SYNACKTIV
DIGITAL SECURITY

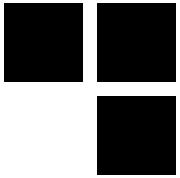
Le tabac, c'est tabou, on en viendra tous à bout !

Bière Sécu Toulouse

November 21 2019

Samuel (@w4kfu) CHEVET

Agenda



1. Introduction
2. The wrong way
3. The right way
4. Conclusion

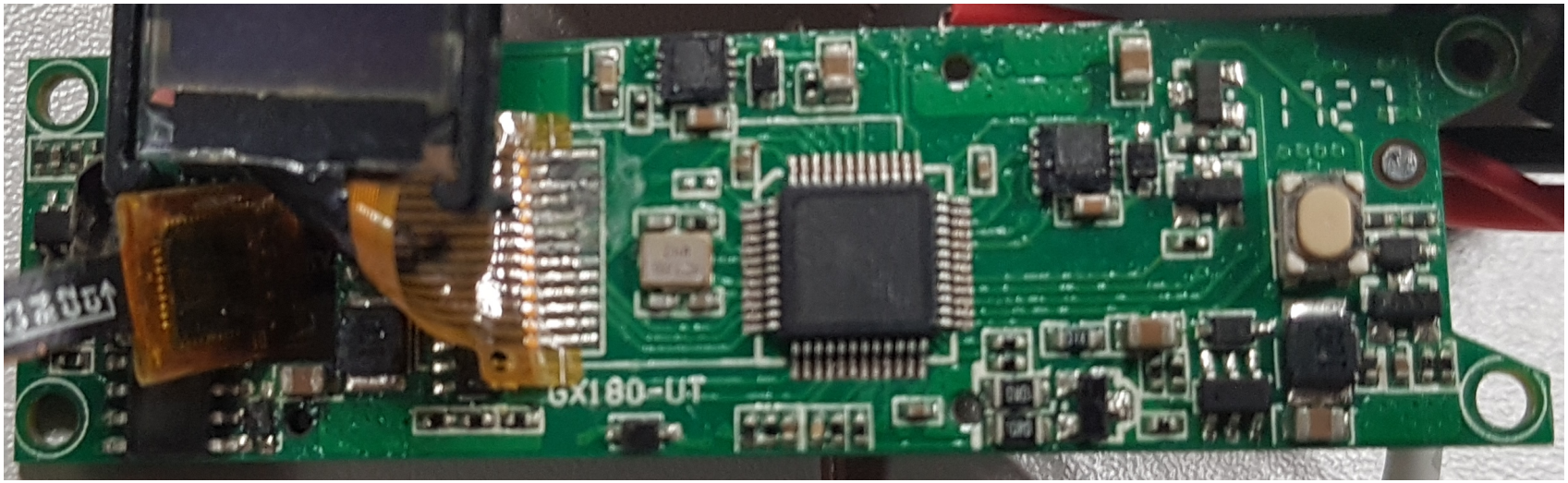
Context

- A co-worker asked me to look at the firmware of its e-cigarette
- Sounds like a fun late night home project

- Target : [redacted] Minikin

Fun fact

- All chips seems brushed : no more free information



Firmware

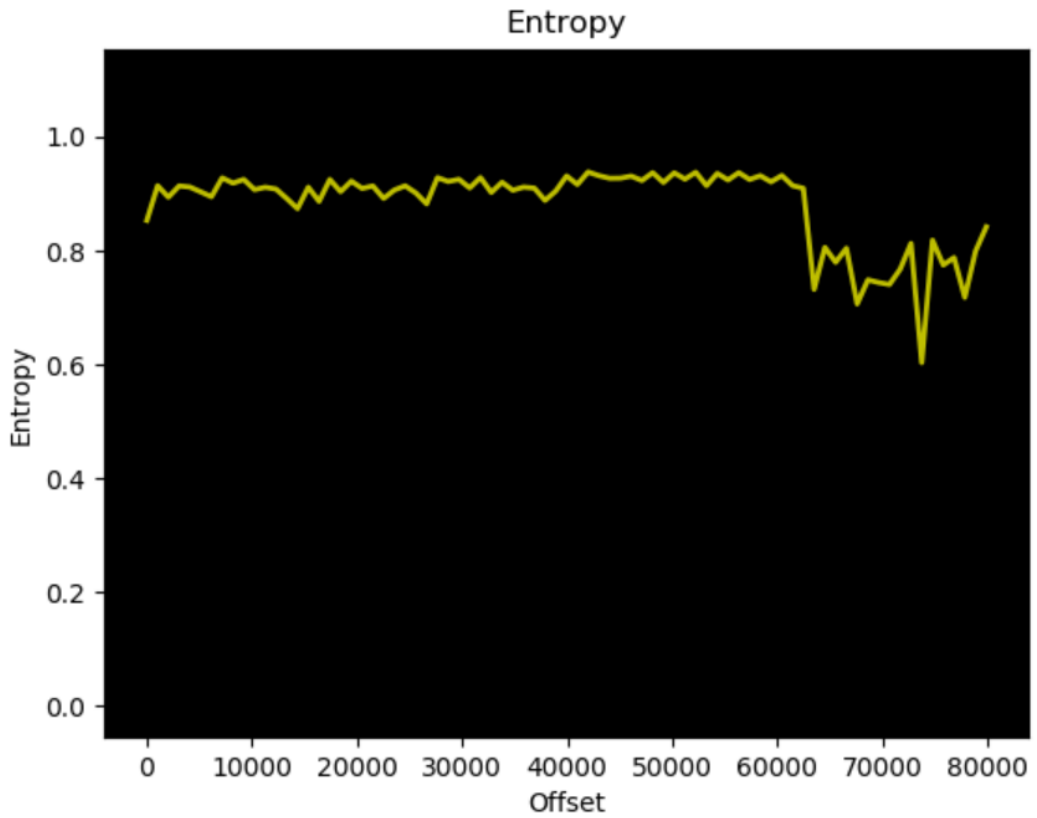
- Firmware files available on vendor website
- Less than 80 KB
- Upgrade tool : Windows executable

```
$ binwalk -Y [redacted].firmware

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
66103         0x10237         ARM executable code, 16-bit (Thumb), big endian, at least 643 valid instructions
```

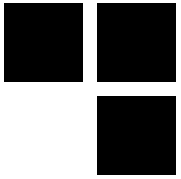
Firmware

- Trying to disassemble the firmware with the ARM CPU detected leads to garbage …



binwalk -E [redacted].firmware

Agenda



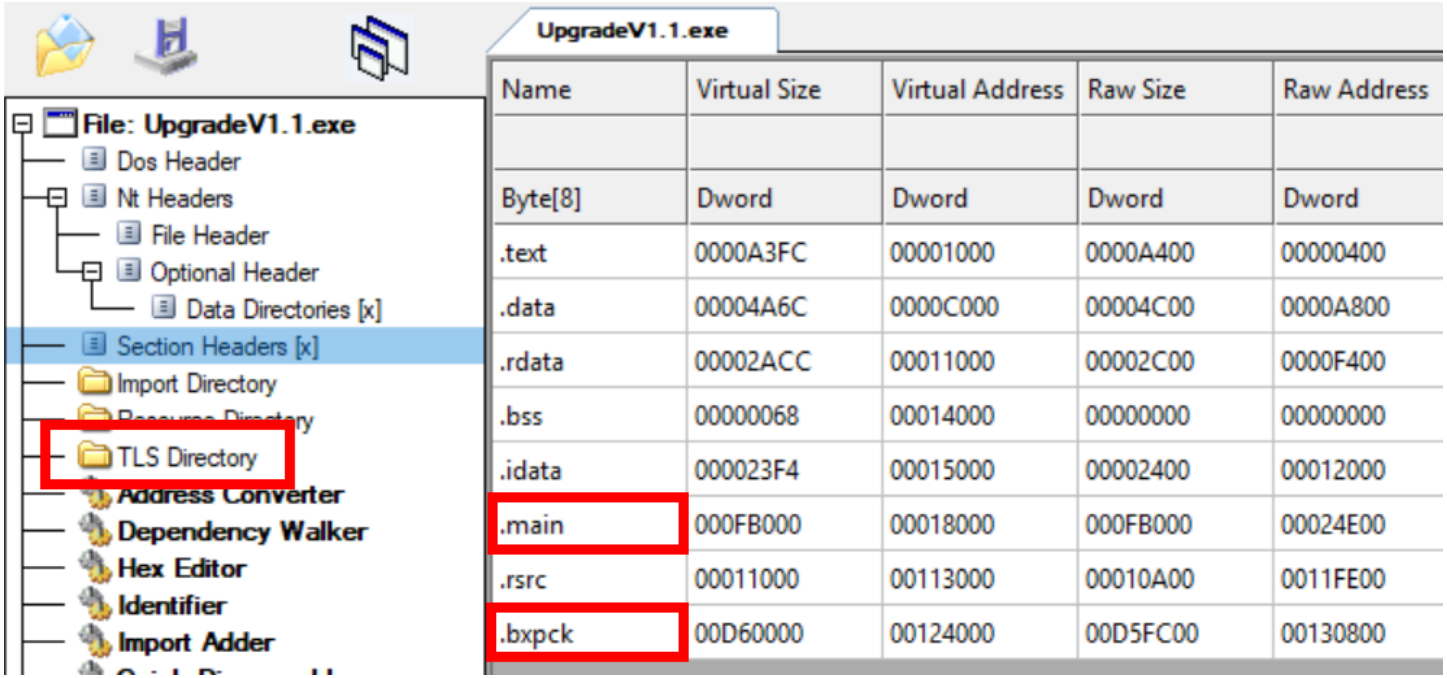
1 Introduction

2 The wrong way

3 The right way

4 Conclusion

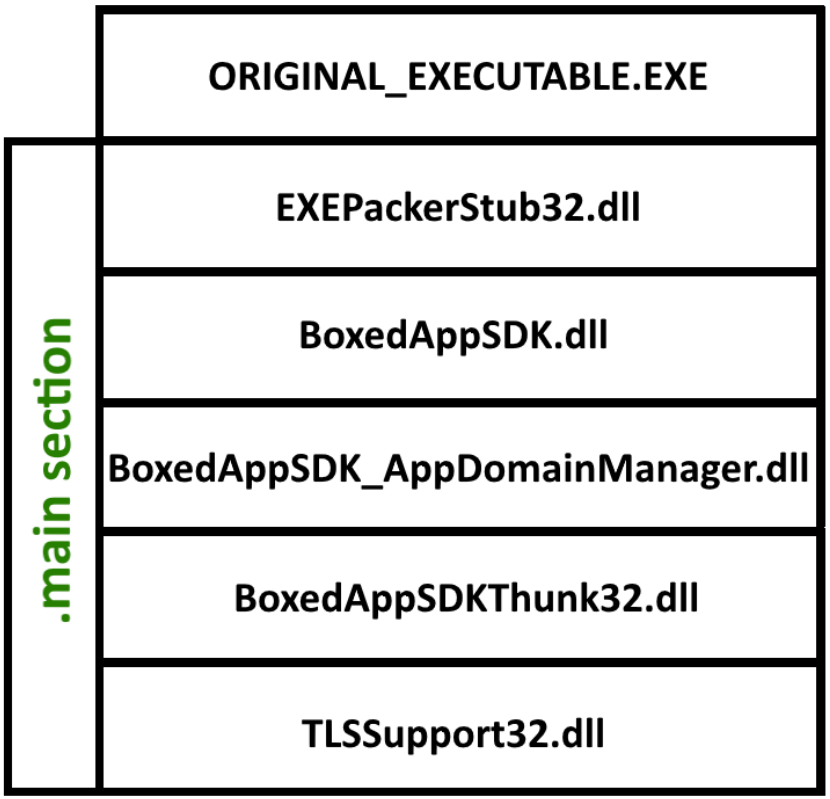
Upgrade tool



Name	Virtual Size	Virtual Address	Raw Size	Raw Address
Byte[8]	Dword	Dword	Dword	Dword
.text	0000A3FC	00001000	0000A400	00000400
.data	00004A6C	0000C000	00004C00	0000A800
.rdata	00002ACC	00011000	00002C00	0000F400
.bss	00000068	00014000	00000000	00000000
.idata	000023F4	00015000	00002400	00012000
.main	000FB000	00018000	000FB000	00024E00
.rsrc	00011000	00113000	00010A00	0011FE00
.bxpck	00D60000	00124000	00D5FC00	00130800

- BoxedApp packer
- Full-fledged applications converted into a single executable
- ’Emulate’ file system and a system registry

.main section



- bxpck section handler
- Virtual Environment / FileSystem / Registry
- .NET manager

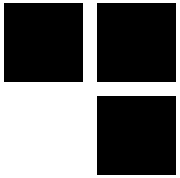
.bxpck section

- Series of different entry type

  + 0x00 : LENGTH [DWORD]
  + 0x04 : TYPE [WORD]

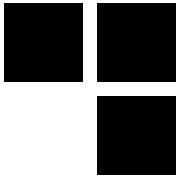
    - 0x02 : Command line overriding
    - 0x03 : Import section information
    - 0x05 : Virtual file entry
    - 0x06 : Start ROOT
    - 0x07 : Virtual registry sub-entry
    - 0x08 : Virtual registry entry
    - 0x0A : End ROOT

Upgrade tool



- Virtual file container contains 6 uninteresting DLLs :
  - hidapi.dll
  - libgcc_s_dw2-1.dll
  - mingwm10.dll
  - msvcr100d.dll
  - qtcore4.dll
  - qtgui4.dll
- Application allows to put the device in DFU mode and send the firmware image without any treatment ☹

Agenda



1 Introduction

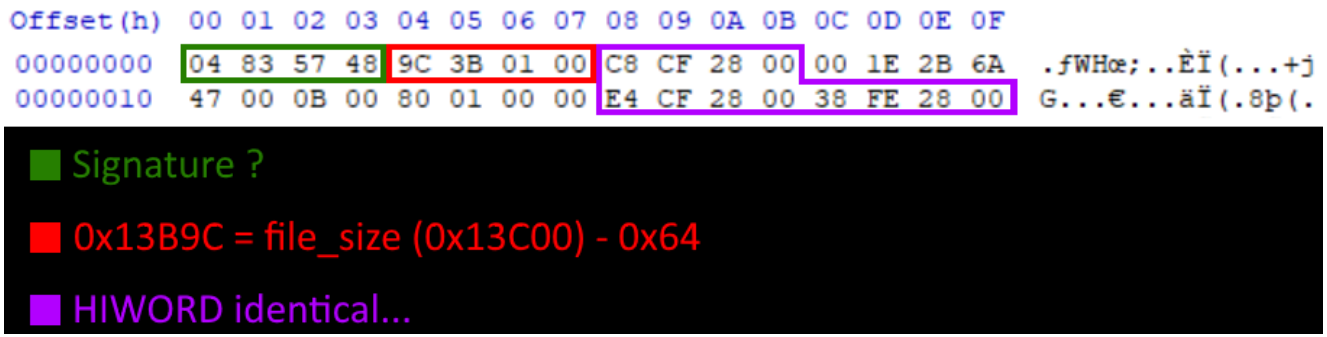
2 The wrong way

3 The right way

4 Conclusion

Guess 100

- Looking at the file under an hex editor
- Values seems stored in Little-Endian



Guess 100

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	04	83	57	48	9C	3B	01	00	C8	CF	28	00	00	1E	2B	6A	.ƒWHœ;..ÈÏ(...+j
00000010	47	00	0B	00	80	01	00	00	E4	CF	28	00	38	FE	28	00	G...€...äÏ(.8þ(.
00000020	CF	36	00	20	3E	1B	00	08	F2	D5	00	08	A4	D5	00	08	Ï6. >...òÕ..¤Õ..
00000030	A2	D5	00	08	A0	D5	00	08	BE	D5	00	08	EF	26	00	00	¢Õ.. Õ..¾Õ..ï&..
00000040	5C	56	00	00	5C	56	00	00	5C	56	00	00	0F	A5	00	08	\V..\V..\V...¥..
00000050	09	A5	00	08	5C	56	00	00	0B	A5	00	08	05	A5	00	08	.¥..\V...¥...¥..
00000060	E2	2C	00	08	E2	2C	00	08	E2	2C	00	08	E2	2C	00	08	â,..â,..â,..â,..
00000070	E2	2C	00	08	52	E2	00	08	E2	2C	00	08	E2	2C	00	08	â,..Râ..â,..â,..
00000080	F3	6B	00	08	FB	B6	00	08	F3	6B	00	08	BD	B2	00	08	ók..û¶..ók..½²..
00000090	F3	6B	00	08	F3	6B	00	08	F3	6B	00	08	F3	6B	00	08	ók..ók..ók..ók..
000000A0	3D	4A	00	08	3D	4A	00	08	3D	4A	00	08	3D	4A	00	08	=J..=J..=J..=J..
000000B0	3D	4A	00	08	3D	4A	00	08	3D	4A	00	08	43	96	00	08	=J..=J..=J..C–..
000000C0	AE	32	00	08	AE	32	00	08	AE	32	00	08	AE	32	00	08	®2..®2..®2..®2..
000000D0	2A	E8	00	08	AE	32	00	08	AE	32	00	08	AE	32	00	08	*è..®2..®2..®2..
000000E0	45	19	00	08	45	19	00	08	45	19	00	08	45	19	00	08	E...E...E...E...
000000F0	45	19	00	08	45	19	00	08	45	19	00	08	45	19	00	08	E...E...E...E...
00000100	63	F8	00	08	A9	21	00	08	A9	21	00	08	A9	21	00	08	cø..©!..©!..©!..

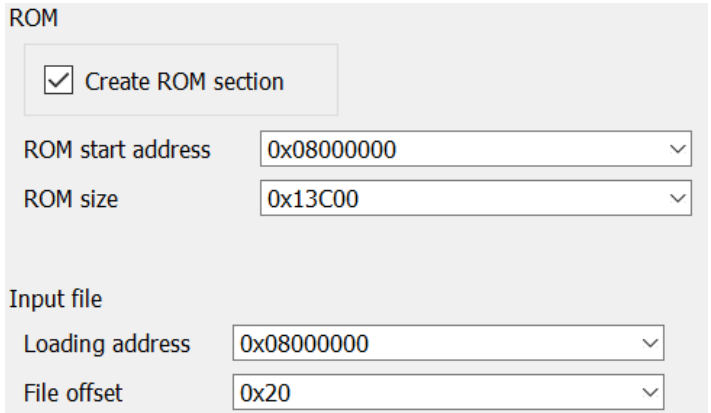
Unique

HIWORD NULL

Pattern of 4 bytes

- We are probably dealing with some ARM Cortex
- Based on the vector table entries (pattern of 4 bytes), is the ROM start address : $0x08000000$?

Vector table




```
ROM:08000000                    CODE16
ROM:08000000                    DCD 0x200036CF          ; Initial SP Value
ROM:08000004                    DCD 0x8001B3E           ; Reset Vector
ROM:08000008                    DCD 0x800D5F2           ; NMI Vector
ROM:0800000C                    DCD 0x800D5A4           ; Hard Fault
ROM:08000010                    DCD 0x800D5A2           ; Memory Management Fault
ROM:08000014                    DCD 0x800D5A0           ; Bus Fault
ROM:08000018                    DCD 0x800D5BE           ; Usage Fault
ROM:0800001C                    DCD 0x26EF              ; RESERVED
ROM:08000020                    DCD 0x565C              ; RESERVED
ROM:08000024                    DCD 0x565C              ; RESERVED
ROM:08000028                    DCD 0x565C              ; RESERVED
```

Vector table

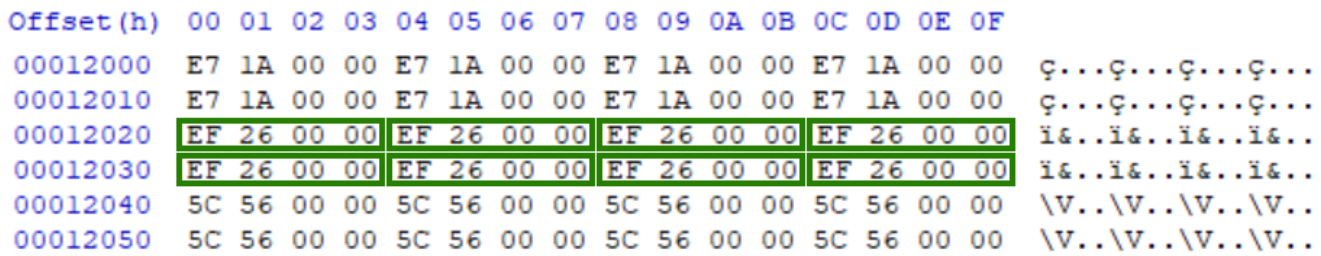
```
ROM:08000000                CODE16
ROM:08000000                DCD 0x200036CF          ; Initial SP Value
ROM:08000004                DCD 0x8001B3E           ; Reset Vector
ROM:08000008                DCD 0x800D5F2           ; NMI Vector
ROM:0800000C                DCD 0x800D5A4           ; Hard Fault
ROM:08000010                DCD 0x800D5A2           ; Memory Management Fault
ROM:08000014                DCD 0x800D5A0           ; Bus Fault
ROM:08000018                DCD 0x800D5BE           ; Usage Fault
ROM:0800001C                DCD 0x26EF              ; RESERVED
ROM:08000020                DCD 0x565C              ; RESERVED
ROM:08000024                DCD 0x565C              ; RESERVED
ROM:08000028                DCD 0x565C              ; RESERVED
```

Problems

- Initial SP Value not aligned ?
- Reserved vector not NULL ?

Grep

- Grepping the sequence of bytes $\mathrm{EF\ 26\ 00\ 00}$

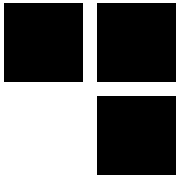


```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00012000   E7 1A 00 00 E7 1A 00 00 E7 1A 00 00 E7 1A 00 00   ç...ç...ç...ç...
00012010   E7 1A 00 00 E7 1A 00 00 E7 1A 00 00 E7 1A 00 00   ç...ç...ç...ç...
00012020   EF 26 00 00 EF 26 00 00 EF 26 00 00 EF 26 00 00   ï&..ï&..ï&..ï&..
00012030   EF 26 00 00 EF 26 00 00 EF 26 00 00 EF 26 00 00   ï&..ï&..ï&..ï&..
00012040   5C 56 00 00 5C 56 00 00 5C 56 00 00 5C 56 00 00   \V..\V..\V..\V..
00012050   5C 56 00 00 5C 56 00 00 5C 56 00 00 5C 56 00 00   \V..\V..\V..\V..
```

- $\mathrm{0x000026EF}$ repeated 8 times
- The other reserved vector ($\mathrm{0x0000565C}$) repeated 8 times
- …

Cryptography ?

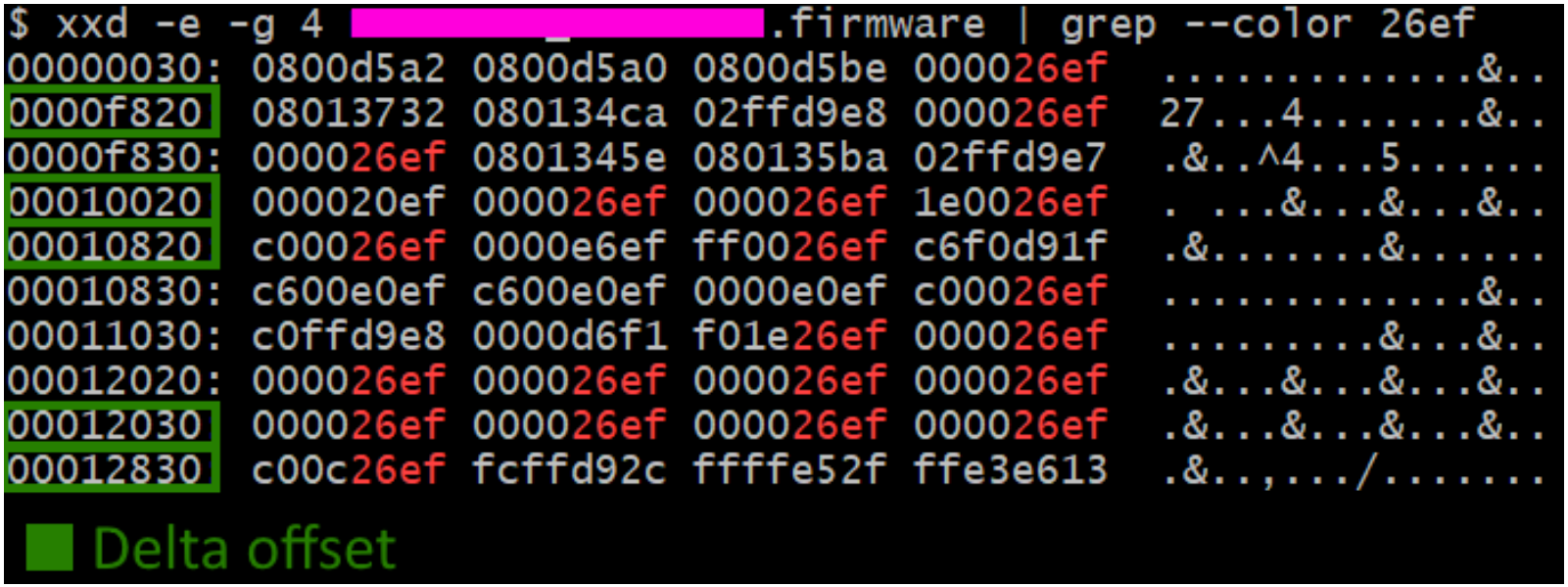


- Sequence of bytes seems to have lead us to ZERO pages
- We are probably dealing with a XOR cipher
- Each key entry seems only on 16 bits

Key format ?

$$
\begin{array}{ll}
+\ 0\text{x}00 : \text{key\_}00 & [\text{DWORD}] * 0\text{x}08 \\
+\ 0\text{x}20 : \text{key\_}01 & [\text{DWORD}] * 0\text{x}08 \\
+\ 0\text{x}40 : \text{key\_}02 & [\text{DWORD}] * 0\text{x}08 \\
\ldots &
\end{array}
$$

Identifying key length



- Key length : 0x800 bytes

Full key extraction

- Firmware doesn't seem to contain ZERO page of 0x800 bytes
- 3 different versions of the firmware available

Solution

1. Split firmware in block of 0x800 bytes
2. Compute the number of occurence of every 0x04 bytes repeated every 0x20 bytes
3. Repeat the task for all firmwares available

Wait …strings

Firmware_1.out

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000070A0   FE F7 6B B8 B4 02 00 20 B0 23 01 08 9F 5A 01 08   þ÷k¸´.. °#..ŸZ..
000070B0   5D 43 01 08 53 48 4F 52 54 00 00 00 41 54 4F 4D   ]C..SHORT...ATOM
000070C0   49 5A 45 52 00 00 00 00 48 49 47 48 00 00 00 00   IZER....HIGH....
000070D0   54 45 4D 50 45 52 00 00 57 4F 52 4B 00 00 00 00   TEMPER..WORK....
000070E0   4F 56 45 52 54 49 4D 45 00 00 00 00 4C 4F 57 00   OVERTIME....LOW.
```

Firmware_2.out

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000074B0   64 07 00 20 03 6F 01 08 B9 52 01 08 0B 6D 01 08   d.. .o..¹R...m..
000074C0   A3 48 4F 52 A4 00 00 00 B1 54 4F 4D B9 5A 45 52   £HOR¤...±TOM¹ZER
000074D0   F0 00 00 00 B8 49 47 48 F0 00 00 00 A4 45 4D 50   ð...¸IGHð...¤EMP
000074E0   45 52 00 00 57 4F 52 4B 00 00 00 00 4F 56 45 52   ER..WORK....OVER
```

$(0x74C0 \ \% \ 0x800) \ / \ 0x20 = 0x26$; Key index broken

Wait ...bindiff

Firmware_1.out

```
ROM:0800B92C 40 1C                    ADDS            R0, R0, #1
ROM:0800B92E C5 F8 88 00              STR.W           R0, [R5,#0x88]
ROM:0800B932 58 45                    CMP             R0, R11
ROM:0800B934 01 D9                    BLS             loc_800B93A
ROM:0800B936 C5 F8 88 60              STR.W           R6, [R5,#0x88]
ROM:0800B93A
ROM:0800B93A                  loc_800B93A                             ; CODE X
ROM:0800B93A FD F7 9F FF              BL              sub_800987C
ROM:0800B93E
ROM:0800B93E                  loc_800B93E                             ; CODE X
ROM:0800B93E FF F7 FA FB              BL              sub_800B136
ROM:0800B942
ROM:0800B942                  loc_800B942                             ; CODE X
ROM:0800B942 85 F8 18 62              STRB.W          R6, [R5,#0x218]
ROM:0800B946 08 20                    MOVS            R0, #8
ROM:0800B948 FC F7 B1 FF              BL              sub_80088AE
ROM:0800B94C 66 72                    STRB            R6, [R4,#9]
ROM:0800B94E 66 73                    STRB            R6, [R4,#0xD]
```

Firmware_2.out

```
ROM:0800C40C 40 1C                    ADDS            R0, R0, #1
ROM:0800C40E C5 F8 88 00              STR.W           R0, [R5,#0x88]
ROM:0800C412 58 45                    CMP             R0, R11
ROM:0800C414 01 D9                    BLS             loc_800C41A
ROM:0800C416 C5 F8 88 60              STR.W           R6, [R5,#0x88]
ROM:0800C41A
ROM:0800C41A                  loc_800C41A                             ; CODE X
ROM:0800C41A FD F7 F9 FE              BL              sub_800A210
ROM:0800C41A                  ; ---------------------------------------------------
ROM:0800C41E FF               unk_800C41E     DCB 0xFF                ; CODE X
ROM:0800C41F F7                               DCB 0xF7
ROM:0800C420 76                               DCB 0x76 ; v
ROM:0800C421 03                               DCB    3
ROM:0800C422                  ; ---------------------------------------------------
ROM:0800C422
ROM:0800C422                  loc_800C422                             ; CODE X
ROM:0800C422 85 F8 BD 9A              STRB.W          R9, [R5,#0xABD]
ROM:0800C426 08 20                    MOVS            R0, #8
ROM:0800C428 71 0F                    LSRS            R1, R6, #0x1D
```

$(0xC440 \ \% \ 0x800) \ / \ 0x20 = 0x22$; Key index broken

Full key extraction

- After some adjustments we have the full key
  - Index 30, 34, 38, 60, 62 were broken

Full key extraction

- After some adjustments we have the full key
  - Index 30, 34, 38, 60, 62 were broken
- Some key are on 32 bits ...wat ?

$$\text{keys}[54] = \text{0xD6ADBCEF}$$
$$\text{keys}[58] = \text{0xDEADBEEF}$$
$$\text{keys}[63] = \text{0xE3ADBEEF}$$

Last step

- Loading the ROM at address $0x08000000$ doesn't seem to work
- Vector table entry points to nonsense disassembly

Solution 1

- Grepping the disassembly for something that looks like a $Reset\_Handler$

```
ROM:080001CE 00 00      MOVS            R0, R0
ROM:080001D0 09 48      LDR             R0, =(sub_800F41A+1)
ROM:080001D2 80 47      BLX             R0          ; sub_800F41A
ROM:080001D4 09 48      LDR             R0, =(loc_8003CF4+1)
ROM:080001D6 00 47      BX              R0          ; loc_8003CF4
```

- Compute the delta between the vector table entry address and the address found

Last step

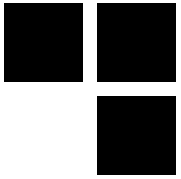
- Loading the ROM at address 0x08000000 doesn't seem to work
- Vector table entry points to nonsense disassembly

Solution 2

- Grepping the immediate value of Vector Table Offset Register : 0xE000ED08

```
ROM:0800B854 2E 49  LDR             R1, =0xE000ED08
ROM:0800B856 2D 48  LDR             R0, =0x8003C00
ROM:0800B858 08 60  STR             R0, [R1]
                                    ...
ROM:0800B90C 00 3C+dword_800B90C    DCD 0x8003C00
ROM:0800B910 08 ED+off_800B910      DCD 0xE000ED08
```

Agenda



1 Introduction

2 The wrong way

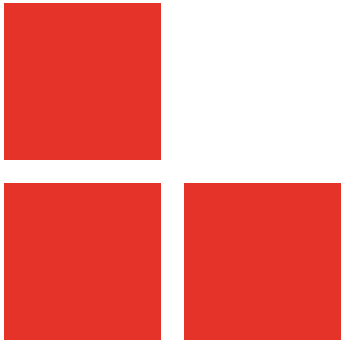
3 The right way

4 Conclusion

Conclusion

- The BoxedApp executable is useless in our case
- The firmware key can be extracted from only observation
- Generation of the XOR cipher key seems broken (32 bits values sometimes)
- My co-worker can now reverse engineer it without problems !

QUESTIONS?



Thank you for your attention

SYNACKTIV
DIGITAL SECURITY