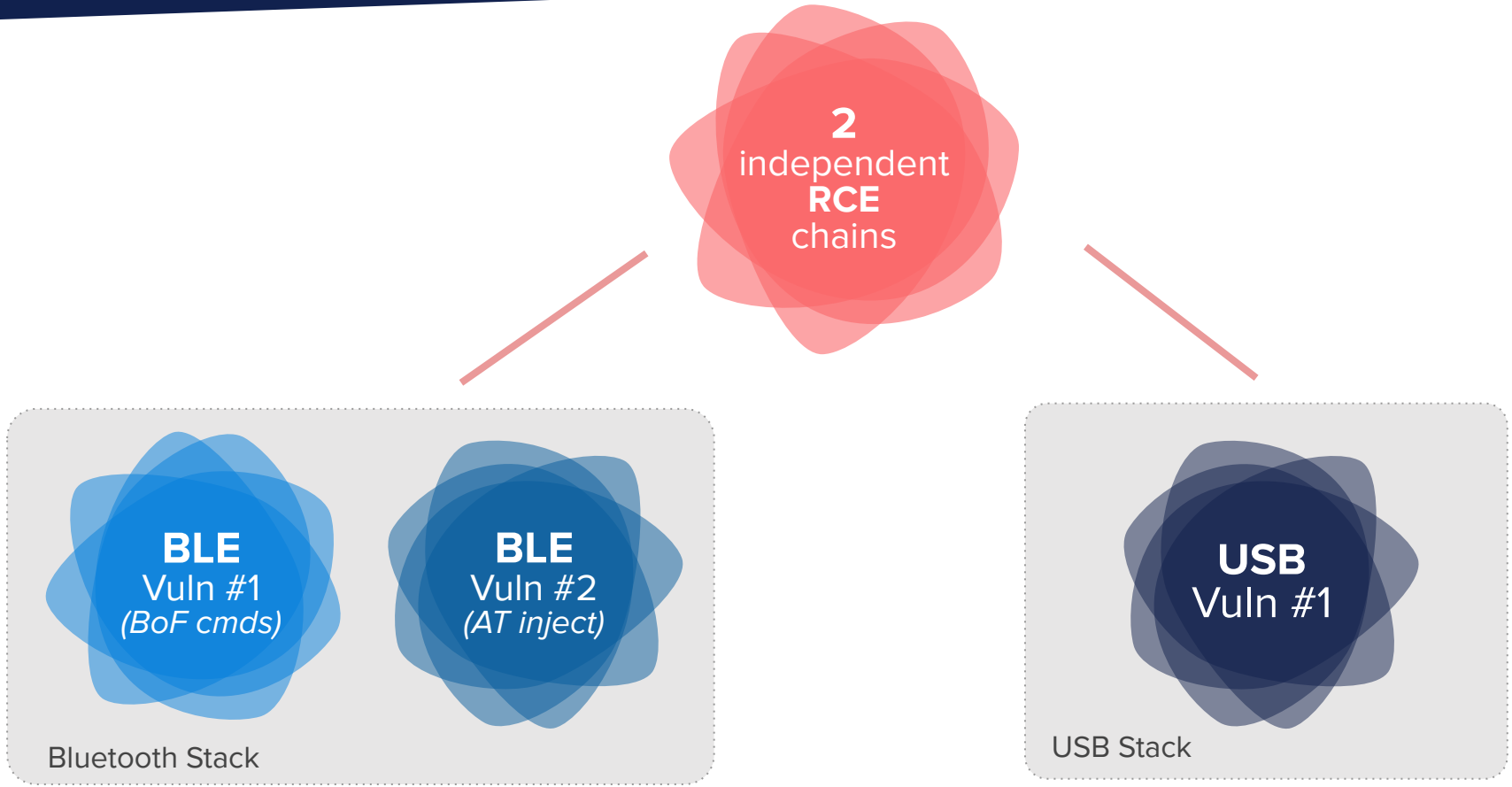
Vulnerabilities



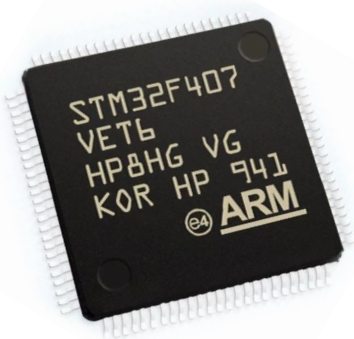
Vulnerabilities



Vulnerabilities



Bluetooth



AT commands

BLE messages

Translate BLE message into AT commands:

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

size

```
+WRITE:0,0,0,0,4,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

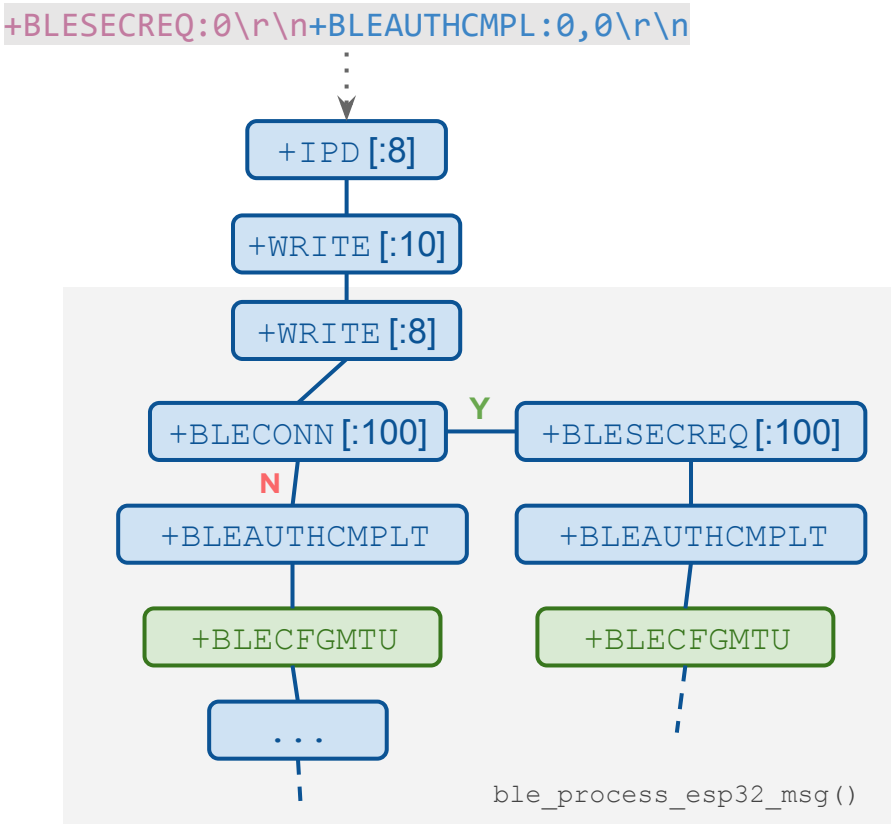
Goal: Trying to see if we can **forge** a message that would result in a BLECFGMTU command

Chain #1 - Buffered AT command parsing

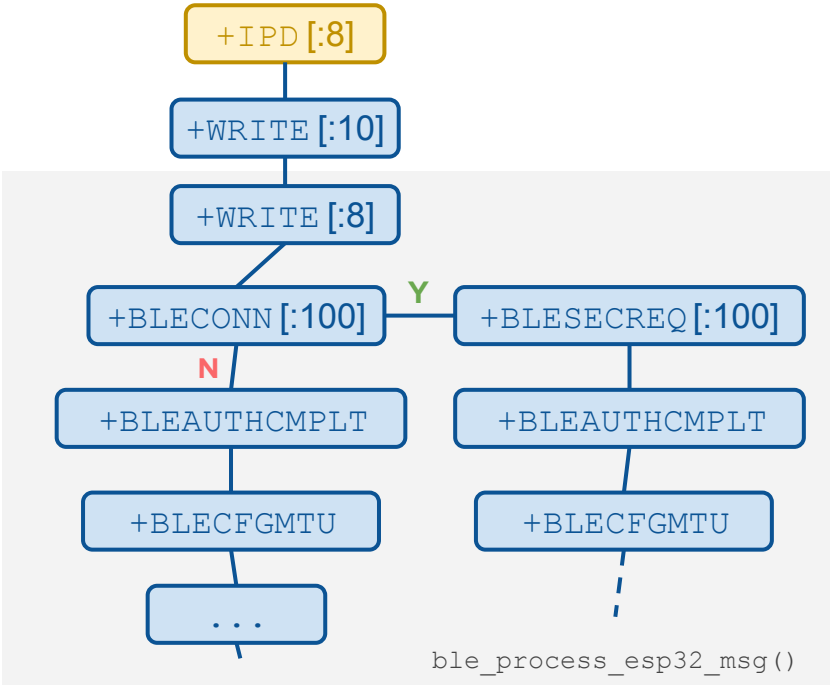
Observations:

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

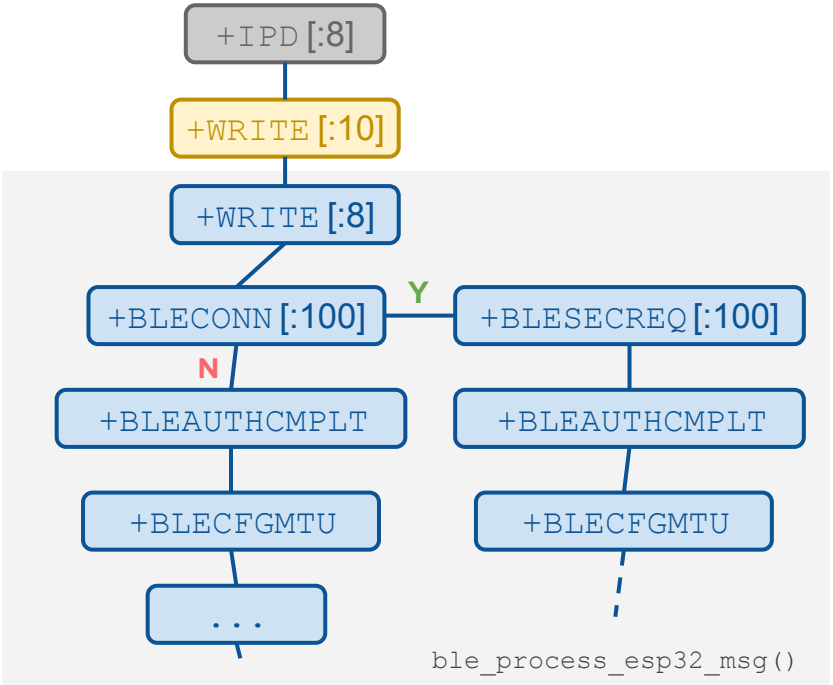
Can reliably force the ESP32 to send two commands in the same message.



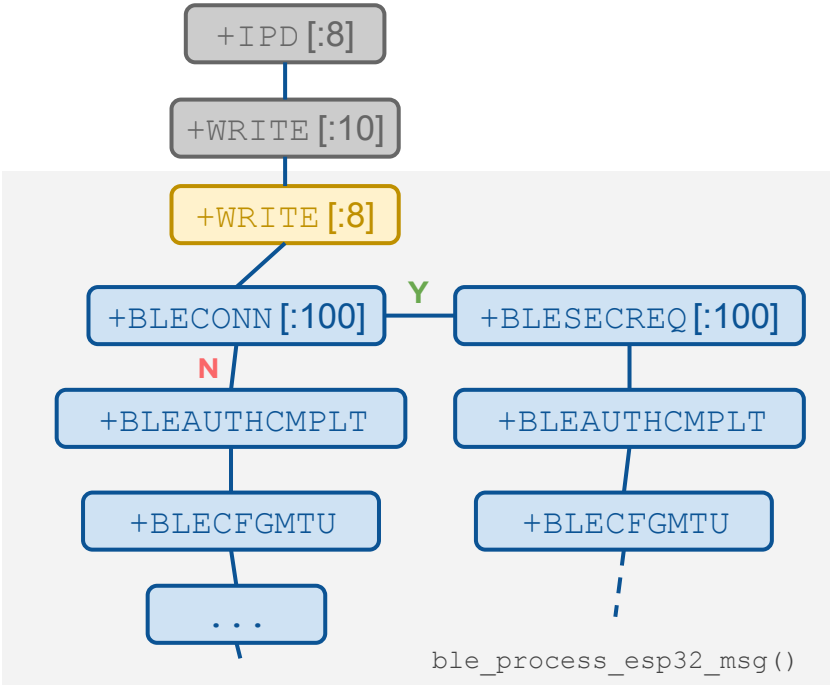
Chain #1 - Vuln #2: AT command injection



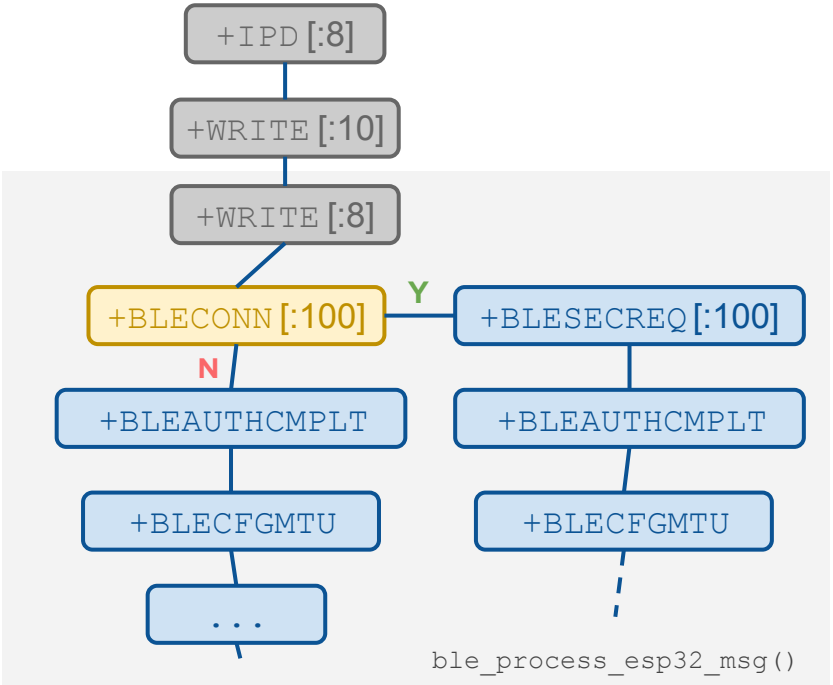
Chain #1 - Vuln #2: AT command injection



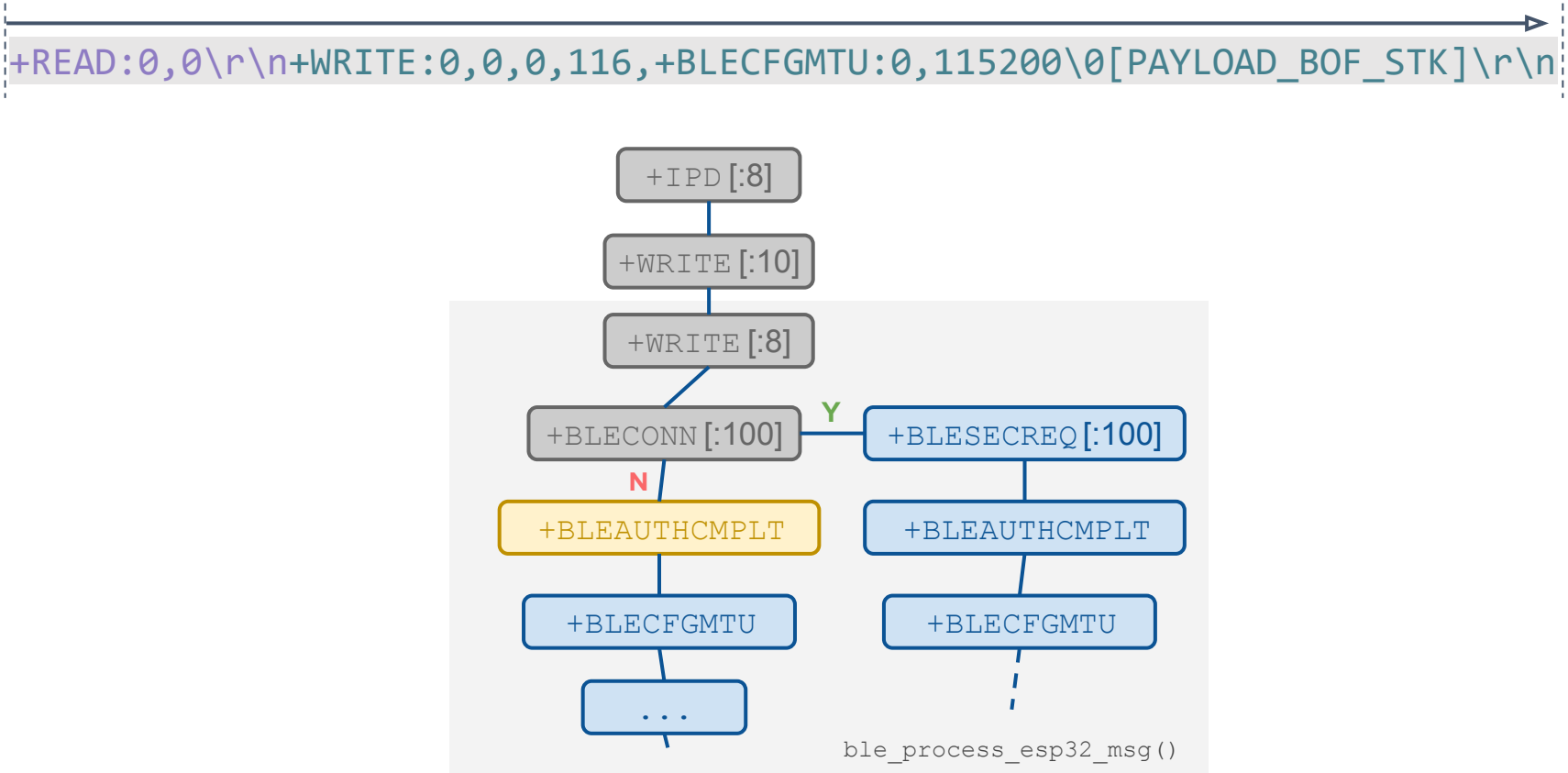
Chain #1 - Vuln #2: AT command injection



Chain #1 - Vuln #2: AT command injection



Chain #1 - Vuln #2: AT command injection

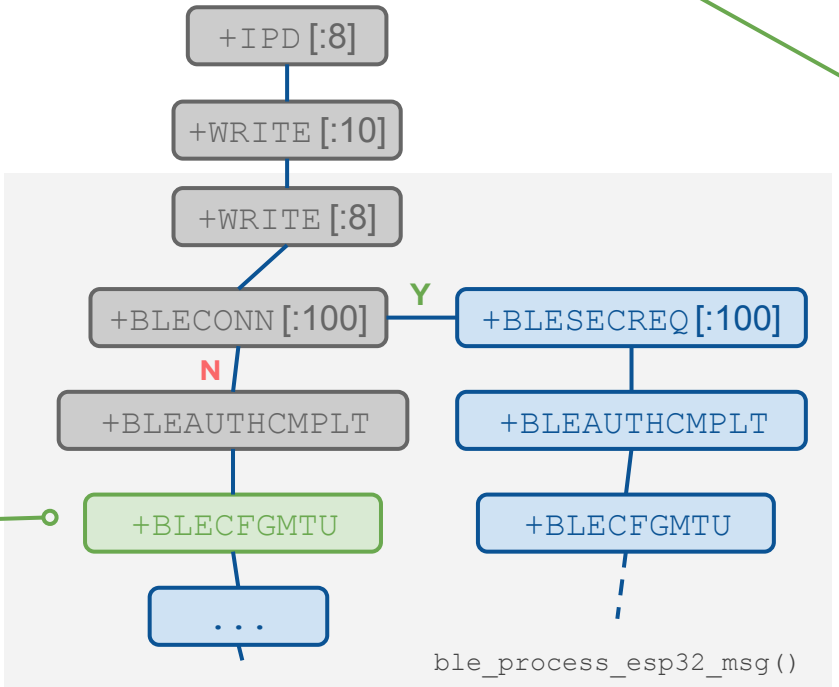


Chain #1 - Vuln #2: AT command injection

found !

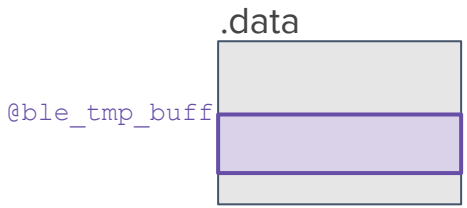
```
+READ:0,0\r\n+WRITE:0,0,0,116,+BLECFGMTU:0,115200\0[PAYLOAD_BOF_STK]\r\n
```

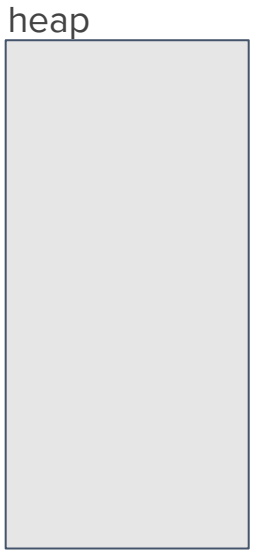
`WRITE` arguments improperly considered as a command !



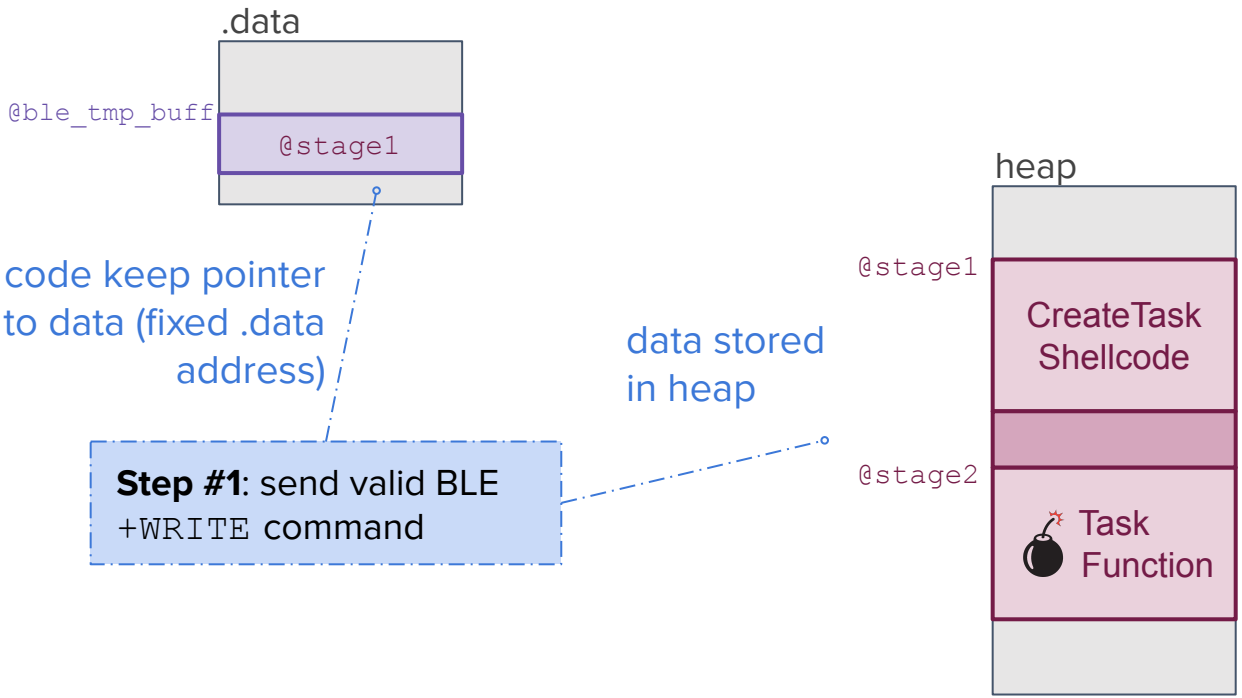
Call the command handler with injected command and trigger stack BoF !

Chain #1 - Full Exploit

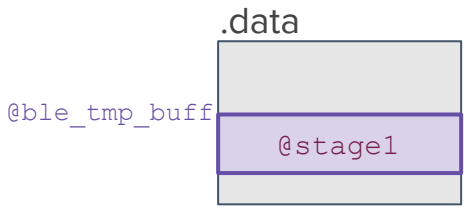




Chain #1 - Full Exploit



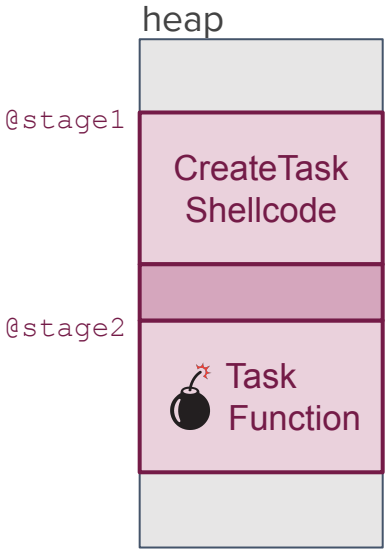
Chain #1 - Full Exploit



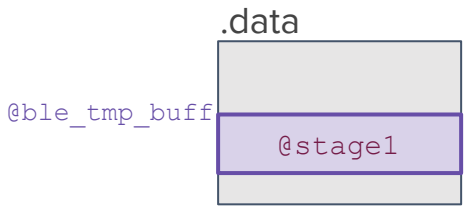
stack frame

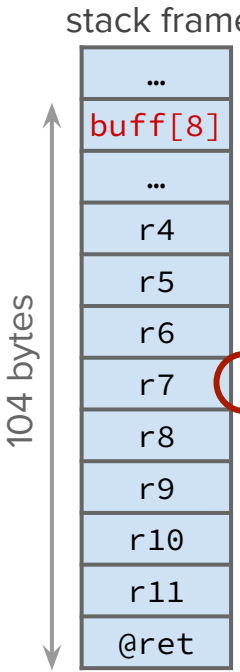
…
buff[8]
…
r4
r5
r6
r7
r8
r9
r10
r11
@ret

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



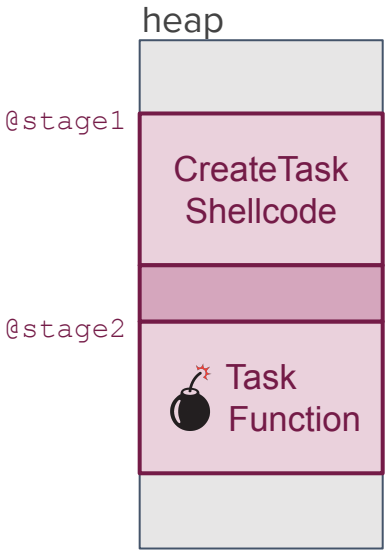
Chain #1 - Full Exploit



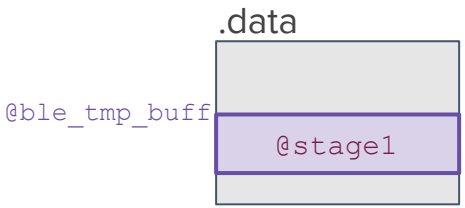


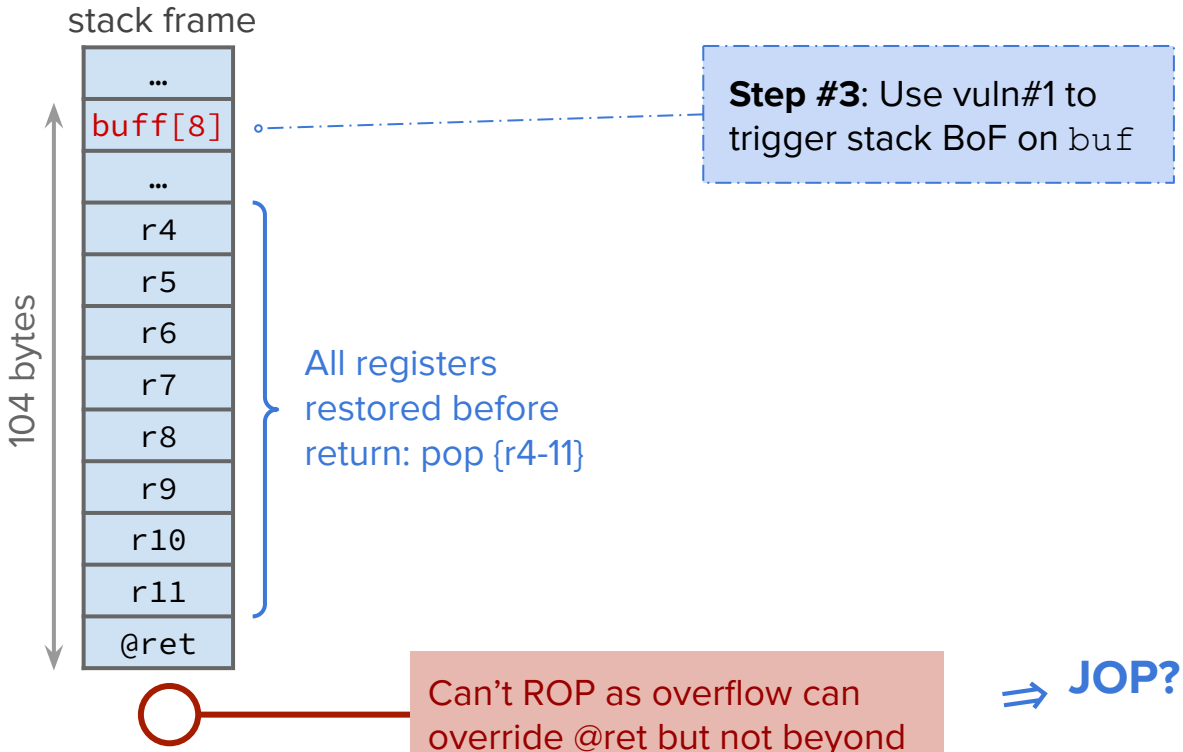
Step #3: Use vuln#1 to trigger stack BoF on `buf`

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

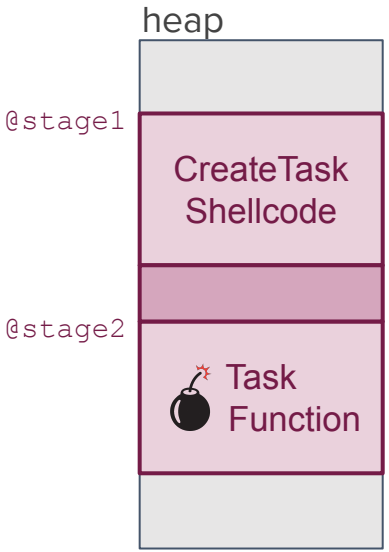


Chain #1 - Full Exploit

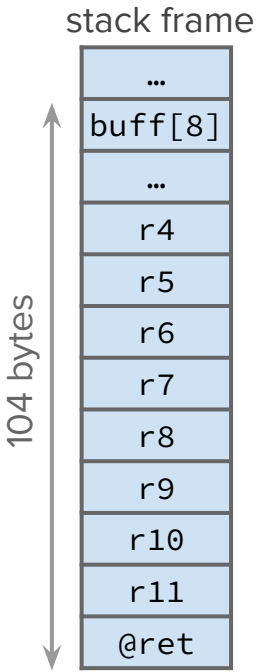


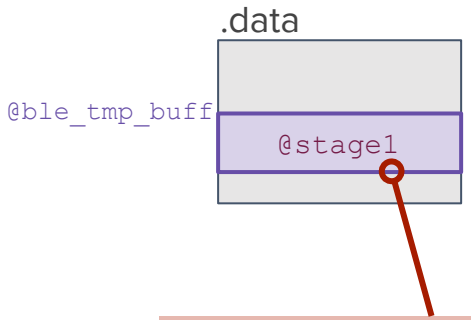


⇒ **JOP?**



Chain #1 - Full Exploit

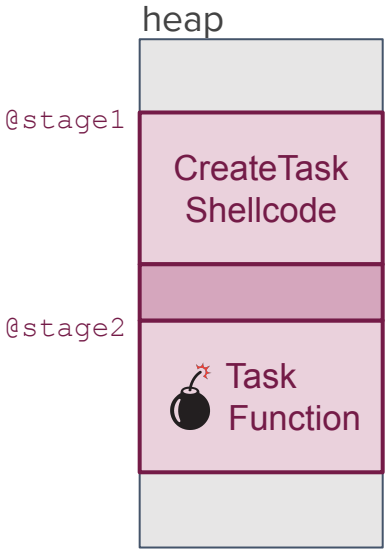




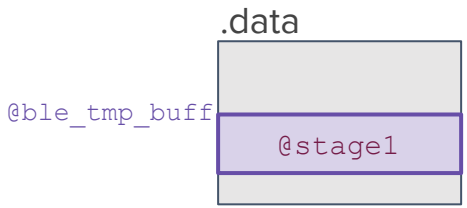
@stage1 points to data, in THUMB, LSB should be set to jump on code

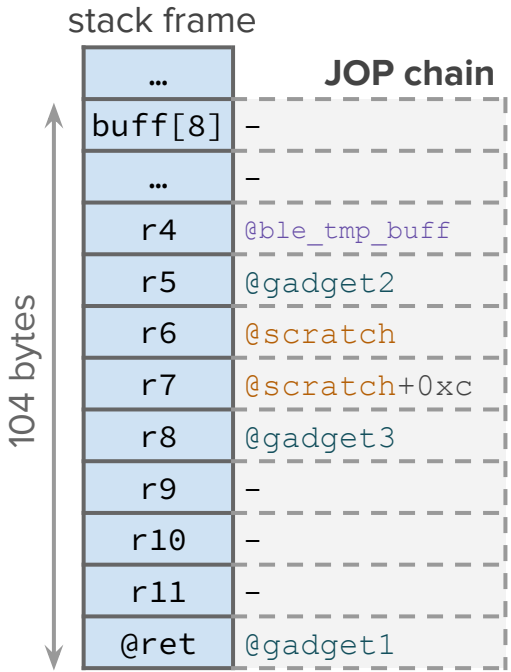
⇓

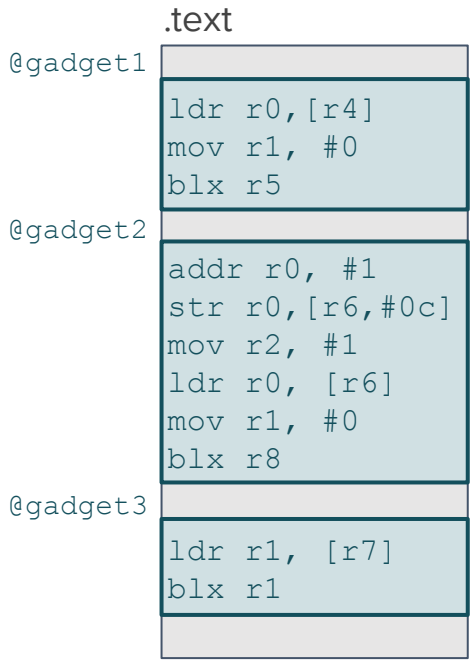
Step #4: Write a JOP chain to jump on @stage1

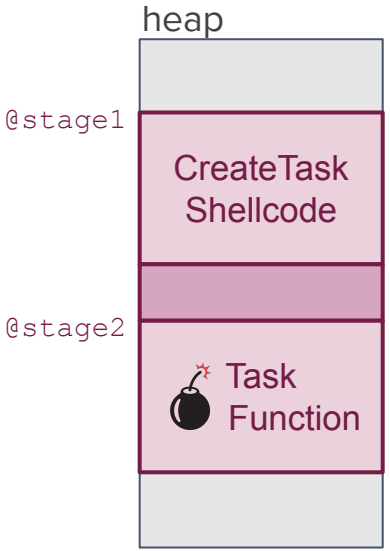


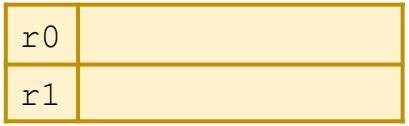
Chain #1 - Full Exploit





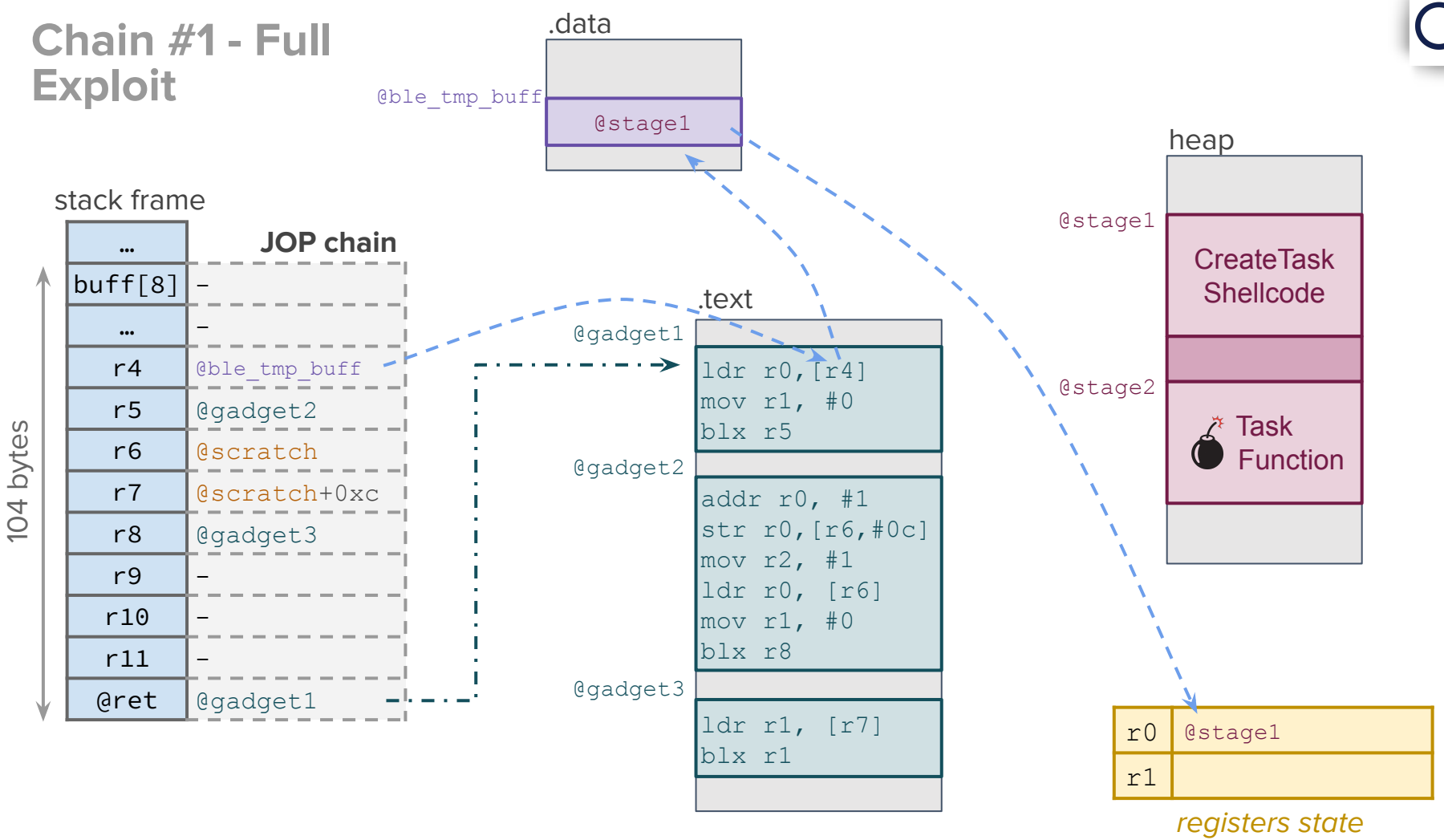




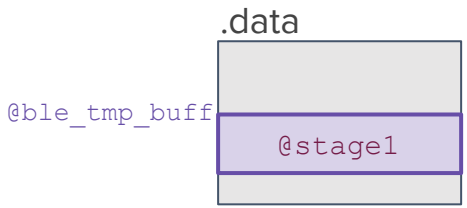


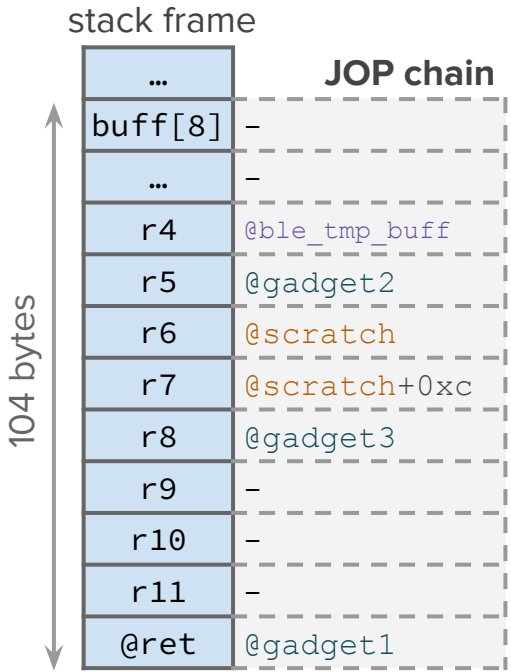
registers state

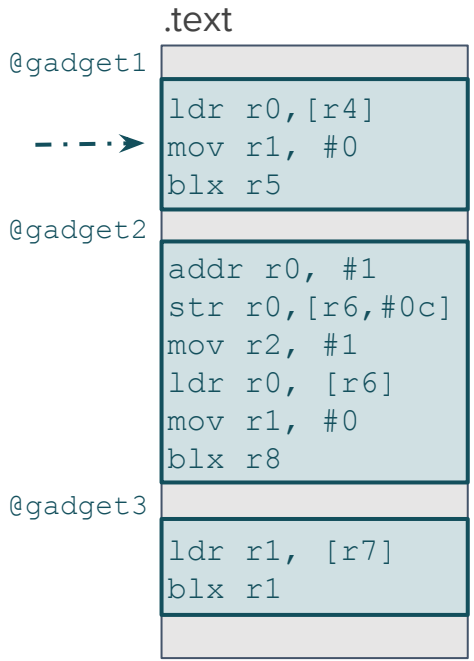
Chain #1 - Full Exploit

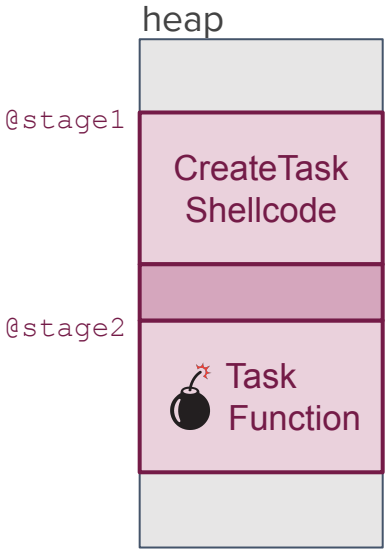


Chain #1 - Full Exploit





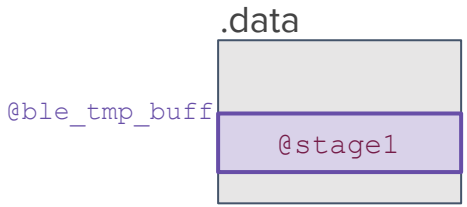


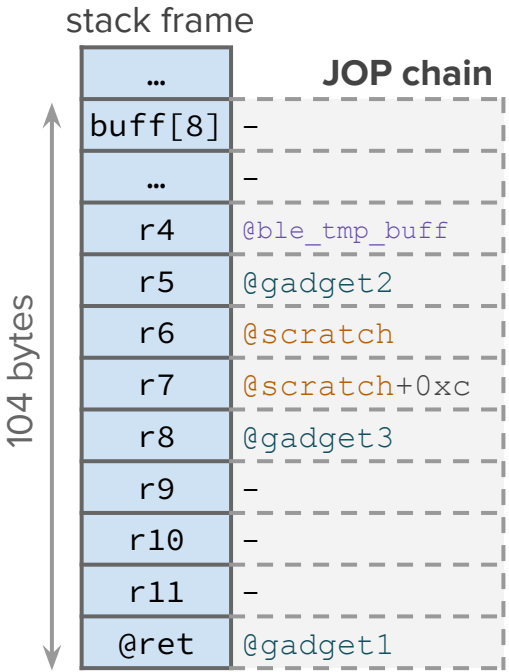


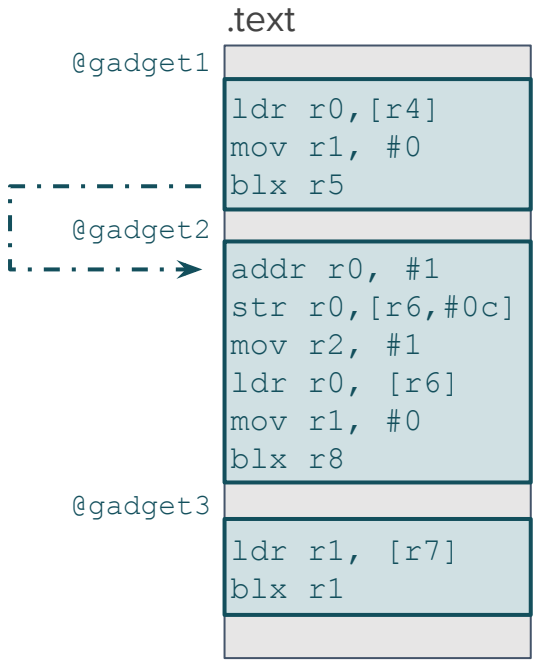
r0	@stage1
r1	#0

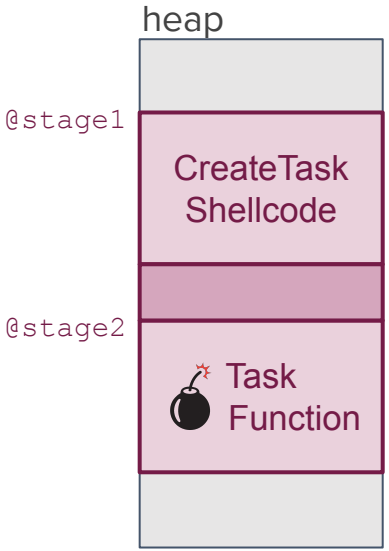
registers state

Chain #1 - Full Exploit





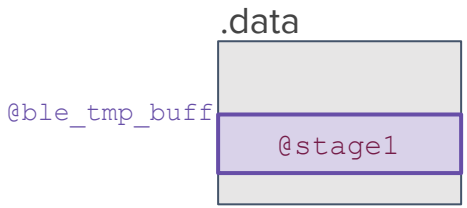


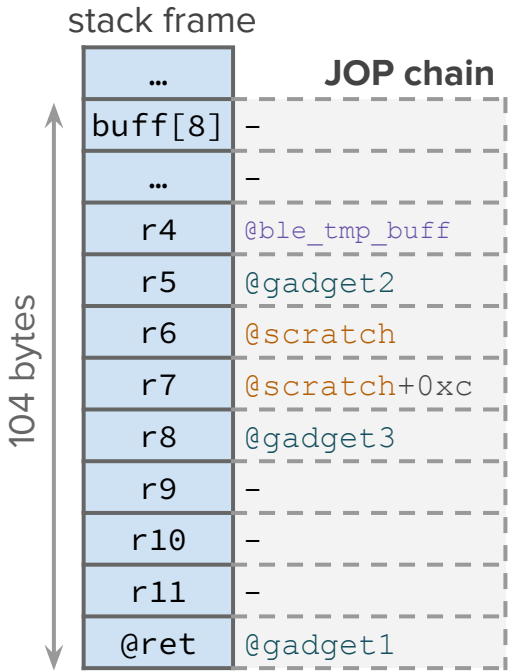


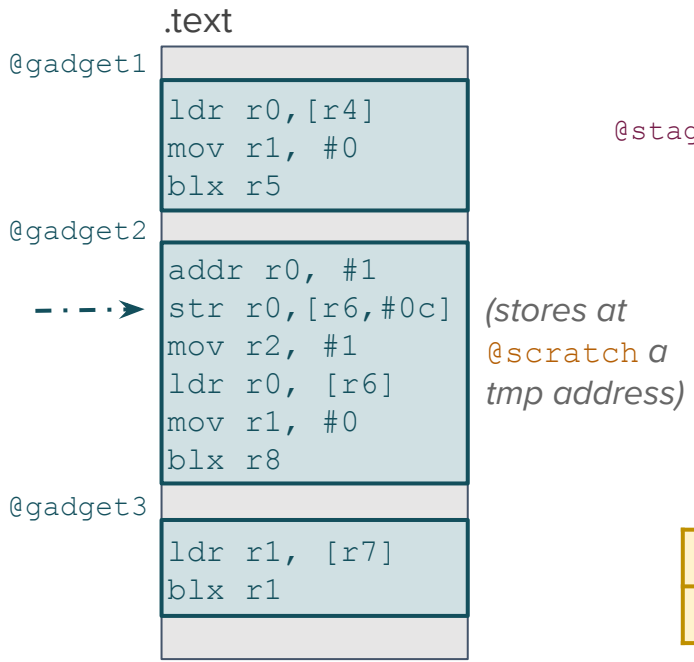
r0	@stage1 + 1
r1	#0

registers state

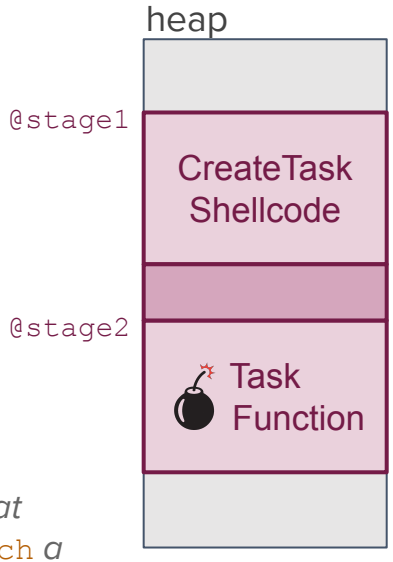
Chain #1 - Full Exploit







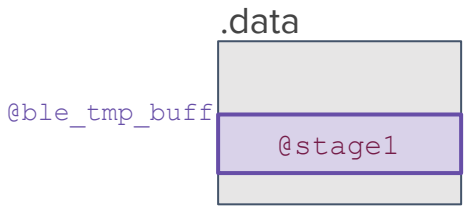
(stores at `@scratch` a tmp address)

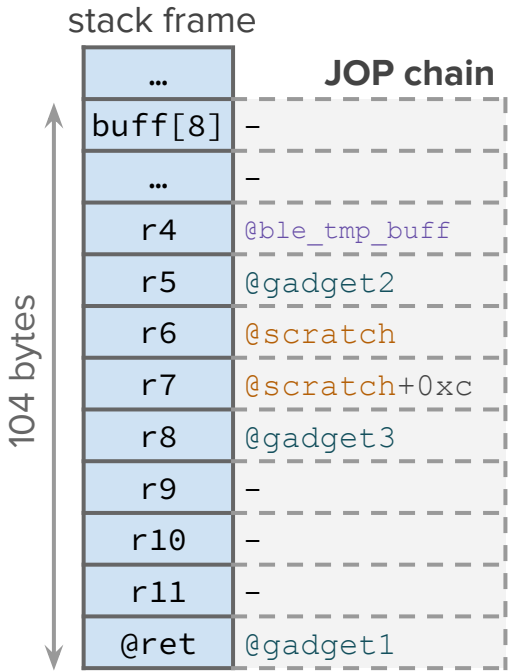


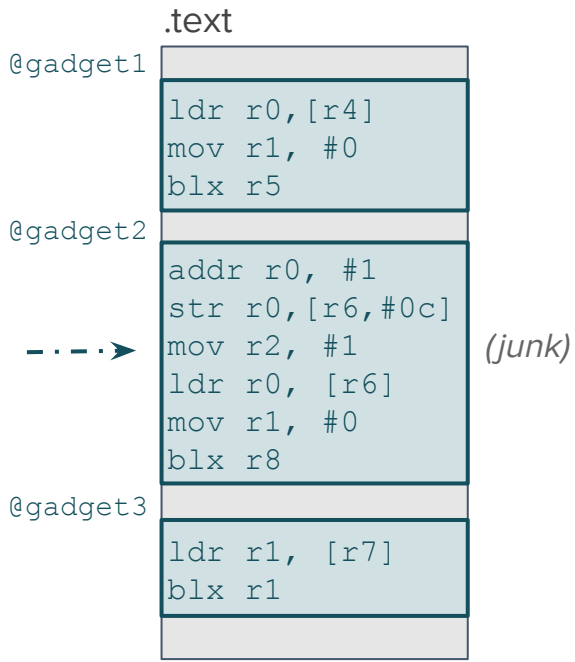
r0	@stage1 + 1
r1	#0

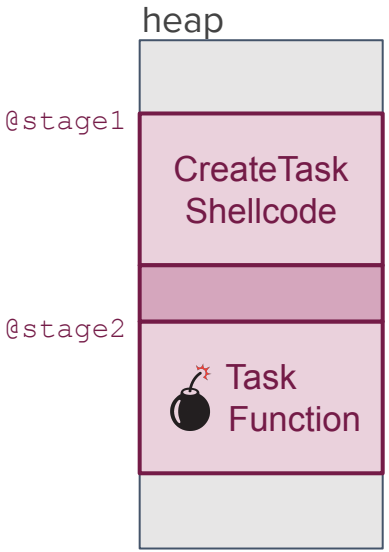
registers state

Chain #1 - Full Exploit





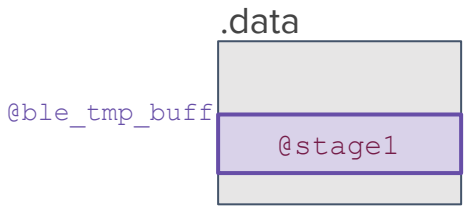


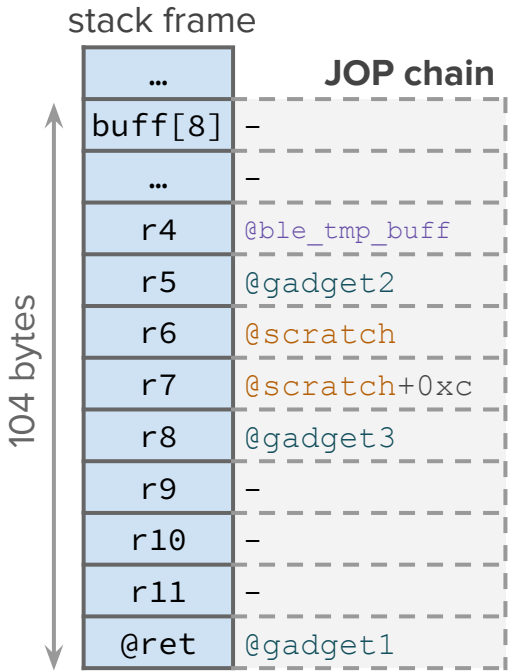


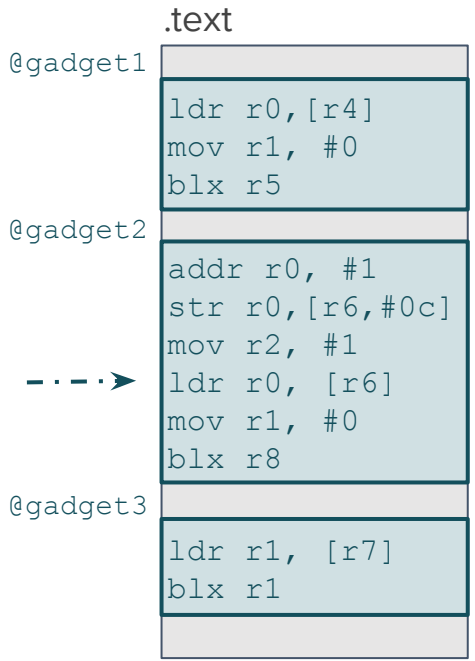
r0	@stage1 + 1
r1	#0

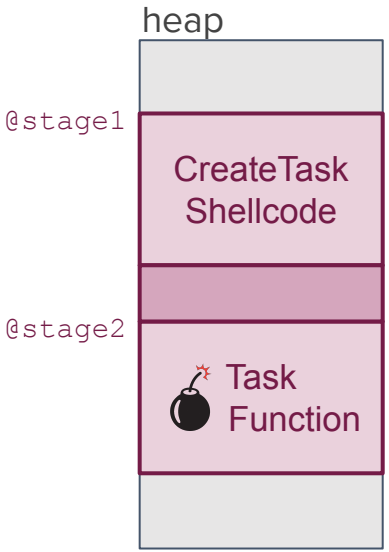
registers state

Chain #1 - Full Exploit







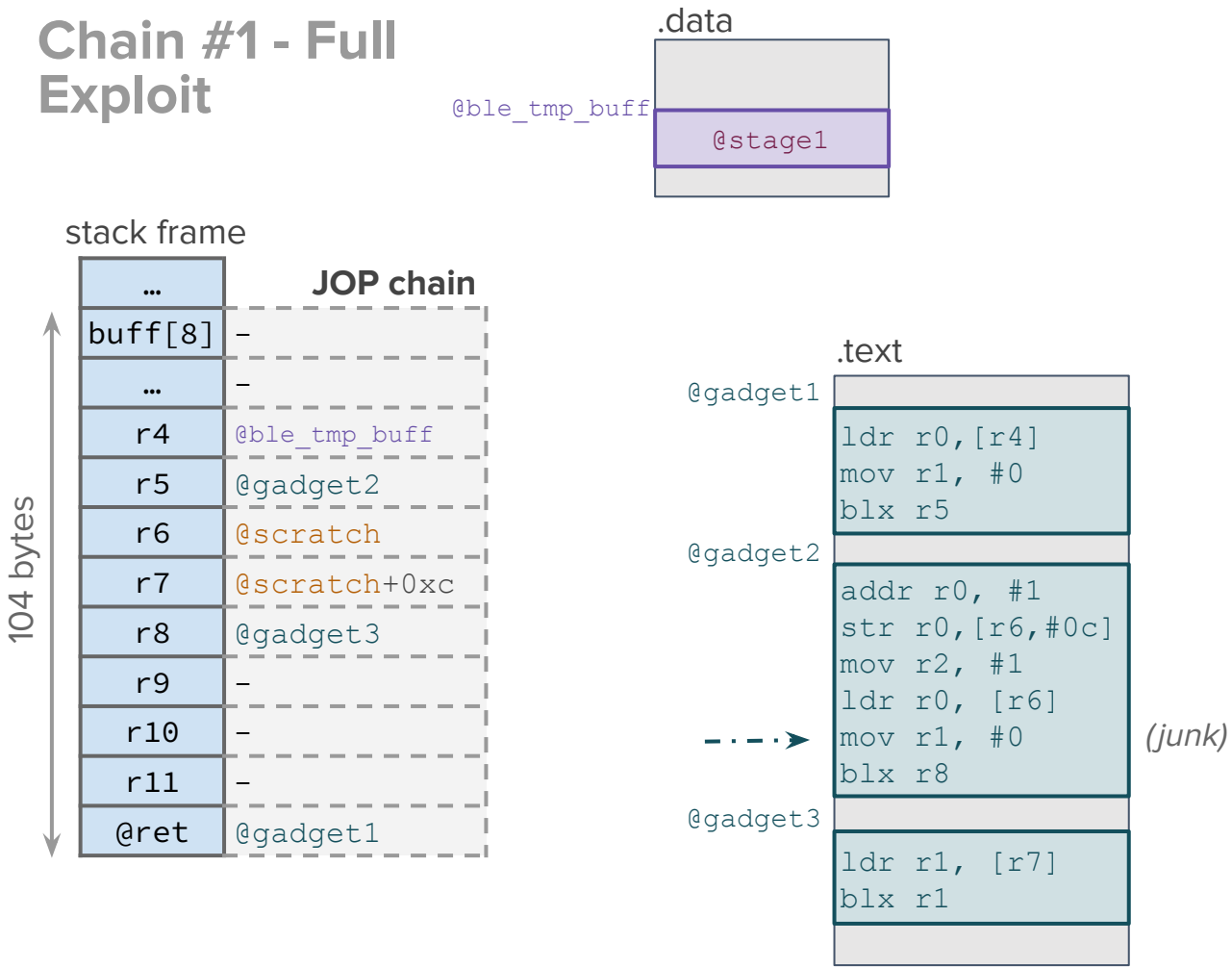


(erase our address!)

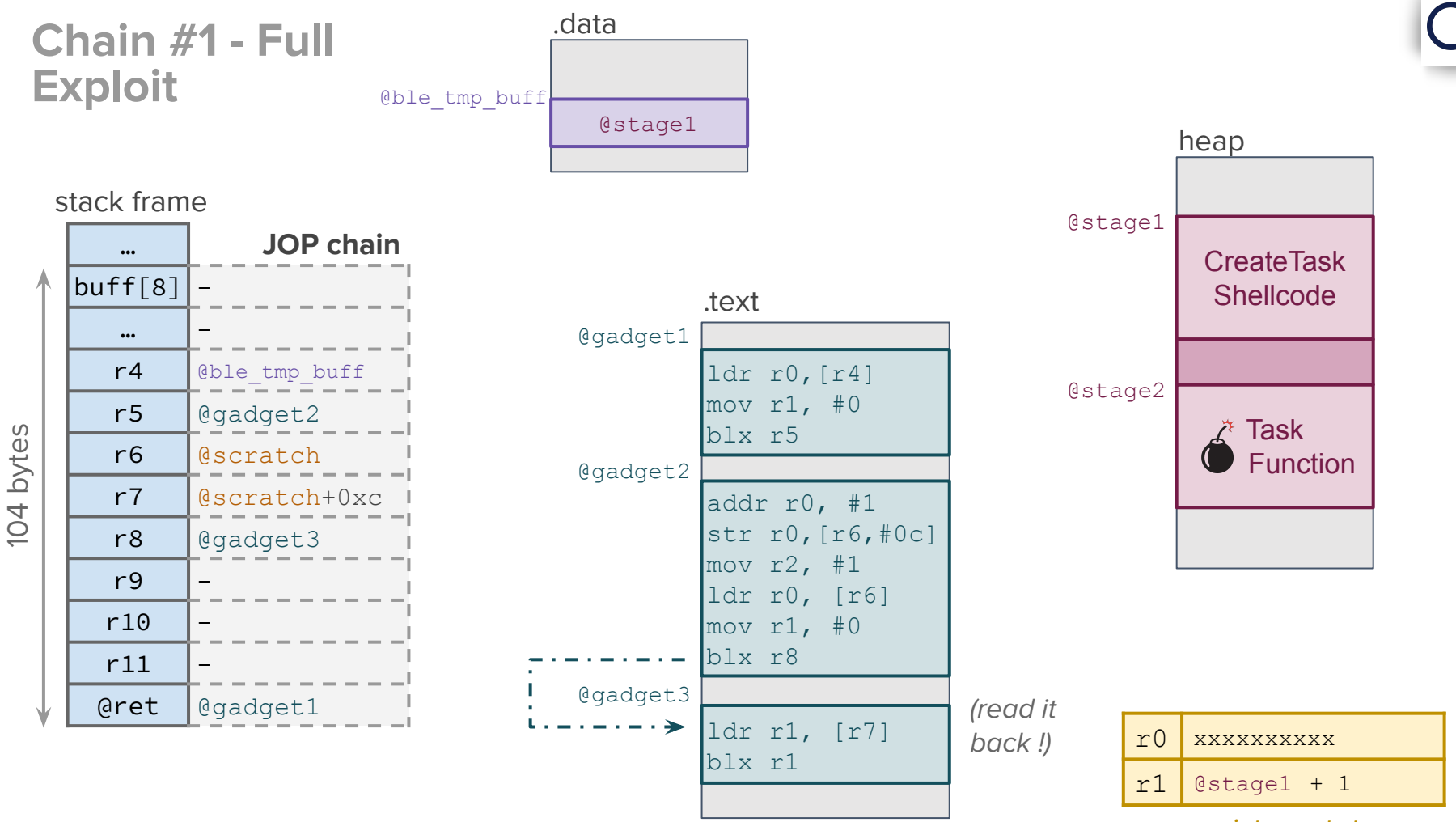
r0	xxxxxxxxxx
r1	#0

registers state

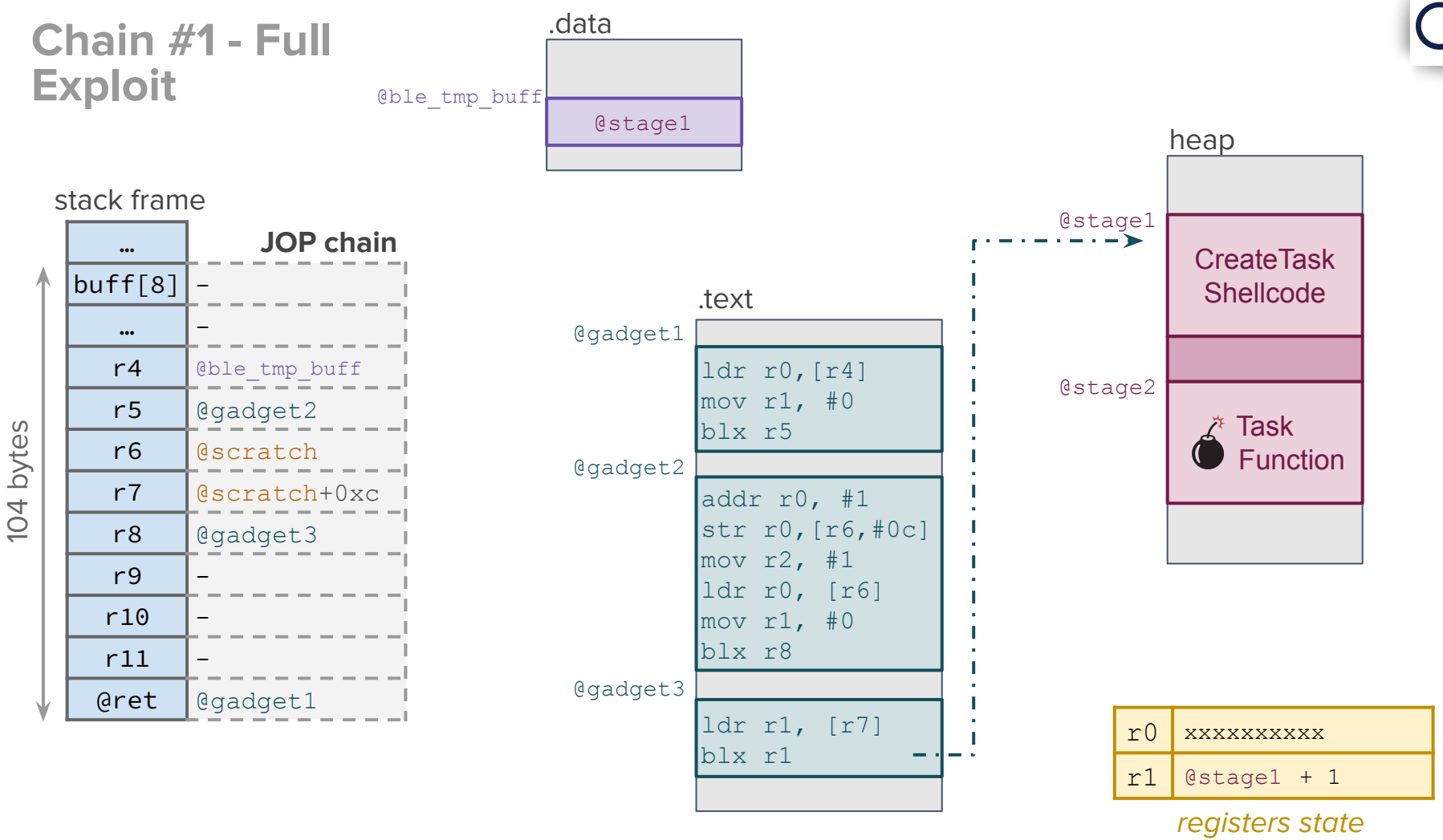
Chain #1 - Full Exploit



Chain #1 - Full Exploit



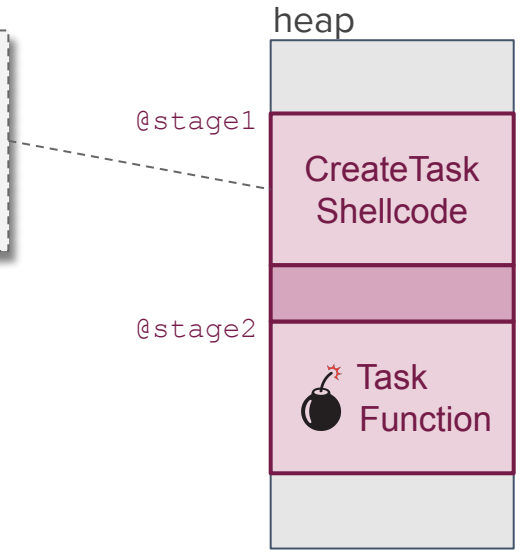
Chain #1 - Full Exploit



Chain #1 - Full Exploit

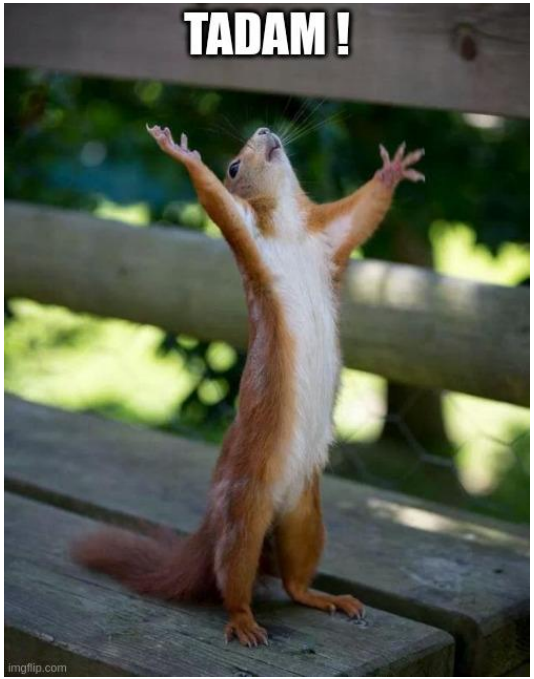
1. Calls FreeRTOS: `xTaskCreate(@stage2, "exploit", …)`
2. Returns where BLECFGMTU function should have

⇒ **Execution flow restored properly**

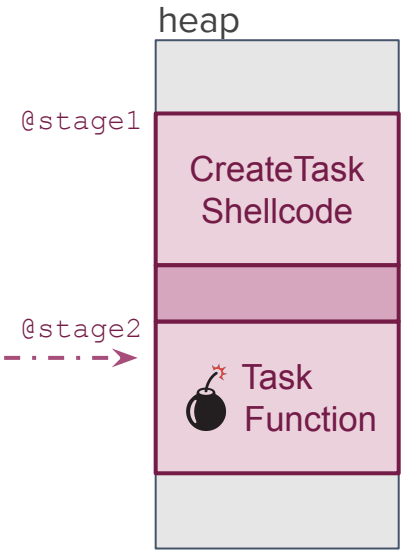


(In reality the task code is copied at another location than the heap that might be overridden).

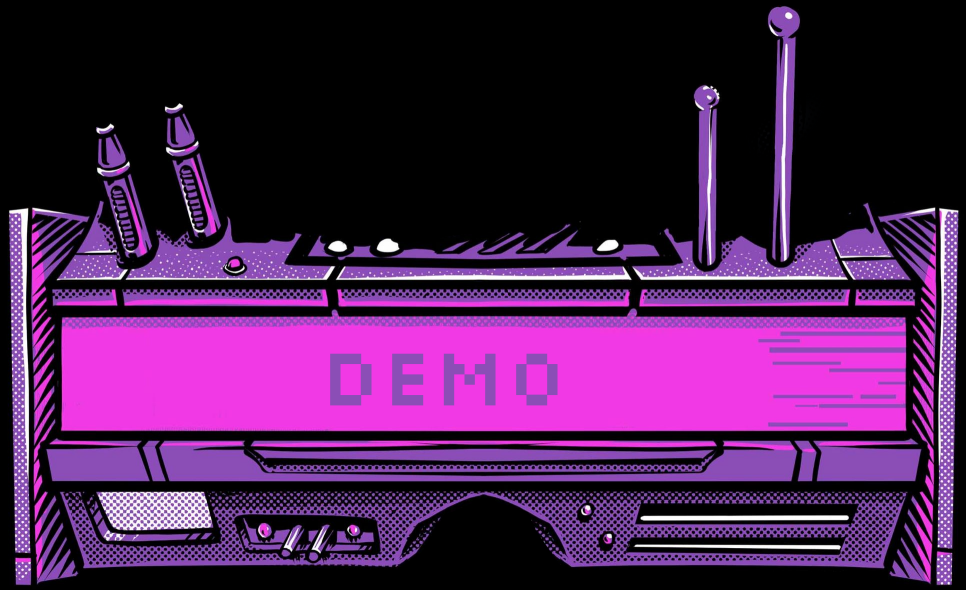
Chain #1 - Full Exploit



FreeRTOS creates the task and jumps on the final stage



+ 100% stable
+ do not cause any service interruption



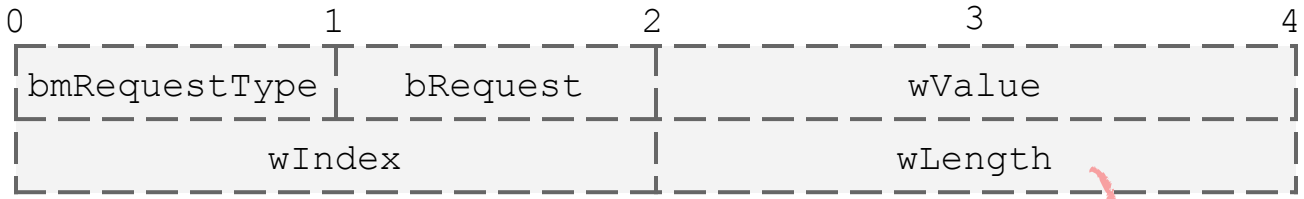
Quarkslab

Chain #2: USB

⇒ RE **uneasy** *(plenty callbacks function pointers)* but likely based on an **OSS library**

(libusb_stm32, stm32-usbx)

Header



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

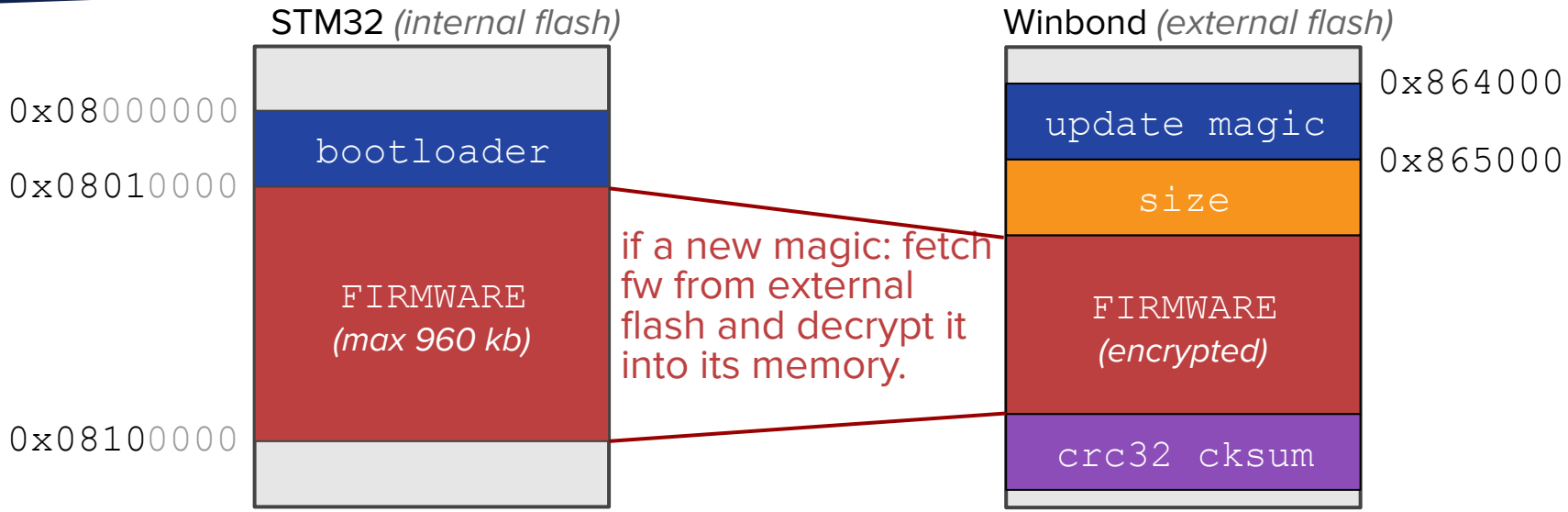
Device is **compromised**

and what now ?

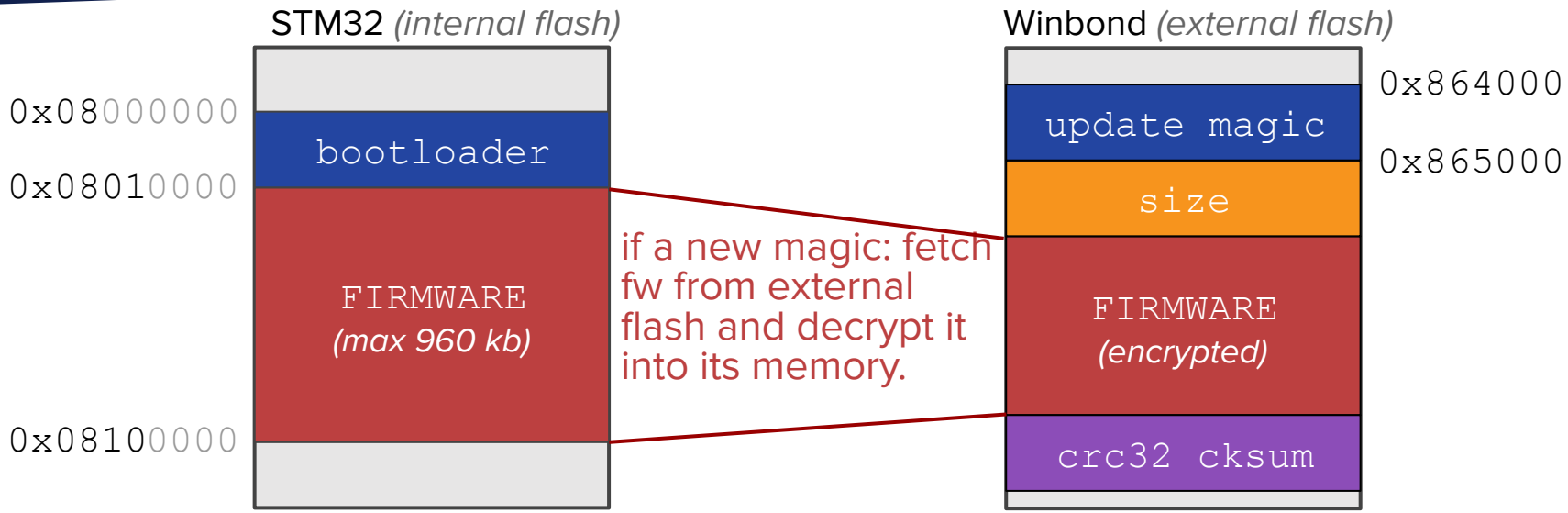
(besides participating to pwn2own)

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

Persistency

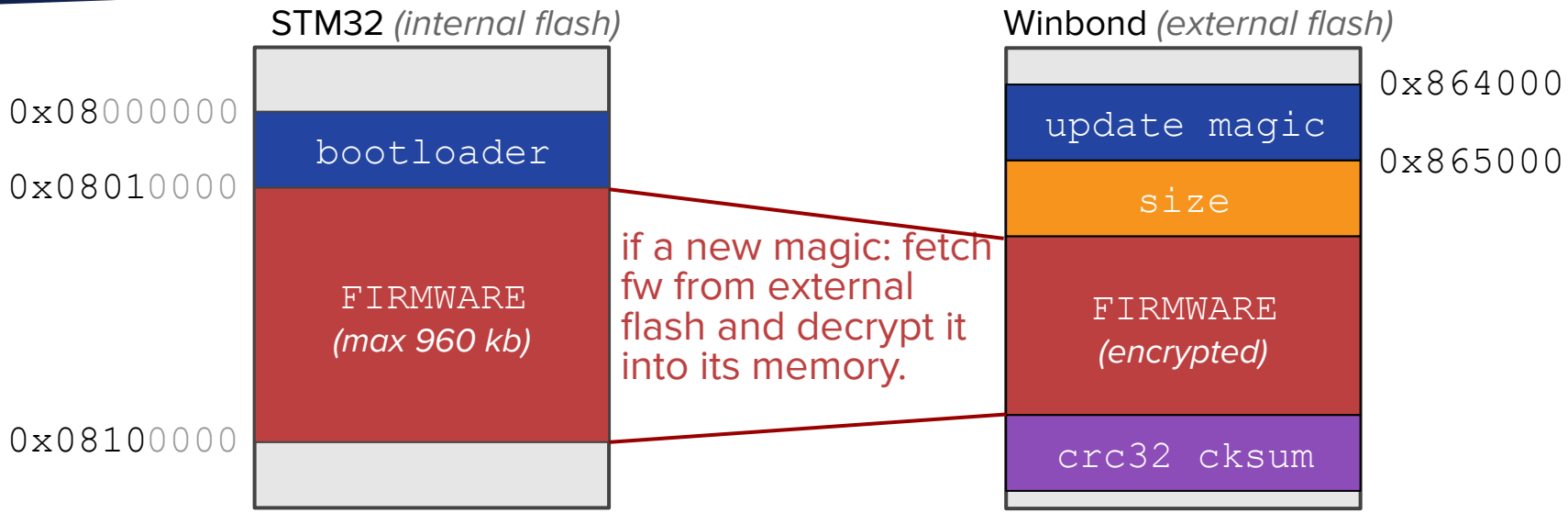


Persistency



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

Persistency



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**

Lateral Movement

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)*

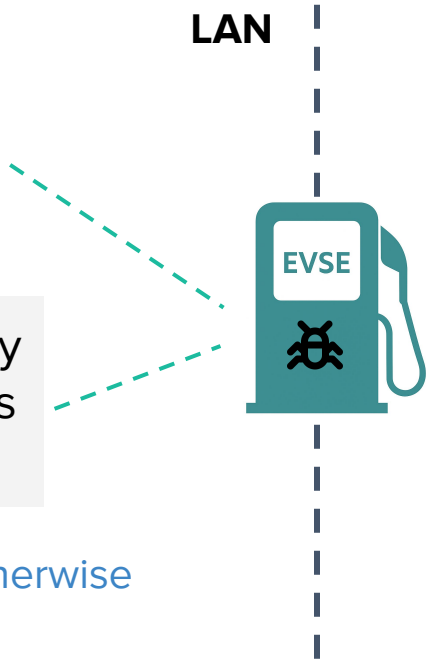
Device used both by **enterprises** and **domestic** users

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, ...)*



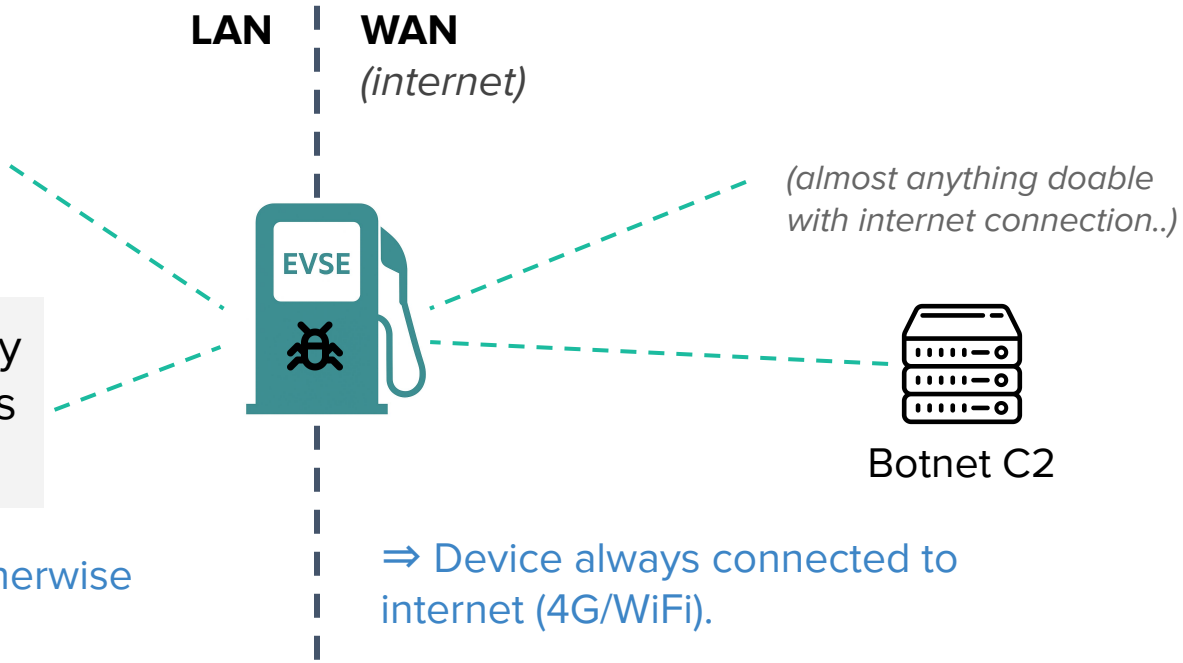
⇒ Expose internal network otherwise unreachable.

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, ...)*

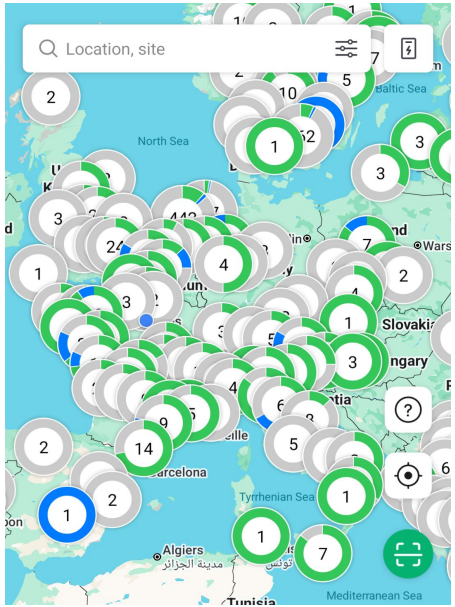


⇒ Expose internal network otherwise unreachable.

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

Finding the devices

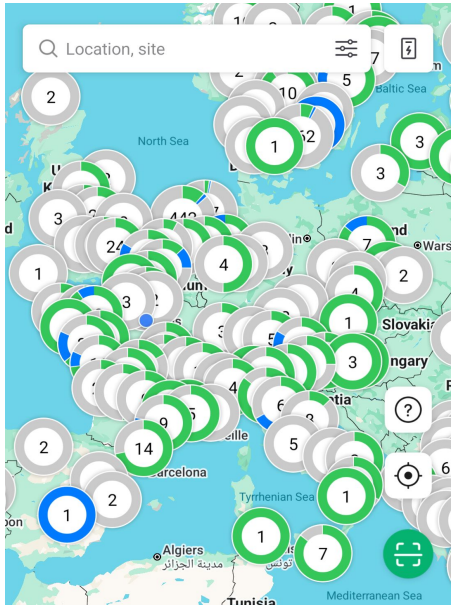
Find the Devices — There's a **Map** for That!

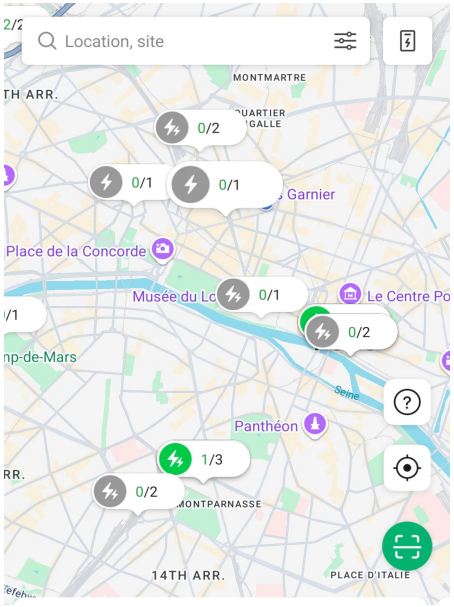


Only public EVSE are shown. Private ones don't appear.

Finding the devices

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

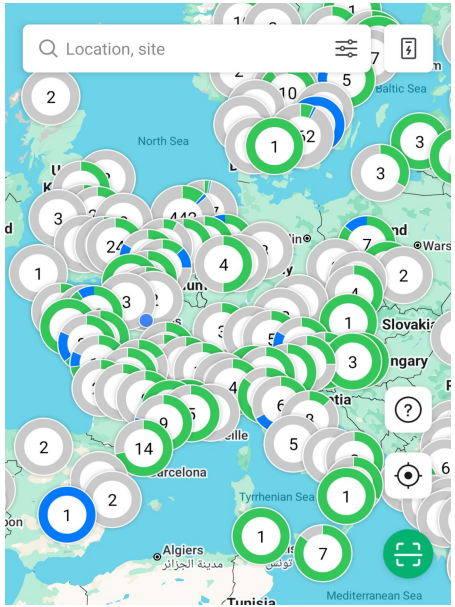


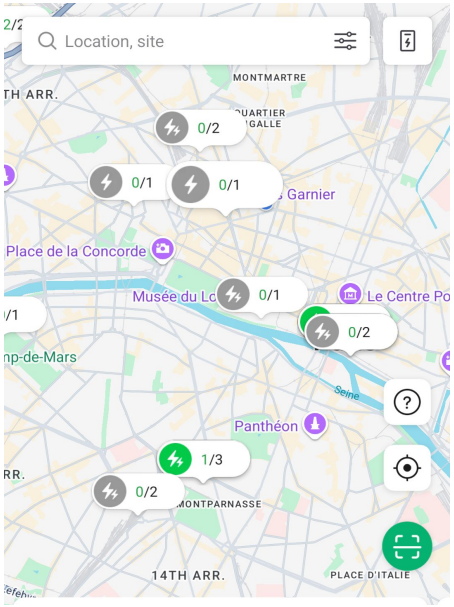


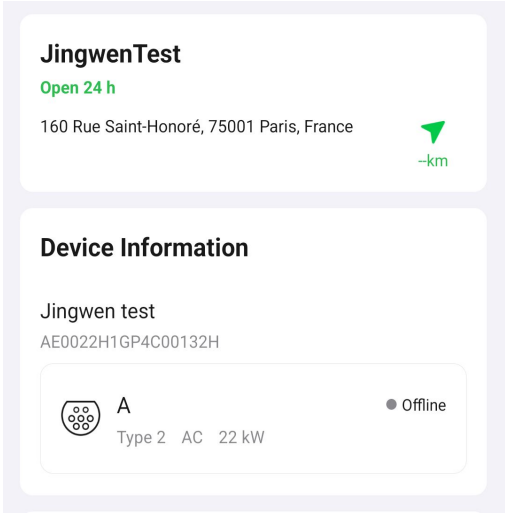
Only public EVSE are shown. Private ones don't appear.

Finding the devices

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



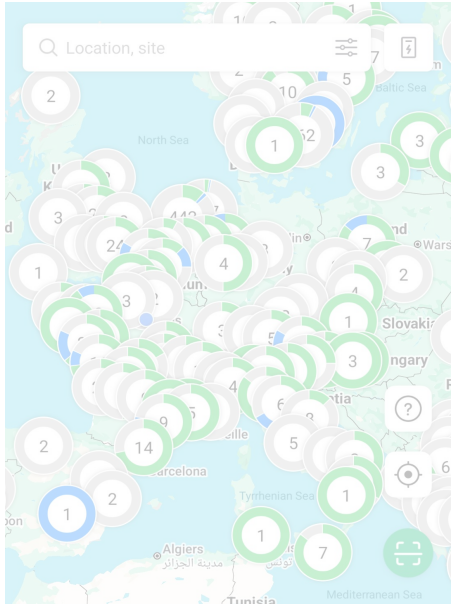


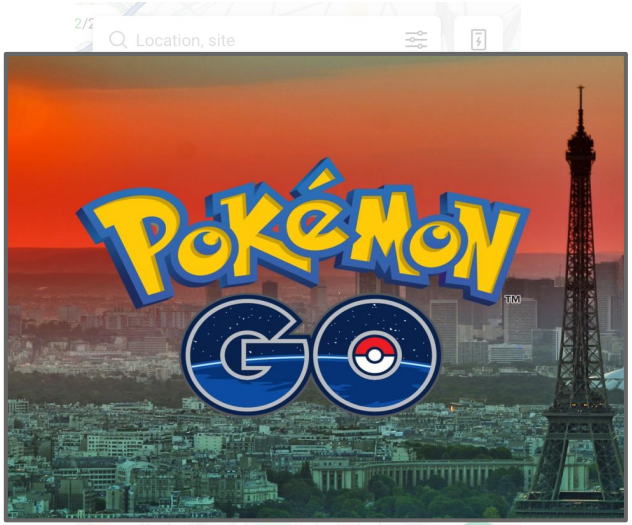


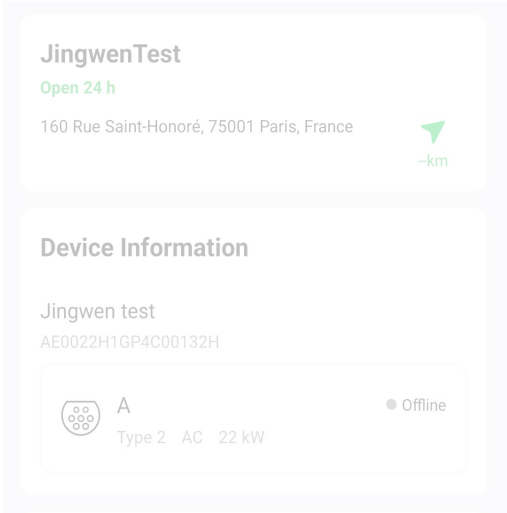
Only public EVSE are shown. Private ones don't appear.

Finding the devices

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



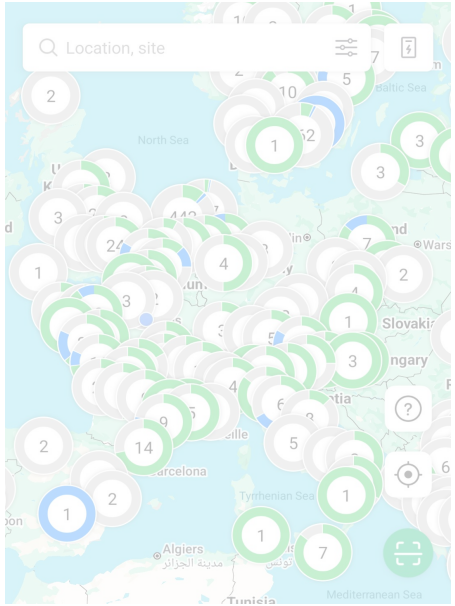


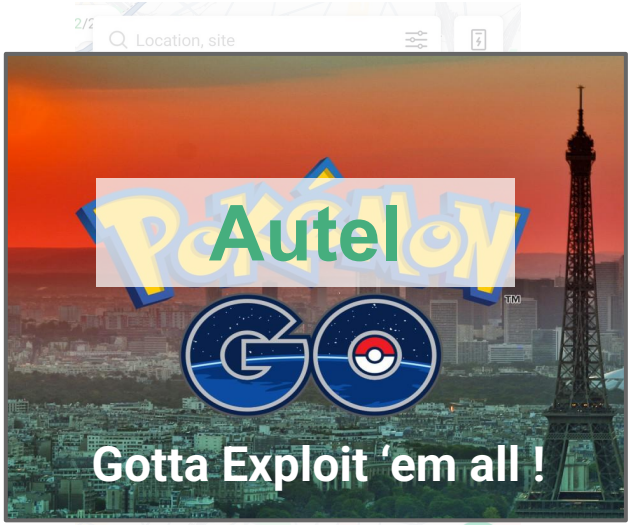


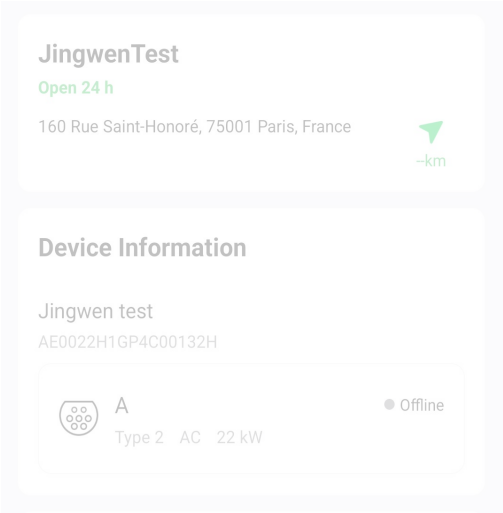
Only public EVSE are shown. Private ones don't appear.

Finding the devices

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







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

Pivot #2: Backend Infrastructure (OCPP, etc)

⇒ **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)*

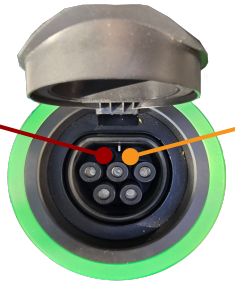
Pivot #3: Car *(side)*

Complete control of **Power Board**

Abusing EVSE for
Free Electricity

Interacting with the car

Proximity Pilot
Determine if connected and supported current



Control Pilot
PWM signal for EV state (ready, charging ...)

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

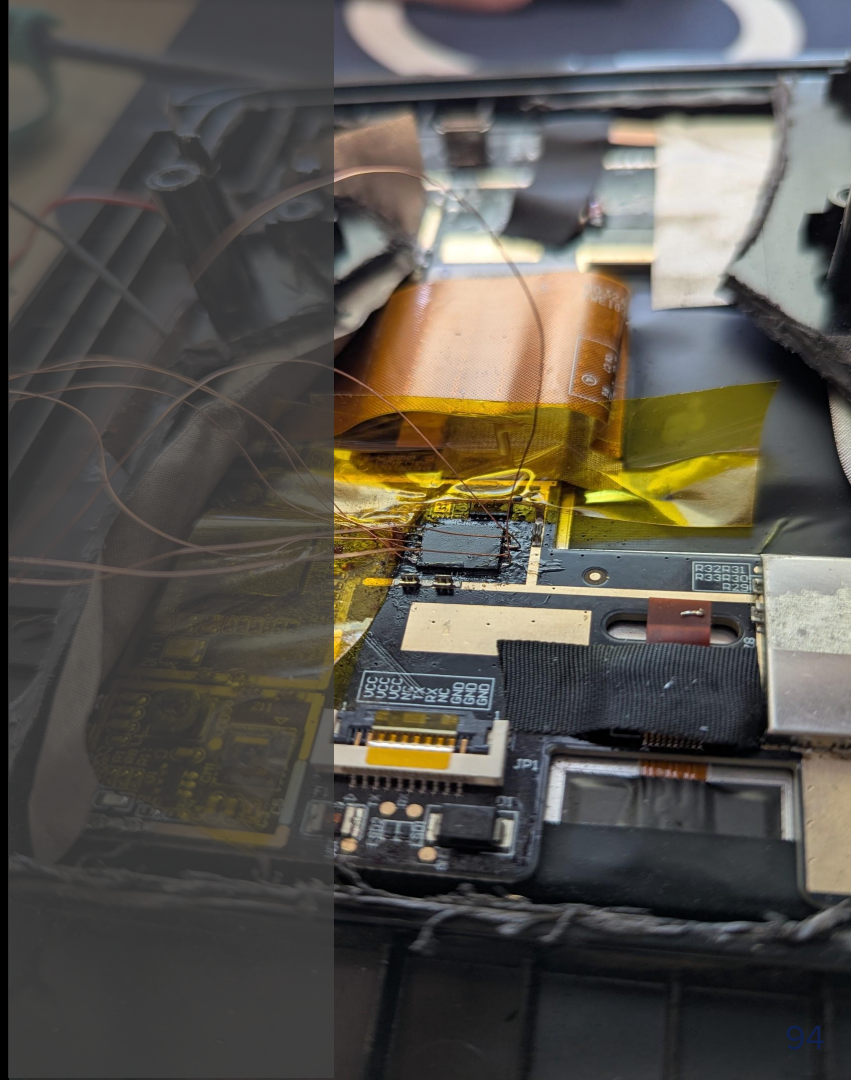
Conclusion & Takeaways

⇒ 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)



THANK YOU !

Quarkslab