SEC Consult

an Atos business

Windows CE Memory Archaeology

Recovering Files from Windows CE 5.0 on NXP i.MX28 NAND Flash

Gerhard Hechenberger
Principal Security Consultant
June 2026 | BSides Vienna

© SEC Consult - Public

$ whoami

Gerhard Hechenberger

Principal Security Consultant
@ SEC Consult

- Head of IoT and Embedded Systems Security
- Head of Hardware Lab
- Hacker / Penetration Tester

Areas:

- IoT and embedded systems
- OT devices and infrastructure

© SEC Consult - Public

How It Started …

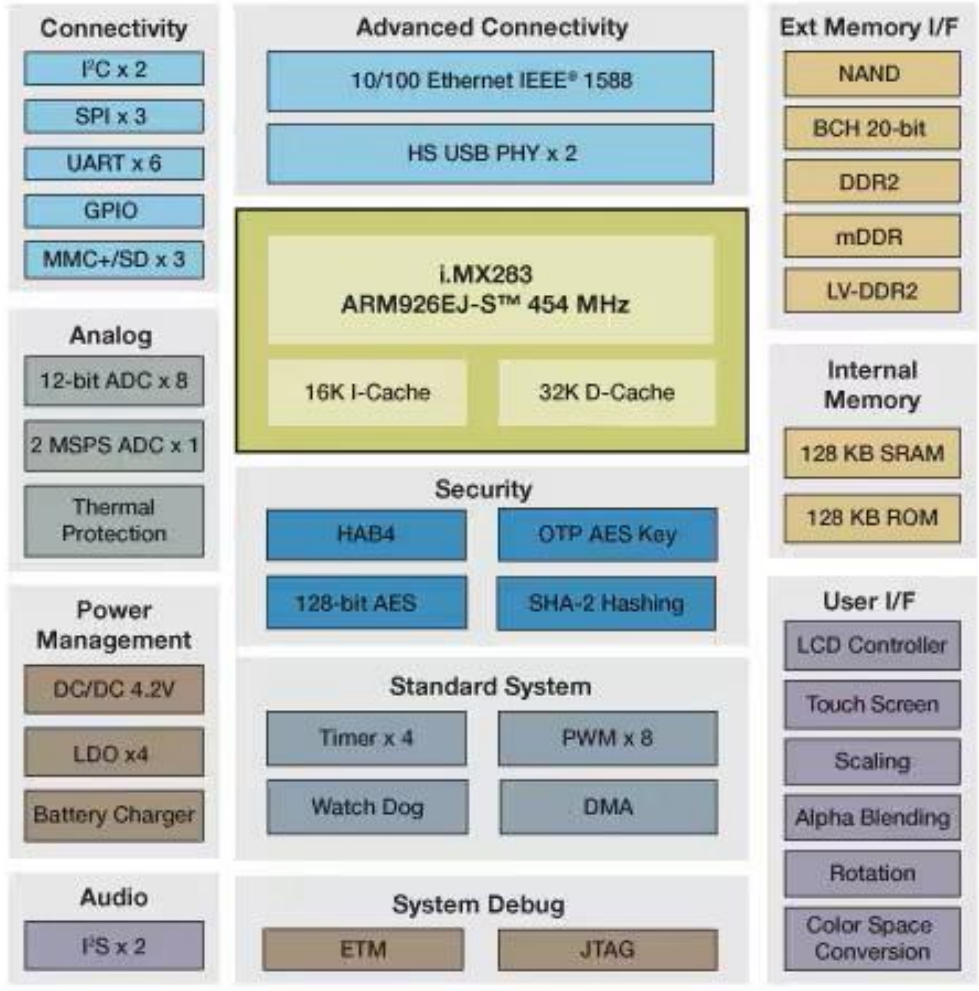
New project had an interesting device running Windows CE 5.0 …

Main processor

- NXP i.MX283
- ARM926EJ-S core

Main memory

- Macronix MX30LF4G18AC
- 3V, 4G-bit, NAND Flash



© SEC Consult - Public

Microsoft Windows CE

- Real-time operating system
- Supports embedded systems (x86, ARM, SuperH, MIPS, PowerPC)
- Widely used in PDAs
- Foundation for Windows Phones
- Today Windows 10 IoT Core

Analyze/play with Windows CE

- Windows XP
- Visual Studio 2005
- Windows CE Platform Builder extension
- BSP provided by vendor

© SEC Consult - Public

Bootloader Access

Assessing Viable Attack Paths

```
[…]
Microsoft Windows CE Bootloader Common Library Version 1.4
Microsoft Windows CE Ethernet Bootloader 1.0 for MX28
[…]
Press [ENTER] to launch image stored in NAND flash or [SPACE]
to cancel.

Initiating image launch in 3 seconds.
------------------------------------------------------------
Freescale iMX SOC Menu Item
------------------------------------------------------------
[…]
 [9] Bootloader Shell
 [I] KITL Work Mode : Interrupt
 [K] KITL Enable Mode : Disable
 [P] KITL Passive Mode : Disable
 [S] Save Settings
 [D] Download Image Now
[…]
 Selection:
```

KITL

```
ENET already in use for KITL. Driver load failure!
```

Bootloader Shell

```
-----------  Freescale iMX Boot Shell  ----------
type ? for help
command -- ?

------------------------------ Help ------------------------------
?                                                 Help
e                                                 Exit Shell
d RegAddress                                      Show Reg
s RegAddress RegValue                             Set Reg
b RegAddress BitOffset(0-31) Value(0 or 1)        Set Bit
------------------------------ End -------------------------------
```

Download Mode

```
Got Response from DHCP server, IP address: 10.10.21.1
No ARP response in 2 seconds, assuming ownership of 10.10.21.1
+EbootSendBootmeAndWaitForTftp
Sent BOOTME to 255.255.255.255
```

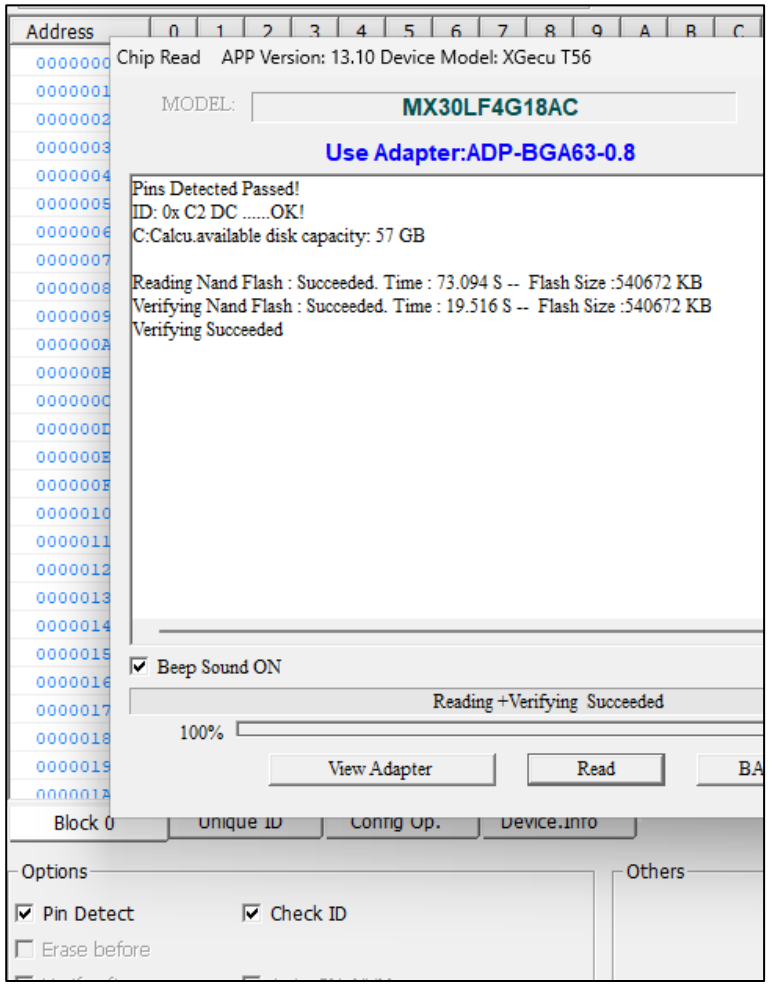
© SEC Consult - Public

Objective: Access the Files

Get all the bytes

- Desolder the chip
- XGecu T56 with BGA63 adapter
- Reading success
- Verifying success

We took the easy route – or so we thought …



© SEC Consult - Public

We Are Finished

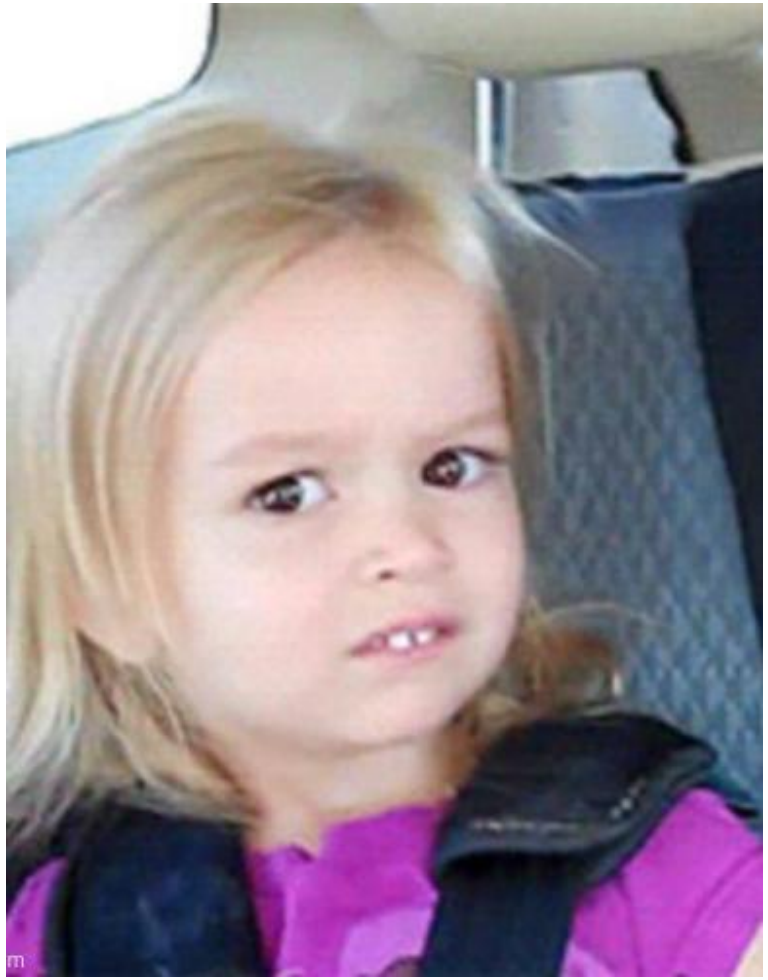
The layman’s approach

1. Extract
2. Profit

Filesize 528M (that‘s odd)

Unblob output

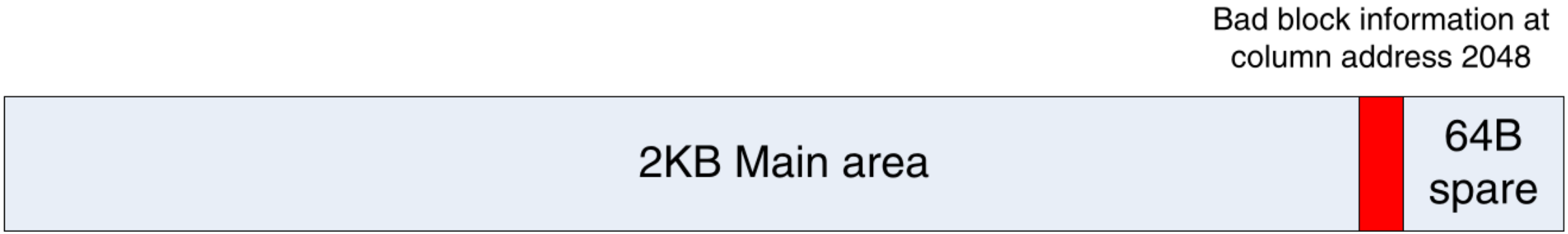
```
│ Extracted files: 1
│ Extracted directories: 0
│ Extracted links: 0
```



© SEC Consult - Public

NAND Memory

NAND is Unreliable



Out Of Band (OOB) data

- Multiple variants
- Here 2048+64
  - 2048 bytes data
  - 64 bytes spare
- Often used for ECC

Bad Block (BB) marker

- Written by factory
- First byte of OOB
- 0xFF -> good block

Described in MCIMX28RM Rev.2

© SEC Consult - Public

Now We Are Finished

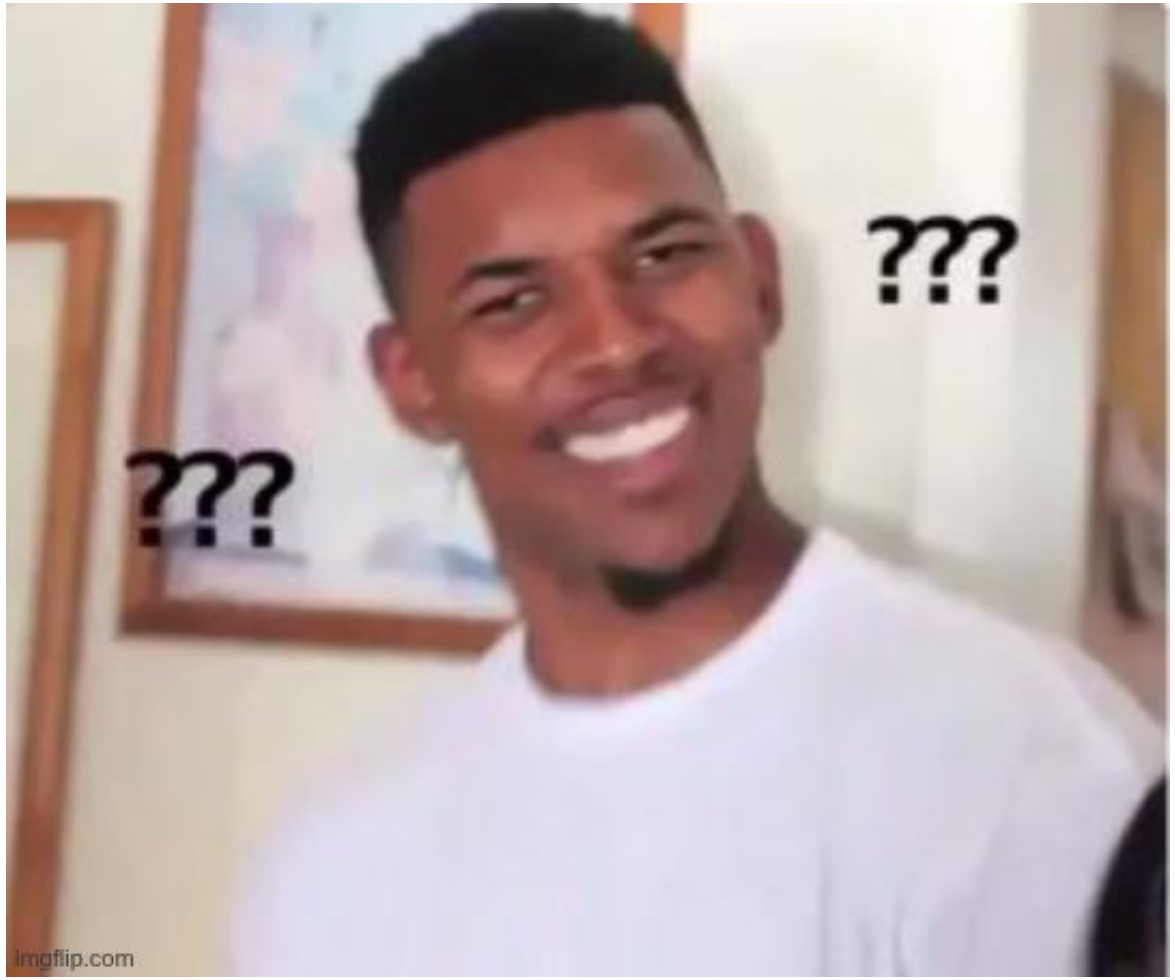
The lazy approach

1. Strip OOB data
   - 64B every 2KB
2. Extract
3. Profit

Filesize 512M

Unblob output

```
│ Extracted files: 1
│ Extracted directories: 0
│ Extracted links: 0
```



© SEC Consult - Public

NAND Memory on NXP i.MX28

Just read the datasheet (IMX28CEC Rev.4) …

“The General-Purpose Media Interface (GPMI) controller is a flexible NAND Flash controller with 8-bit data width, up to 50-MBps I/O speed and individual chip-select and DMA channels for up to 8 NAND devices. It also provides a interface to 20-bit BCH for ECC.”

Everything is configurable … configuration structure FCB (MCIMX28RM Rev.2)

Approach

- Use previous work (and fail)
- Understand the layout
  - AI
  - Previous work
  - Documentation
- Analyse with ImHex
- Build own parsing script

Previous Work

- Digital.Security i.MX NAND Tools
  - https://www.youtube.com/watch?v=1u0OqFhDTdw by Damien Cauquil
  - https://github.com/DigitalSecurity/imx-nand-tools
- SySS NAND Dump Tools
  - https://github.com/SySS-Research/nand-dump-tools

Problems

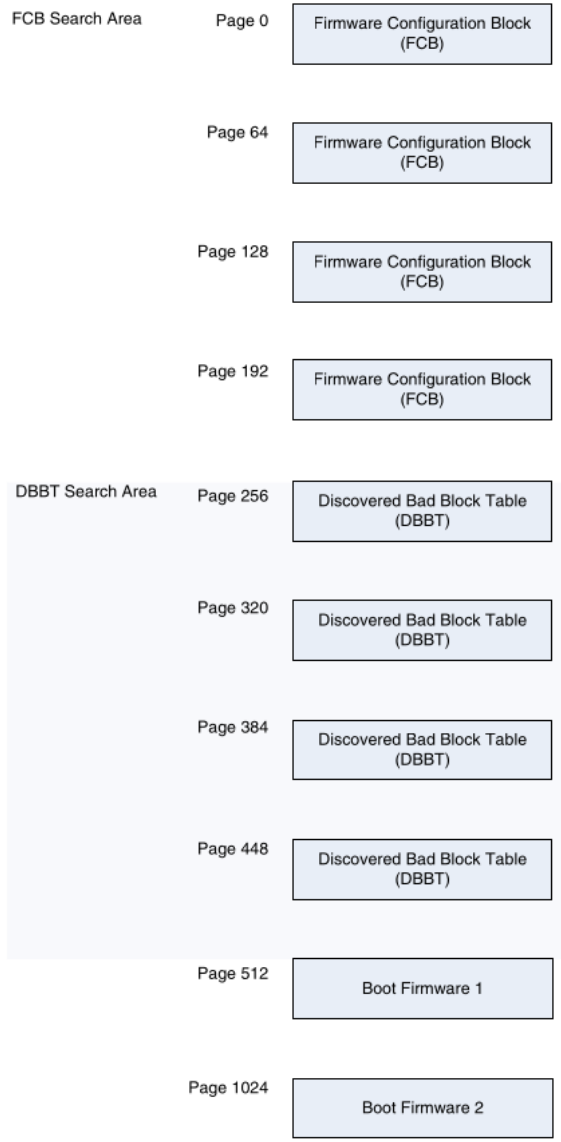
- Outdated dependencies
- Hardcoded parameters
- Targeted to other processor models?

© SEC Consult - Public

NAND Memory on NXP i.MX28

Firmware Configuration Block (FCB)

```
FCB#4 @ 0x00063000
checksum: 1e7afeff
fingerprint: b'FCB '
version: 1
nand_timing: 0f0f0f06403647d0
page_size: 2048
total_page_size: 2112
pages_per_block: 64
number_of_nands: 1
number_of_dies: 1
cell_type: 1
ecc_block_n_type: 2
ecc_block_0_size: 0
ecc_block_n_size: 512
ecc_block_0_type: 4
metadata_bytes: 10
ecc_blocks_per_page: 4
firmware1_start_sector: 640
firmware2_start_sector: 768
firmware1_sectors: 128
firmware2_sectors: 128
dbbt_search_start: 512
bad_block_marker_byte: d5
bad_block_marker_start_bit: 4
bad_block_marker_offset: 2048
```



© SEC Consult - Public

NAND Memory on NXP i.MX28

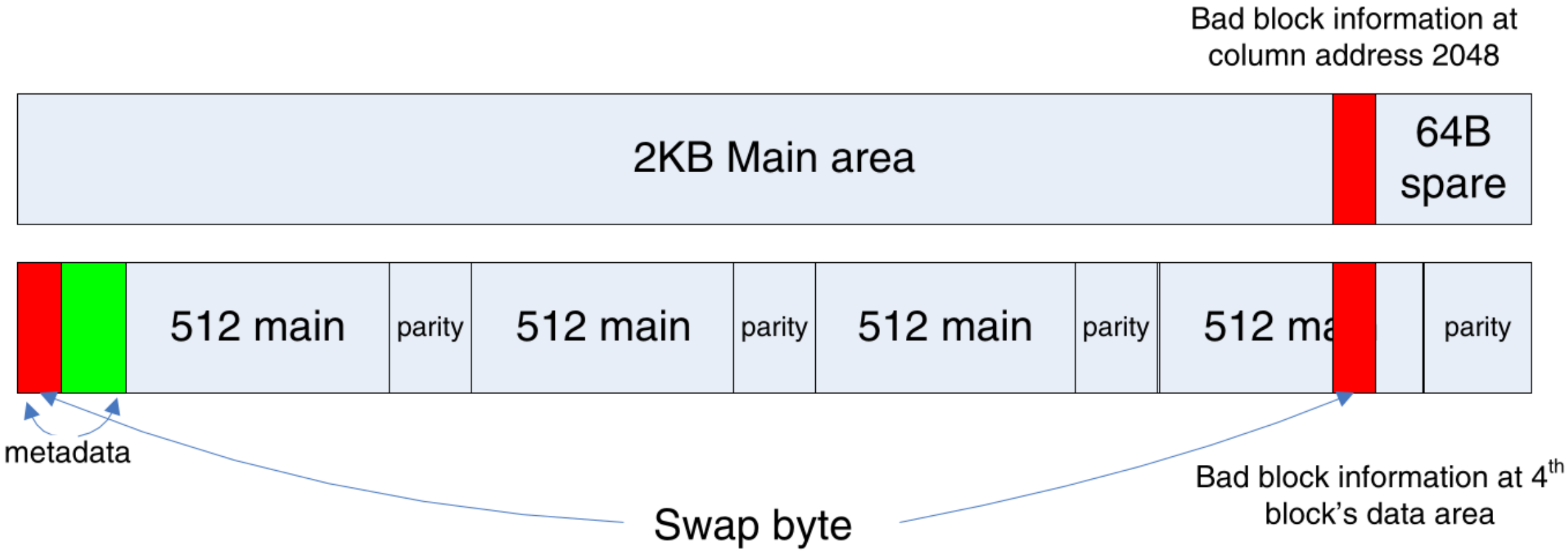


Figure 12-13. Factory Bad Block Marker Preservation

© SEC Consult - Public

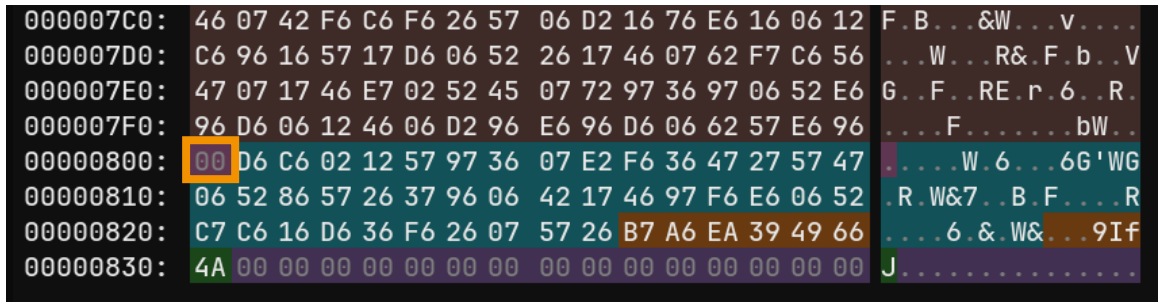
NAND Memory on NXP i.MX28

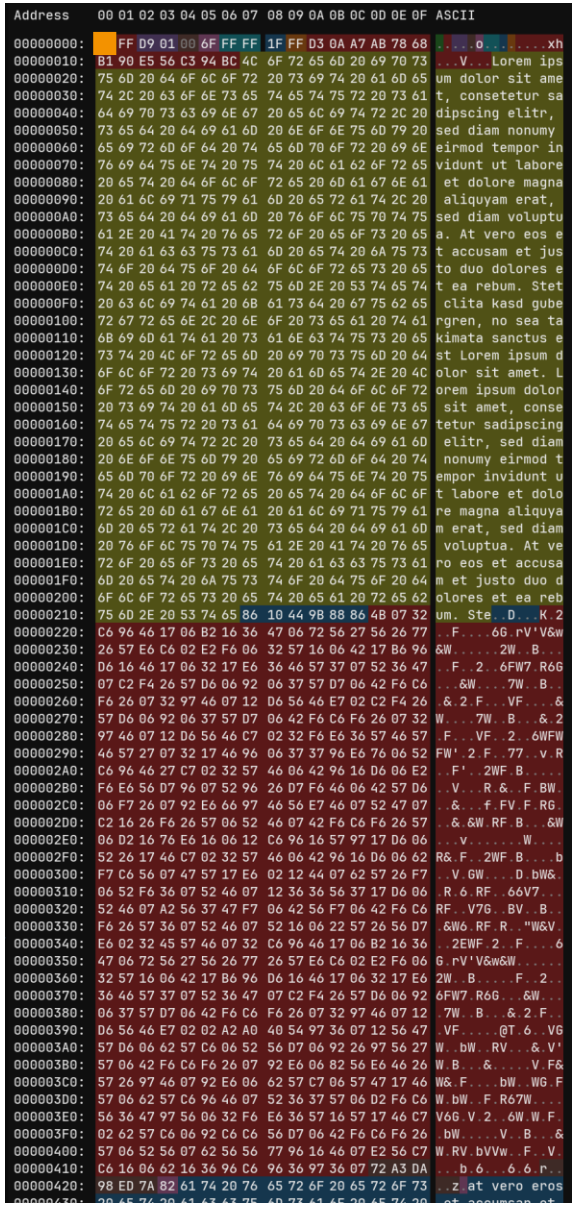
Swap Bad Byte

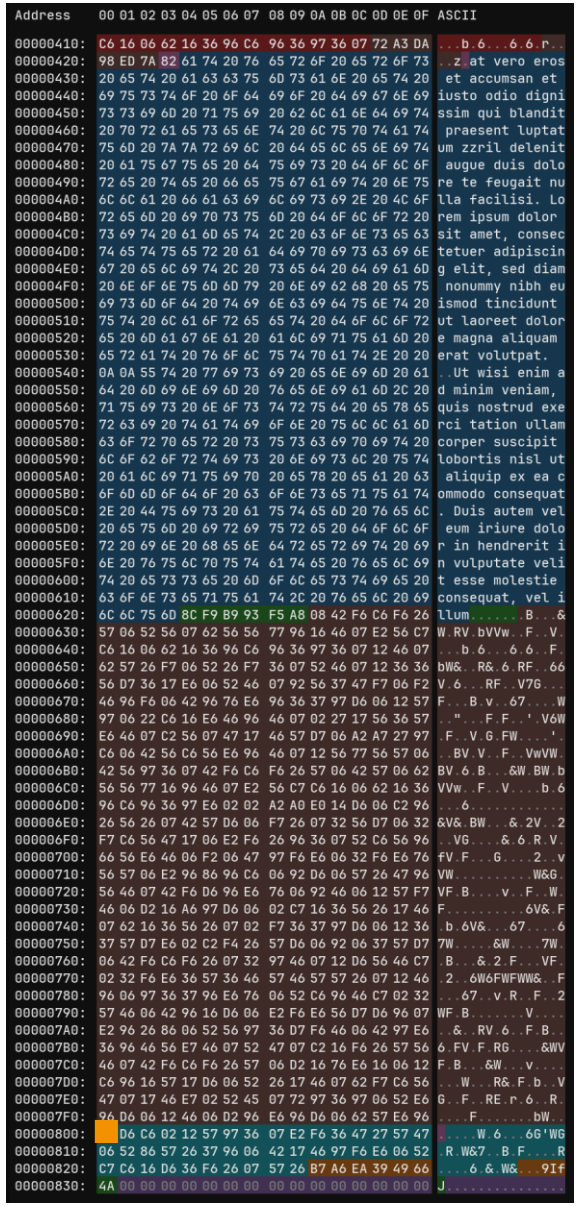
Swap bytes at

- Index 0x0
- Index 0x200 (2048)

ECC is calculated with swapped (wrong) byte!







© SEC Consult - Public

NAND Memory on NXP i.MX28

Calculate ECC

BCH algorithm

- Configurable polynom
  - Here `0x201b`
- Configurable length
  - 4 => 104 bit = 13 bytes
  - 2 => 52 bit = 6.5 bytes (odd again)
    - Example: `86 10 44 9B 88 86 0B`
- Python implementation bchlib (use version 0.7)

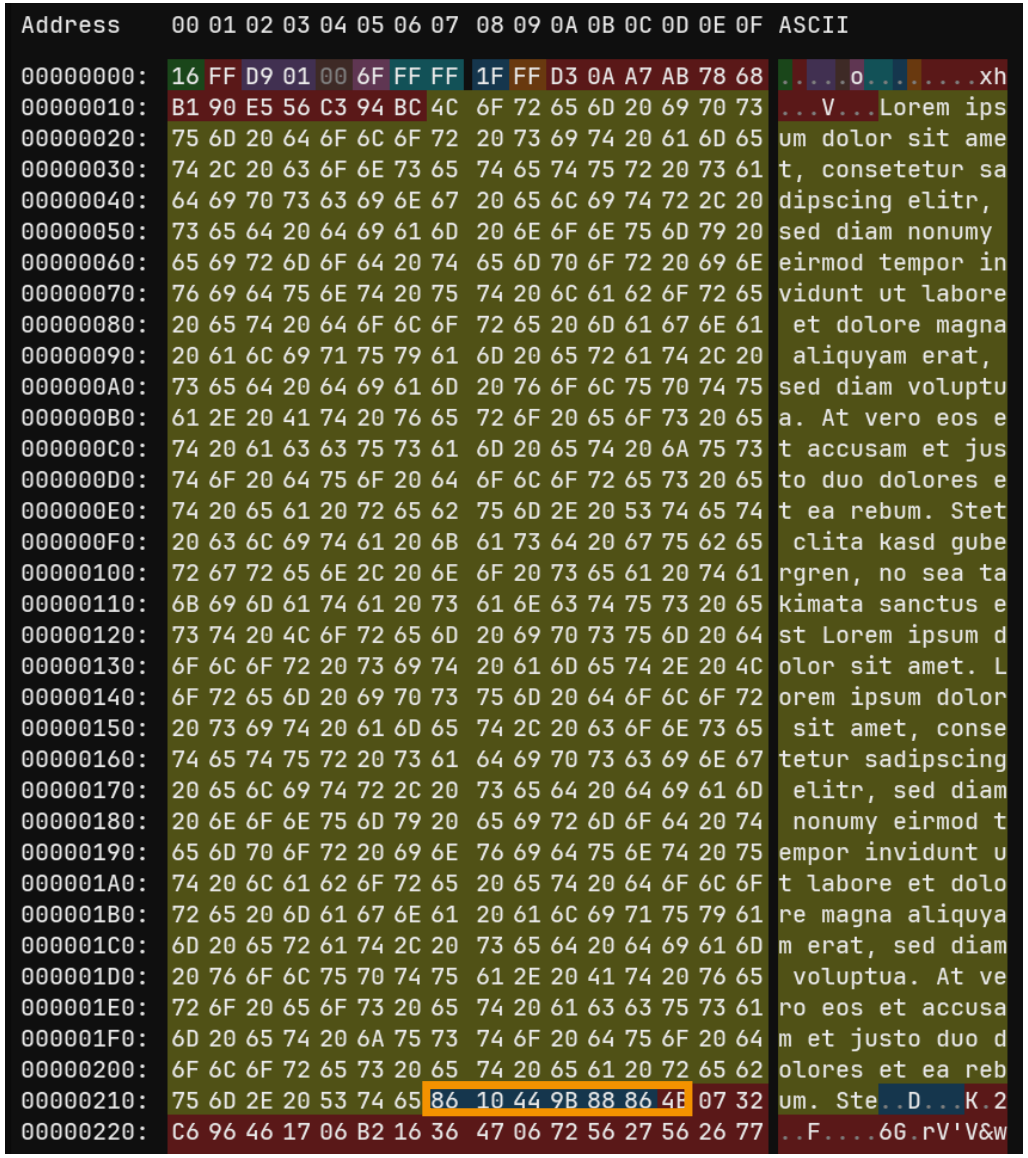
No byte alignment => sectors 2/4 off by 4 bit (!)

GPMI uses reversed bits per byte

- `0b00001011 (0x0B) => 0b11010000 (0xD0)`

Pseudocode:

```
sector_ecc_rev = reverse_bits(sector_ecc)
sector_ecc_rev_calc = bch.encode(reverse_bits(sector))
```



© SEC Consult - Public

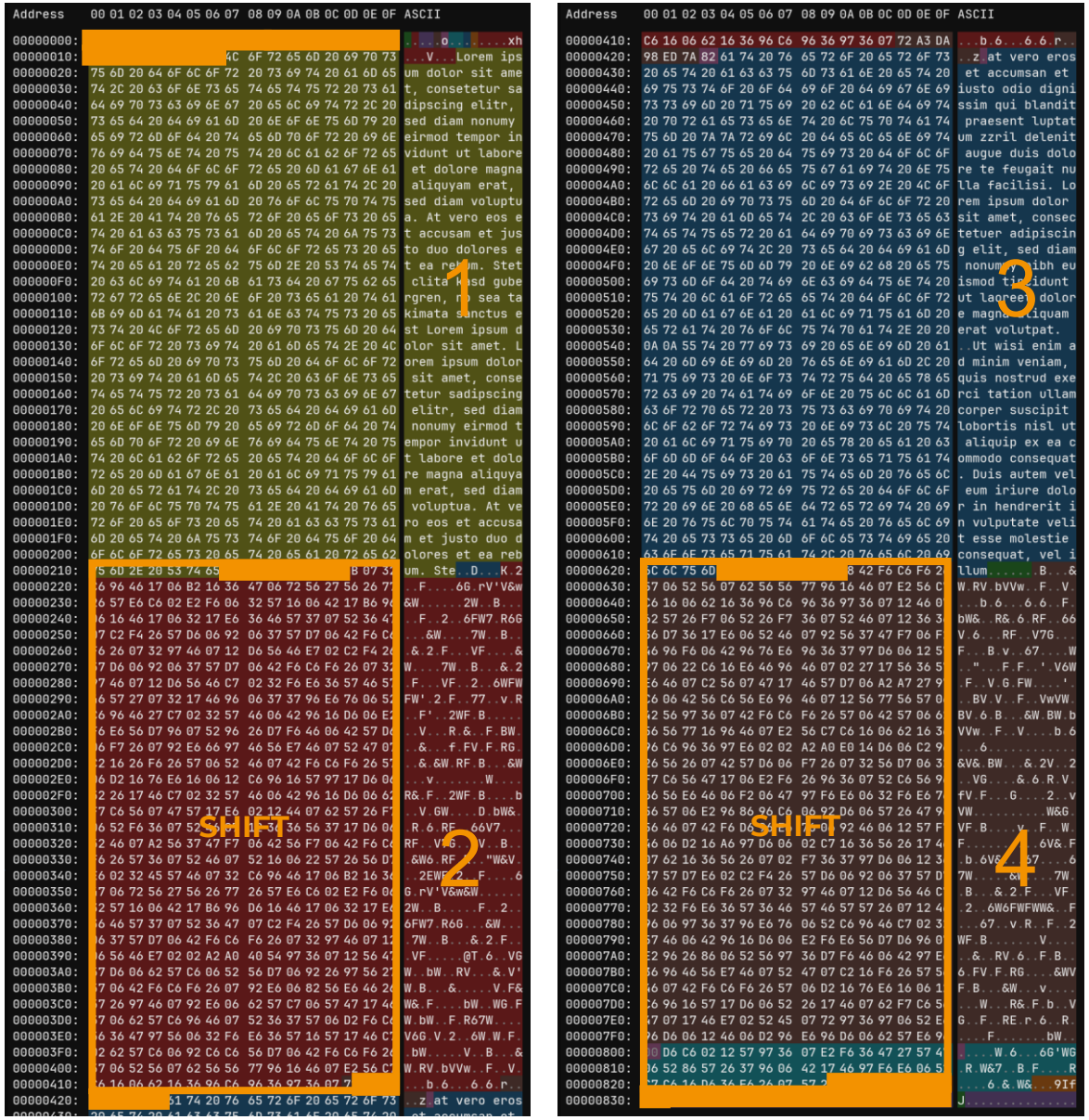
NAND Memory on NXP i.MX28

Strip OOB Data

In total

- 10B metadata
- 13B page ECC
- 4 times
  - 512B sector data
  - 6.5B sector ECC
- 15B unused

Leads to a 4-bit left shift of sectors 2 and 4



© SEC Consult - Public

But Now We Are Really Finished?

Approach

1. Parse FCB
2. Calculate ECC per sector
3. Swap BB per page
4. Strip OOB data
   - Metadata
   - ECC data
5. Extract
6. Profit

Filesize 512M

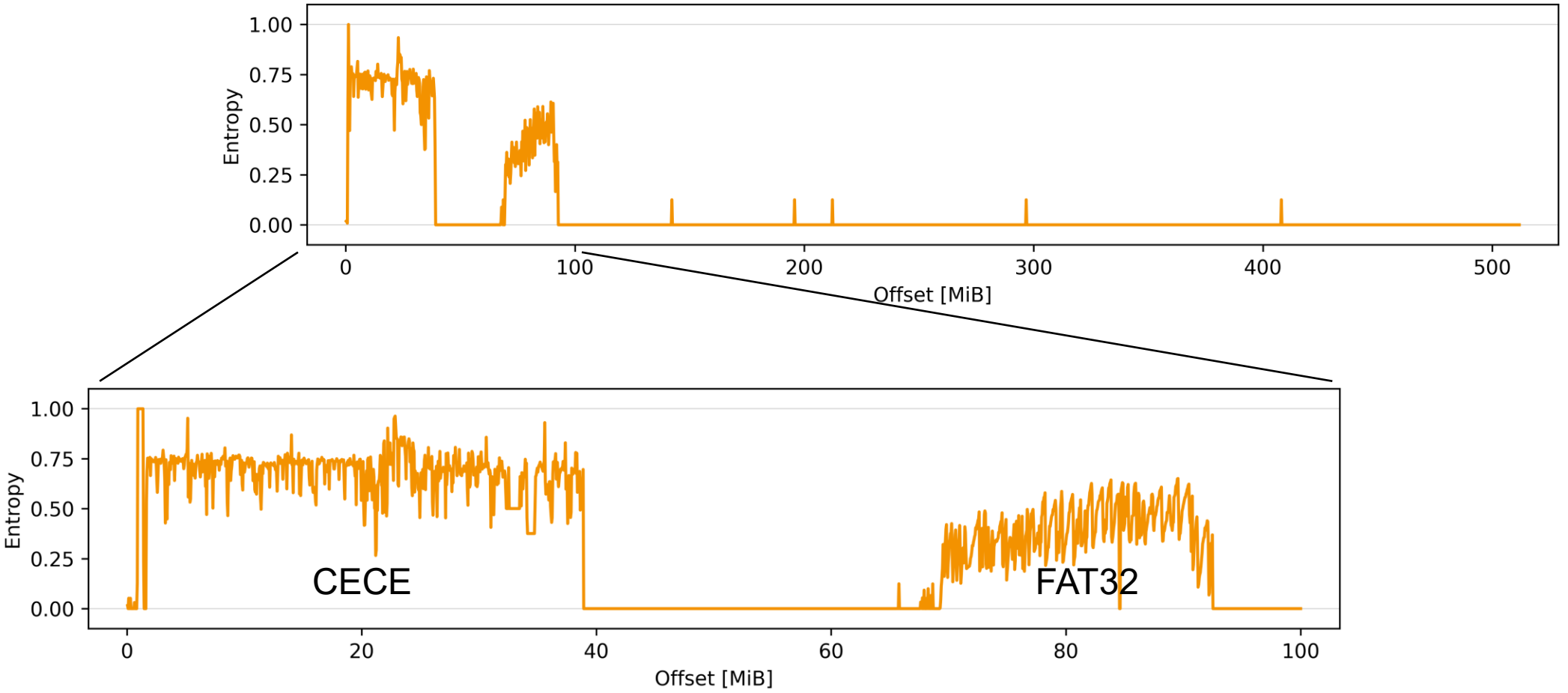
Unblob output

```
│ Extracted files: 1
│ Extracted directories: 0
│ Extracted links: 0
```

© SEC Consult - Public

Recovered Memory Image

Entropy



© SEC Consult - Public

Windows CE FAT32

"*Und bist du nicht willig, so brauch' ich AI*" | ("*And if you don't come willingly, I'll use AI.*")

Good

- The extracted file is a FAT filesystem

Bad

- Unblob could not extract the files
- Manual mounting fails with errors

Next step: Manual analysis

- FAT32 header exists
- There is data before the header
- Content looks scrambled
- Metadata might not be useless, let's try to eyeball this …

AI: "*show me a hexdump of the first 4096 bytes of a typical FAT32 filesystem*"

- First sector: FAT32 magic
- Second sector: FSInfo with `RRaA` pattern
- We can use this …

```
FIRST PAGE
------------------------------------------------------------------
52 52 61 41 00 00 00 00  00 00 00 00 00 00 00 00  RRaA............
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
[…]
------------------------------------------------------------------
LATER
------------------------------------------------------------------
EB FE 90 4D 53 57 49 4E  34 2E 31 00 08 01 20 00  ëþ.MSWIN4.1... .
01 00 00 00 00 F8 00 00  40 00 01 00 00 00 00 00  .....ø..@.......
80 5F 03 00 B0 01 00 00  00 00 00 00 02 00 00 00  ._..°...........
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
80 00 29 04 00 05 08 20  20 20 20 20 20 20 20 20  ..).....
20 20 46 41 54 33 32 20  20 20 00 00 00 00 00 00    FAT32   ......
[…]
------------------------------------------------------------------
```

© SEC Consult - Public

Windows CE FAT32

Recreate Logical Image

Approach

- Parse all numbers
- Look for patterns
  - Logical page number (0x0000)
    - Location
  - Logical sequence number (0x7c)
    - Higher for newer pages
  - Wrap counter (0x0)
    - Counts sequence wraps

Logical page number duplicates?

- Wear-levelling
- Sometimes copy on write

Quick & dirty

- Reorder by logical page number
- Use page with highest physical address for duplicates

```
00 FF 00 00 00 7C FF FF  00 FF 07 DB 38 16 9C 8C | .....|......8...
BD A4 16 5E 5F 98 CC EB  FE 90 4D 53 57 49 4E 34 | ...^_.....MSWIN4
|  |  |     |  |  |      |  |
+--|--|-----|--|--|------|--|--- unused (swapped BB)
   +--|-----|--|--|------|--|--- unused (0xff)
      +-----|--|--|------|--|--- logical page number
            +--|--|------|--|--- unused (0x00)
               +--|------|--|--- logical sequence number
                  +------|--|--- unused (0xffff)
                         +--|--- logical sequence wrap counter
                            +--- unused (0xff)
```

```
lpnum lseq lwrap bb paddr pbnum ppnum
[…]
028 011 00 00 0x04670700 546 028
029 012 00 00 0x04670f40 546 029
030 012 00 00 0x04671780 546 030
031 013 00 00 0x04671fc0 546 031
032 094 00 00 0x04672800 546 032
032 122 00 00 0x04773cc0 554 019
[…]
032 234 00 00 0x04779fc0 554 031
032 006 01 00 0x0477b880 554 034
[…]
032 230 01 00 0x04797600 555 024
032 002 02 00 0x04799700 555 028
[…]
032 109 06 00 0x05f59b00 739 044
033 010 00 00 0x04673040 546 033
034 010 00 00 0x04673880 546 034
035 011 00 00 0x046740c0 546 035
[…]
```

© SEC Consult - Public

Please Let Me Be Finished

Approach

1. Recover logical FAT image
2. Extract
3. Profit

Unblob output

```
│ Extracted files: 43
│ Extracted directories: 11
│ Extracted links: 0
│ Extraction directory size: 1.27 MB
```

```
$ ls
data
Log
settings
SoftwareUpdate
Temp
```

Where are my binaries?

© SEC Consult - Public

If you have no idea what you are doing …

Let's ask AI (an LLM)

Things to ask

- Things you have no clue about
- *"How does X work?"*
- *"Analyze this hexdump. It is from X and I guess it shows Y."*

Let it create

- Unknown C/Python struct definitions
- ImHex patterns
- Parsing script snippets/functions

Answer Quality?

- Most often 90% fit
- Correct the answer based on your real problem
- Makes a great sparring partner

AI: *"I have a hexdump from a windows ce device. at the beginning it shows an ECEC magic. how do I extract its files?"*

Caveats

- Stay within your domain
- Start over with a fresh context
- May provide anonymized data

```
FE 03 00 EA 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
45 43 45 43 24 A2 73 82  24 A2 53 02 00 00 00 00  ECEC$.s.$.S.....
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

© SEC Consult - Public

Windows CE ROM

- Proprietary file data structure
- Read only
- Execute-in-place (XIP)
- Header + ToC (somewhere)

Approach

- Use previous work (and fail)
- Understand the structure
  - AI
  - Previous work
  - BSP by NXP
  - Windows CE source code
- Analyse with ImHex
- Build own parsing script

Previous Work

- dumprom / eimgfs
- https://itsme.home.xs4all.nl/projects/xda/dumprom.html
- https://github.com/nlitsme/eimgfs/
- WinCE Extractor
- https://github.com/KodaSec/wince-extractor

Problems

- 32-bit structures
- Build failures
- Targeted on PDAs/phones (other formats?)

```
FE 03 00 EA 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
45 43 45 43 24 A2 73 82  24 A2 53 02 00 00 00 00  ECEC$.s.$.S.....
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

© SEC Consult - Public

Windows CE ROM

ROMHDR

Value after ECEC magic points to ROMHDR structure

Information gathered

- 0x80200000 load offset
- 182 modules
- 83 files

```
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040  45 43 45 43 24 A2 73 82  24 A2 53 02 00 00 00 00  ECEC$.s.$.S.....
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

```
0253A220  00 00 00 00 34 EF 01 40  00 F0 ED 40 00 00 20 80  ....4..@...@.. .
0253A230  4C C2 73 82 B6 00 00 00  00 00 74 82 00 00 75 82  L.s.......t...u.
0253A240  00 00 00 88 02 00 00 00  78 C9 83 80 00 00 00 00  ........x.......
0253A250  00 00 00 00 53 00 00 00  00 00 00 00 40 40 40 40  ....S.......@@@@
0253A260  00 00 00 00 00 00 00 00  C2 01 02 00 E0 10 20 80  ............ .
0253A270  00 00 00 00 00 00 00 00  07 00 00 00 D1 69 11 64  .............i.d
0253A280  87 D0 DB 01 00 20 01 00  F4 0F 57 80 70 DF 3D 81  ..... ....W.p.=.
0253A290  A0 4F 6E 80 00 00 20 80  07 00 00 00 D4 85 07 35  .On... ........5
```

```
struct S_ROMHDR {
    u32 dllfirst;
    u32 dlllast;
    u32 physfirst;
    u32 physlast;
    u32 nummods;
    u32 ulRAMStart;
    u32 ulRAMFree;
    u32 ulRAMEnd;
    u32 ulCopyEntries;
    u32 ulCopyOffset;
    u32 ulProfileLen;
    u32 ulProfileOffset;
    u32 numfiles;
    u32 ulKernelFlags;
    u32 ulFSRamPercent;
    u32 ulDrivglobStart;
    u32 ulDrivglobLen;
    u16 usCPUType;
    u16 usMiscFlags;
    u32 pExtensions;
    u32 ulTrackingStart;
    u32 ulTrackingLen;
};
```

© SEC Consult - Public

Windows CE ROM

Parse Modules

First module ToC entry starts immediately after ROMHDR

Content

- DLL
- EXE
- Without PE header

Information gathered from ToC entry

- Pointer to filename
- Pointer to O32 structure

```
struct S_TOCentry {
    u32 dwFileAttributes;
    u64 ftTime;
    u32 nFileSize;
    u32 lpszFileName;
    u32 ulE32Offset;
    u32 ulO32Offset;
    u32 ulLoadOffset;
};
```

```
0253A220  00 00 00 00 34 EF 01 40  00 F0 ED 40 00 00 20 80  ....4..@...@.. .
0253A230  4C C2 73 82 B6 00 00 00  00 00 74 82 00 00 75 82  L.s.......t...u.
0253A240  00 00 00 88 02 00 00 00  78 C9 83 80 00 00 00 00  ........x.......
0253A250  00 00 00 00 53 00 00 00  00 00 00 00 40 40 40 40  ....S.......@@@@
0253A260  00 00 00 00 00 00 00 00  C2 01 02 00 E0 10 20 80  ............ .
0253A270  00 00 00 00 00 00 00 00  07 00 00 00 D1 69 11 64  .............i.d
0253A280  87 D0 DB 01 00 20 01 00  F4 0F 57 80 70 DF 3D 81  ..... ....W.p.=.
0253A290  A0 4F 6E 80 00 00 20 80  07 00 00 00 D4 85 07 35  .On... ........5
```

Parsing:

```
tocentry @ 0x0253a278
dwFileAttributes: 0x7
ftTime: 0x1dbd087641169d1
nFileSize: 0x12000
lpszFileName: 0x80570ff4
ulE32Offset: 0x813ddf70
ulO32Offset: 0x806e4fa0
ulLoadOffset: 0x80200000
```

© SEC Consult - Public

Windows CE ROM

O32 Structure

Another indirection supporting “virtual memory”

Information gathered

- Virtual address offset
- Pointer to file data

Recovery steps

- Use filename from before
- Extract file data to virtual offset
- Rebuild PE header (optional)

```
struct S_O32 {
    u32 o32_vsize;
    u32 o32_rva;
    u32 o32_psize;
    u32 o32_dataptr;
    u32 o32_realaddr;
    u32 o32_flags;
};
```

```
004E4F90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
004E4FA0  B0 DA 00 00 00 10 00 00  00 E0 00 00 00 10 20 80  °Ú.......à.... .
004E4FB0  00 10 20 80 20 00 00 60  00 60 00 00 00 F0 00 00  .. . ..`.`...ð..
004E4FC0  00 00 00 00 F4 0F 57 80  00 00 74 82 80 00 00 C0  ....ô.W...t....À
```

Parsing:

```
o32_rom @ 0x004e4fa0
o32_vsize: 0xdab0
o32_rva: 0x1000
o32_psize: 0xe000
o32_dataptr: 0x80201000
o32_realaddr: 0x80201000
o32_flags: 0x60000020
```

© SEC Consult - Public

Windows CE ROM

Parse Files

First file ToC entry starts immediately after last module ToC entry

Content

- Images
- Text files
- …
- Also executables (with header)

Information gathered

- Compression (none)
- Virtual address offset
- Pointer to file data

Recovery steps

- Extract file name
- Extract file data

```
struct S_FILESentry {
    u32 dwFileAttributes;
    u64 ftTime;
    u32 nRealFileSize;
    u32 nCompFileSize;
    u32 lpszFileName;
    u32 ulLoadOffset;
};
```

```
0253B920  87 D0 DB 01 00 50 00 00  84 D9 A6 80 50 BE 78 80  .....P......P.x.
0253B930  C0 BE 78 80 00 D0 93 81  07 00 00 00 00 33 69 6D  ..x..........3im
0253B940  0C BF D3 01 00 50 04 00  00 50 04 00 94 D9 A6 80  .....P...P......
0253B950  00 0A 5D 81 01 00 00 00  D2 9B EC 14 85 D0 DB 01  ..].............
0253B960  1C 3A 00 00 1C 3A 00 00  C8 D9 A6 80 00 5A 61 81  .:...:.......Za.
```

Parsing:

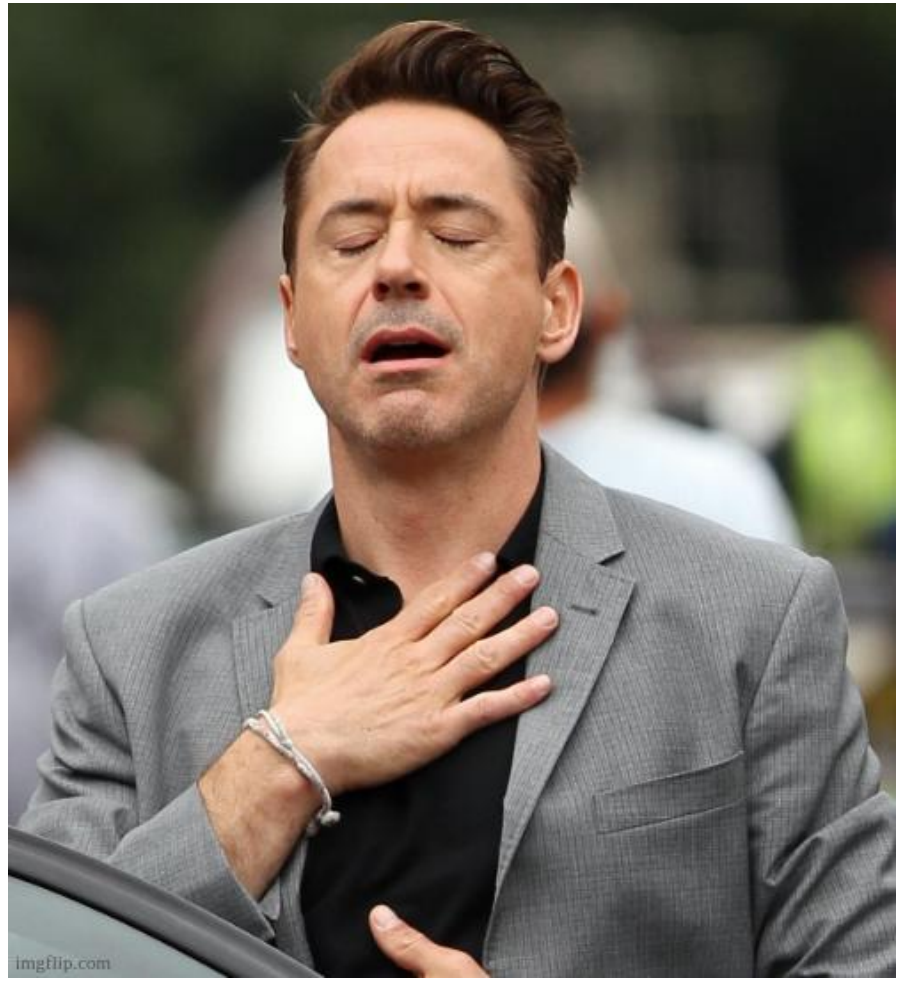
```
filesentry @ 0x0253b938
dwFileAttributes: 0x7
ftTime: 0x1d3bf0c6d693300
nRealFileSize: 0x45000
nCompFileSize: 0x45000
lpszFileName: 0x80a6d994
ulLoadOffset: 0x815d0a00
```

© SEC Consult - Public

We Are Finally Finished (Trust Me)

Approach

1. Find ROMHDR
2. Parse ROMHDR
3. Parse table of content
4. Parse toc module entries
   - Discover name
   - Discover content
   - Rebuild content
   - Rebuild PE header
   - Extract files
5. Parse toc file entries
   - Discover name
   - Discover content
   - Extract files
6. Profit

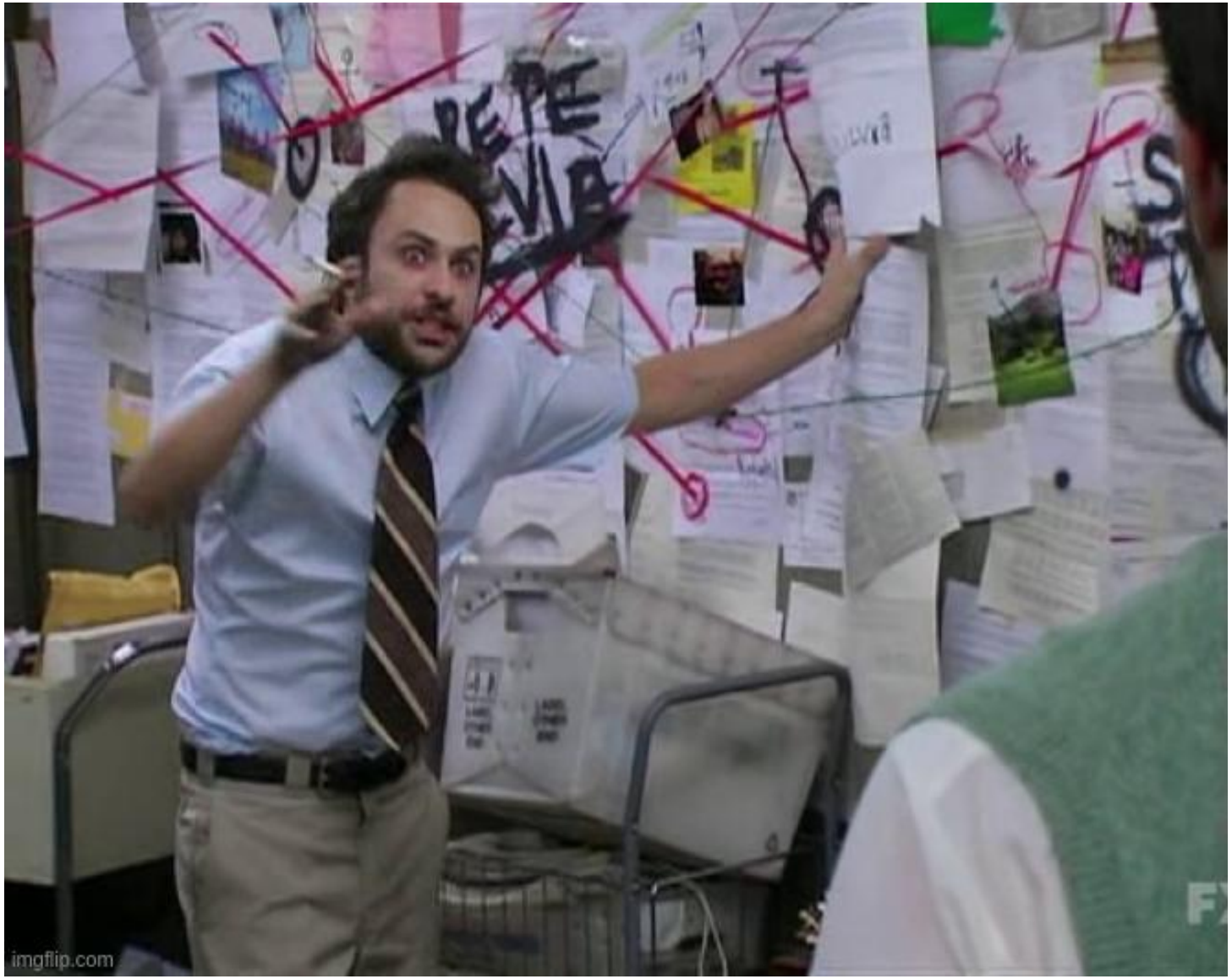


© SEC Consult - Public

Recap

What did we achieve

- Read memory from chip
- Parse FCB
- Calculate ECC per sector
- Swap BB per page
- Strip OOB data
  - Metadata
  - ECC data
  - Shift blocks
- Recover logical FAT image
  - Extract 43 files
- Recover ROM table of content
- Recover module entries
  - Extract 182 files
- Recover file entries
  - Extract 83 files



© SEC Consult - Public

SEC Consult
an Atos business

Questions?

SEC Consult
an Atos business

SEC Consult

an Atos business

Thank you!

Dou you have any further questions?
For more information please contact:

Gerhard Hechenberger
Principal Security Consultant
g.hechenberger@sec-consult.com

Confidential information owned by SEC Consult, an Atos business, to be used by the recipient only. This document, or any part of it, may not be reproduced, copied, circulated and/or distributed nor quoted without prior written approval from SEC Consult.