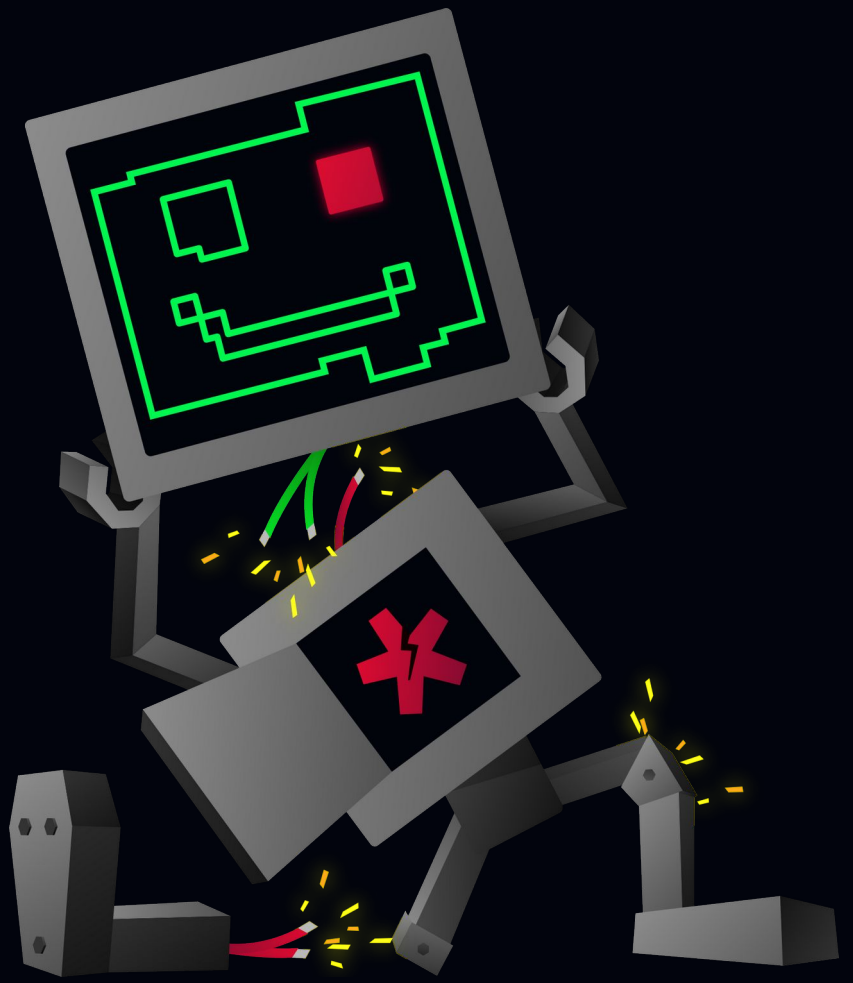
THE H@CK SUMMIT



C:\>

Unveiling the Secrets of Hardware DRM in Printers

Wojciech Cybowski
Cybersecurity Engineer, Nethone

thehacksummit.com 19-20 października 2023 PGE Narodowy + Online

ORGANIZATORZY:

ACADEMIC PARTNERS

whoami

- Wojciech Cybowski
- Cybersecurity Engineer @ Fraud Intelligence @ Nethone
- Privately an enthusiast of reverse engineering hardware and software, as well as creating new things.
- All the presented materials (and more!) will be available here: www.github.com/wcyb/cartridge_chip_resetter
- www.linkedin.com/in/wojciech-cybowski

Agenda

1. What are DRMs?
2. What is the difference between software DRM and hardware DRM?
3. Where can we find hardware DRMs?
4. DRMs in printers - types and how they work.
5. Analysis with examples.
6. Summary.

What are DRMs?

- Wikipedia sums it up nicely in one sentence: “*access control technologies that limit the usage of digital content and devices”*.
- It allows the manufacturer to control access to the content and products we purchase.
- When we buy a product or software, it is the manufacturer who decides whether we can use it.

What is the difference between software DRM and hardware DRM?

Implementation:

- Hardware DRM is implemented in physical devices, such as dedicated chips or dongles, that are separate from the main computing device.
- Software DRM, on the other hand, relies on code and is integrated into the software itself, which is executed on the user's device.

What is the difference between software DRM and hardware DRM?

Security Level:

- Hardware DRM typically provides a higher level of security because it's harder to tamper with physical devices, making it more resistant to hacking or circumvention.
- Software DRM, while still offering protection, may be more vulnerable to reverse engineering and hacking attempts, as the protection mechanisms are embedded in software that runs on a user's device.

What is the difference between software DRM and hardware DRM?

Flexibility:

- Hardware DRM can be less flexible because it often requires dedicated hardware components, which may limit its compatibility with a wide range of devices.
- Software DRM is more versatile as it can be implemented on various platforms and can adapt to different hardware configurations, making it more accessible and user-friendly.

What is the difference between software DRM and hardware DRM?

Cost and Maintenance:

- Hardware DRM typically involves higher initial costs for developing and manufacturing the physical hardware components, as well as potential maintenance issues if updates are required.
- Software DRM is generally more cost-effective to implement and update, as changes can be made through software patches or updates without the need for physical replacements.

Where can we find hardware DRMs?

Gaming Consoles:

- Gaming consoles like the Xbox and PlayStation often employ hardware DRM to protect against software piracy and unauthorized game copies.

Set-Top Boxes:

- Set-top boxes used for digital television services usually incorporate hardware DRM to safeguard digital content, including premium channels and on-demand programming.

Where can we find hardware DRMs?

Electronic components:

- Whether it's a laptop battery or any part for Apple equipment or electronic car parts, nowadays you will most often find some kind of authorization, activation and control system, of which integrated circuits will be a part and at the same time the carrier of digital ID.

Cartridges:

- That is, something the average computer user will come into contact with numerous times, as well as the topic of this talk.

DRMs in printers - types and how they work

Type	Effectiveness
ASIC	Very high
Hardware-encrypted EEPROM	High
RFID tag	High
EEPROM	Medium
Fuse	Low

Analysis with examples

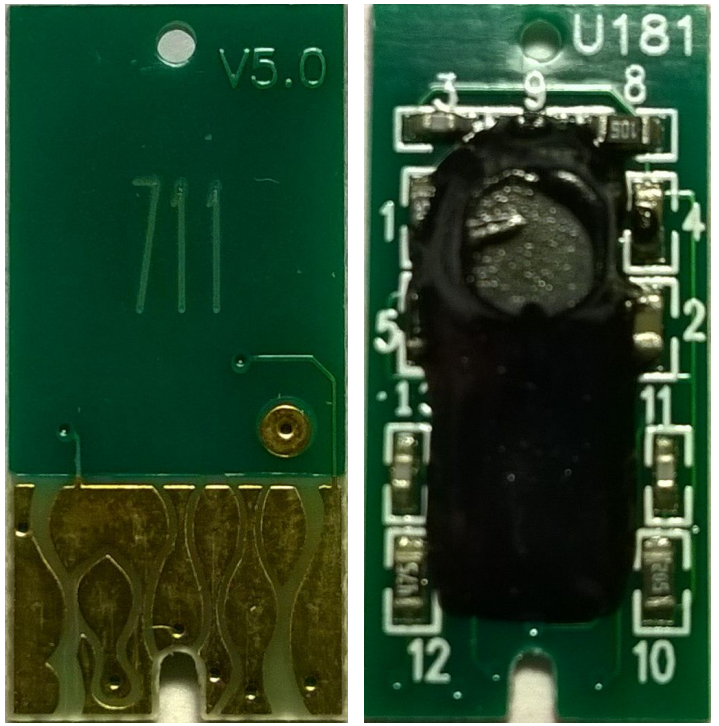
DRM 1

Analysis with examples

What can be noticed?

- Shape and arrangement of terminals.
- A fair number of surface-mounted components.
- Epoxy resin potting.

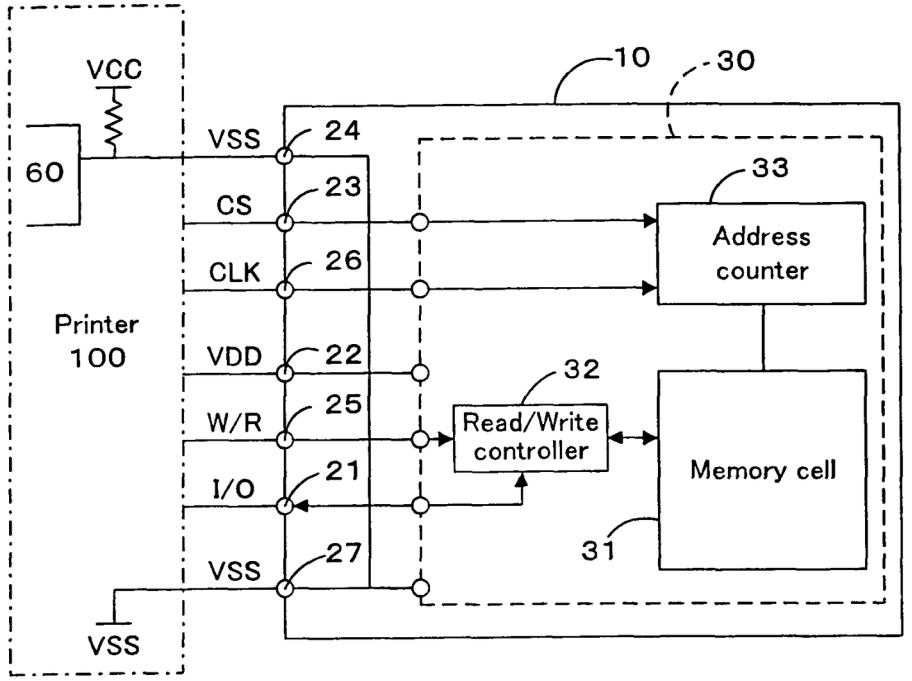
What can we do in this situation?



Analysis with examples

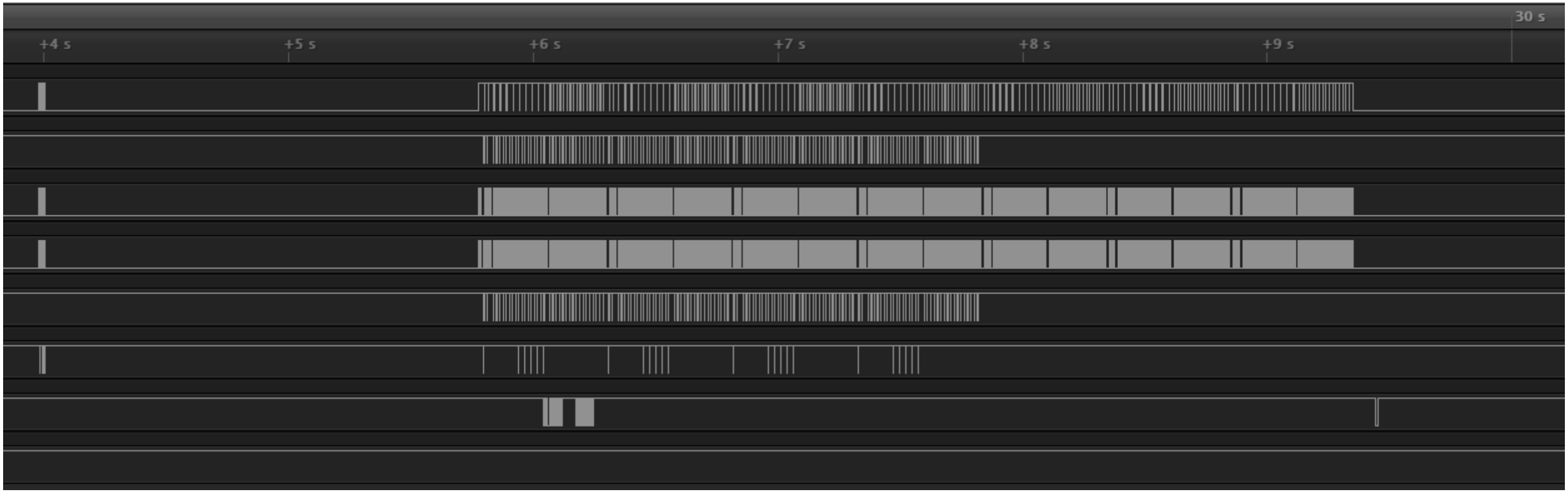
- Check the voltages at the connector on the printer first.
- How about patents? US7125100B2!
- We know what to expect,
  it's time to hook up the logic analyzer.

Fig.4



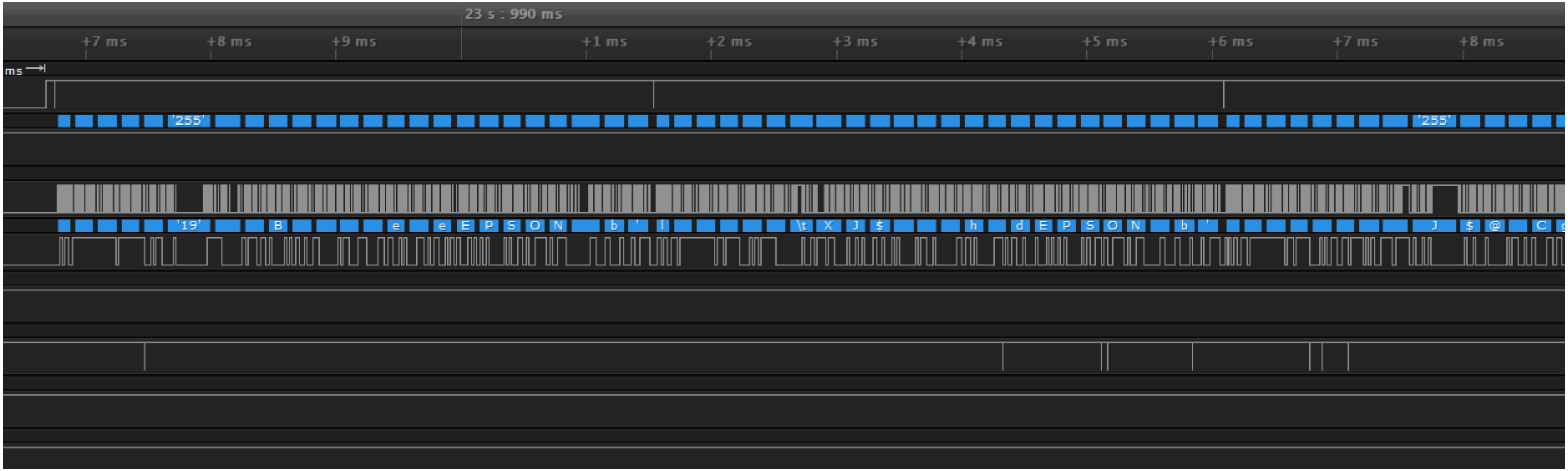
Analysis with examples

- Not so fast! Where do we find the data we are interested in?



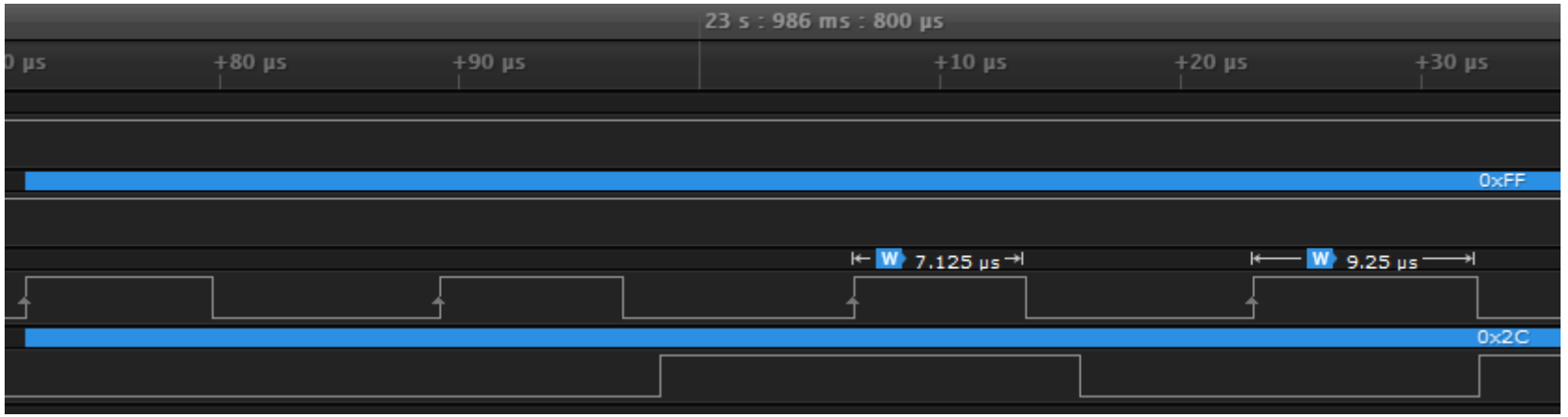
Analysis with examples

- Almost at the beginning. Just set the appropriate decoder options and you're done.



Analysis with examples

- It seemingly looks like SPI, but it's not quite so.
- Here one data line is used for transmitting and receiving - as in the I2C protocol.
- What about addressing? If the data line is common to all chips, how is the printer able to address the selected one?



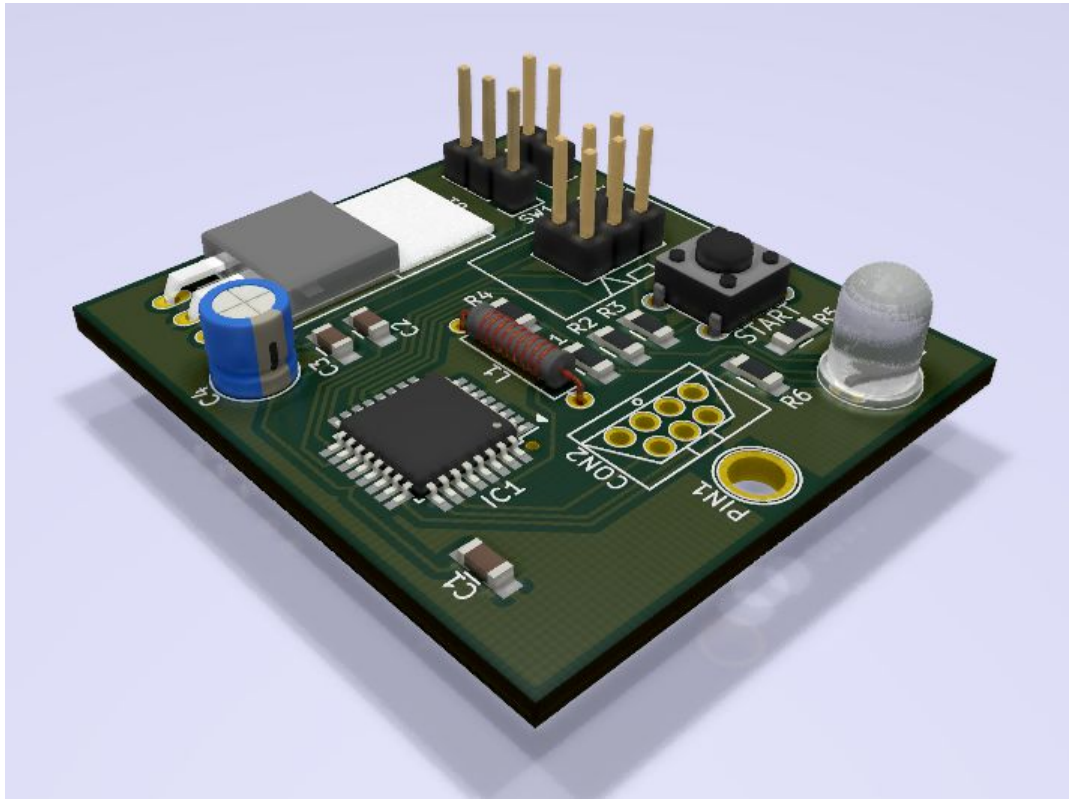
Analysis with examples

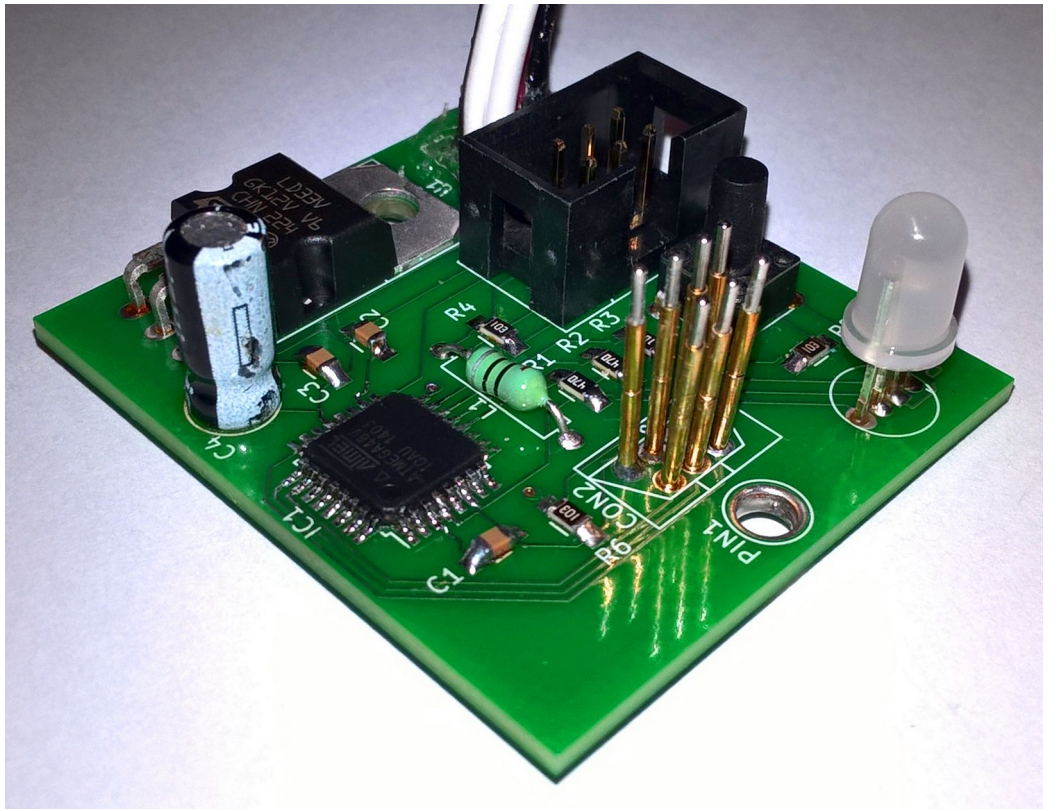
- What data does the chip memory contain?
- How can we determine what is what?

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x00	Chip ID	Ink dot counter			AU	Static data		
0x08	Static data			0xB2	Ink color		Static data	
0x10	„EPSON” string					Static data		
0x18	0x00							

Analysis with examples

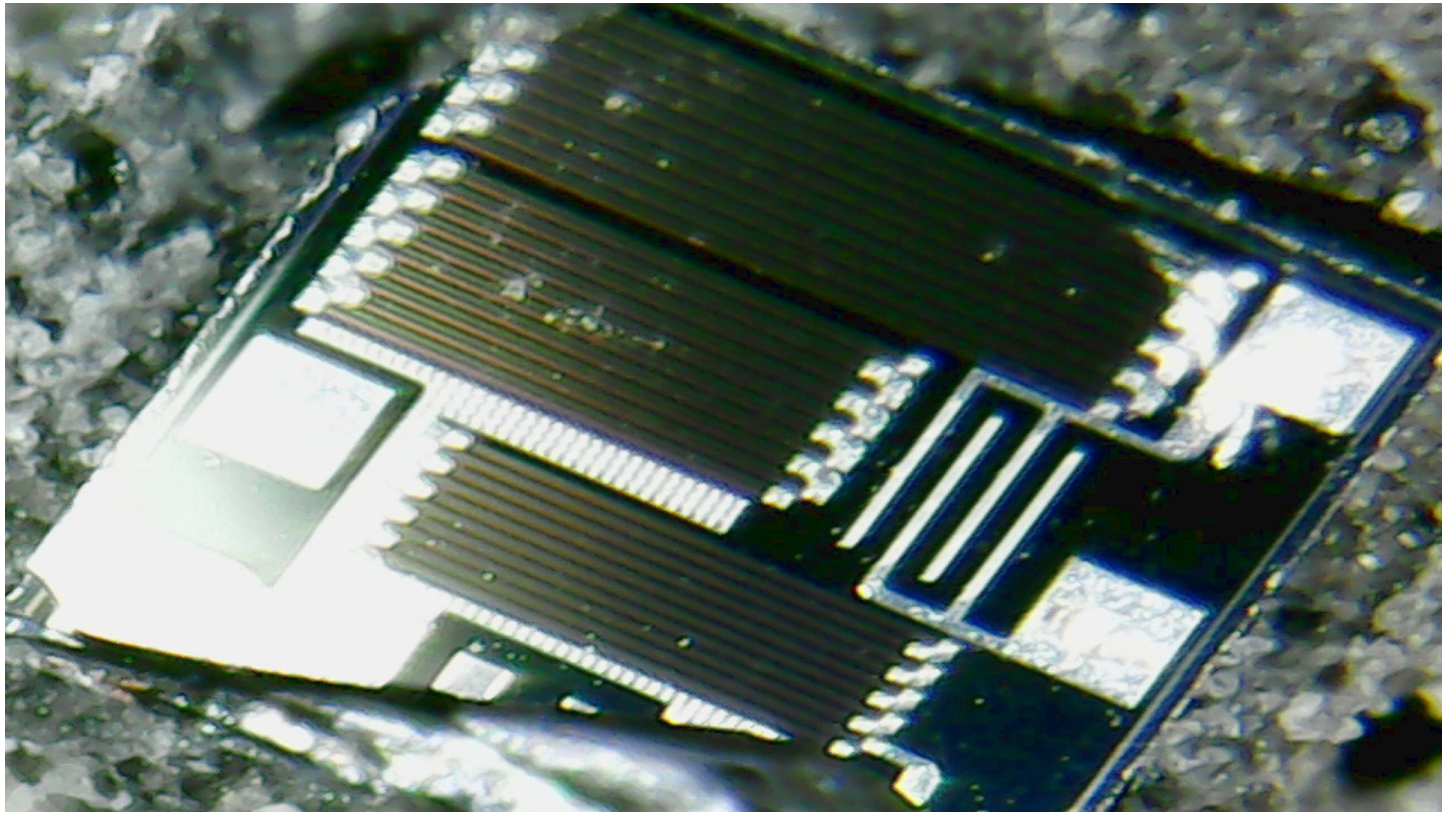
- It's time to create a resetter!





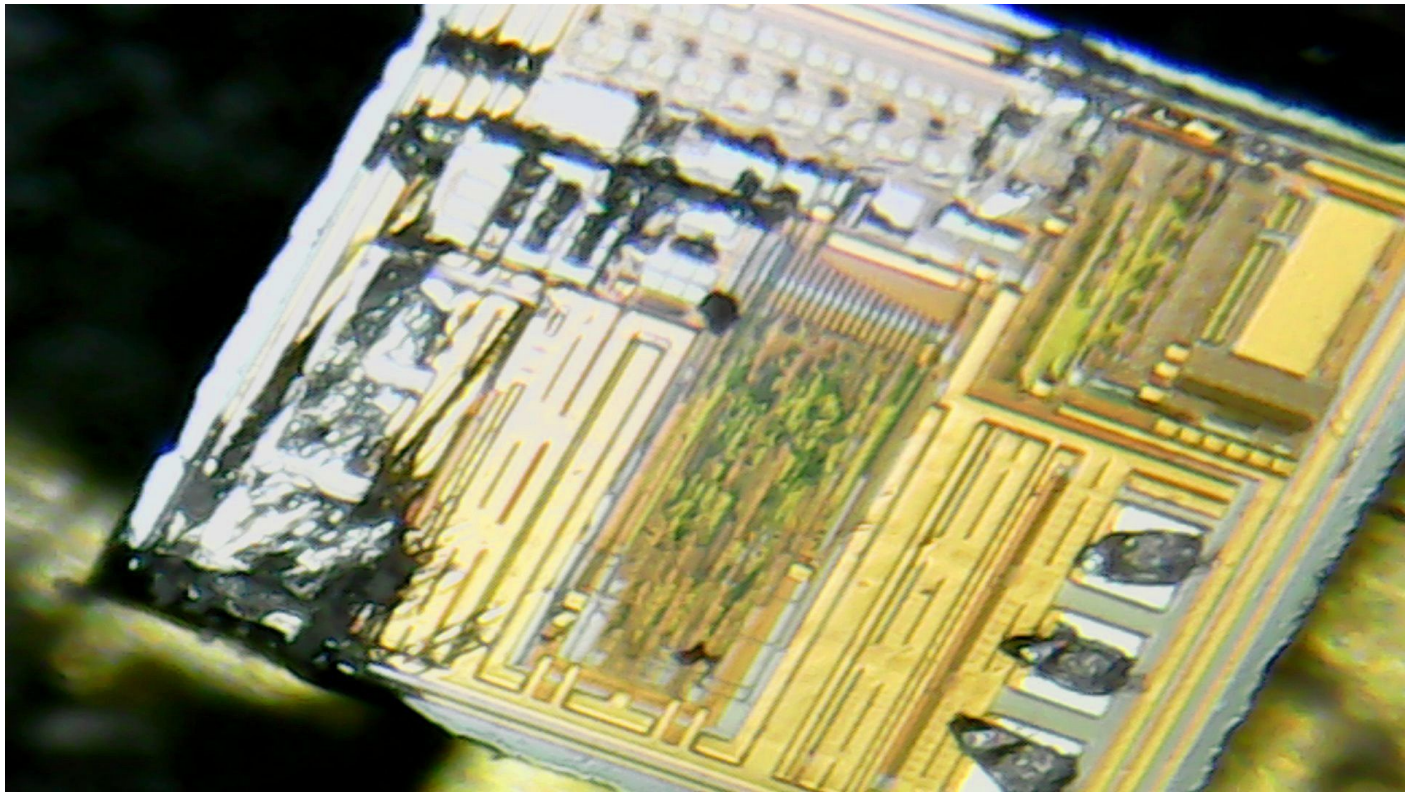
Analysis with examples

- What is underneath the epoxy resin?



Analysis with examples

- What is underneath the epoxy resin?

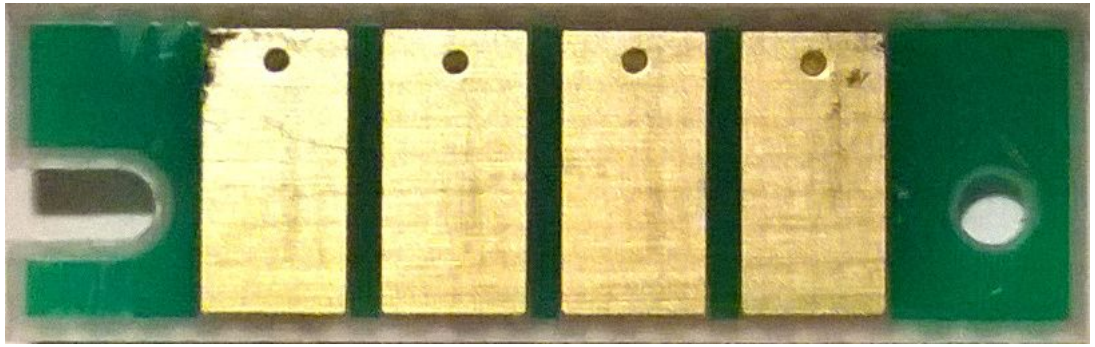


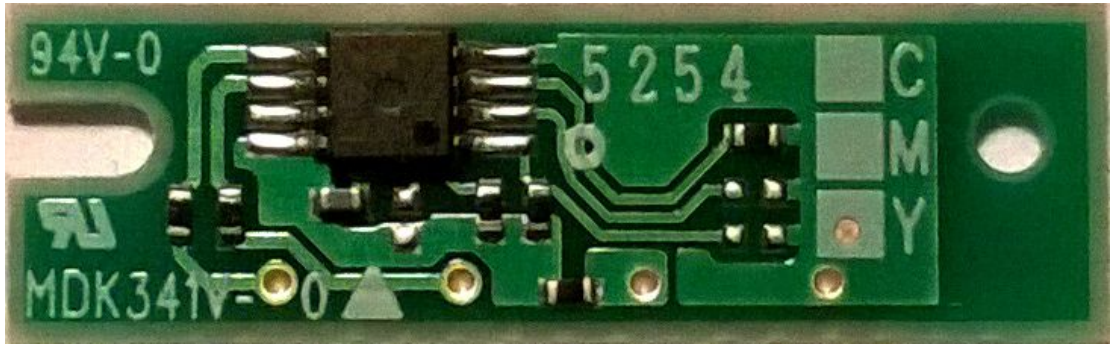
Analysis with examples

DRM 2

Analysis with examples

- *What can be noticed?*
  - Only four terminals.
  - A small number of surface-mounted components.
  - Visible IC marking.
- *What can we do in this situation?*





Analysis with examples

- It is not so bad - it is a simple EEPROM 24C02 memory. We can read and write it without any problems.
- The data analysis is the same as in the previous case.
- In this case, we do not have to worry about an unusual interface or method of communication. Here everything is standard.

Analysis with examples

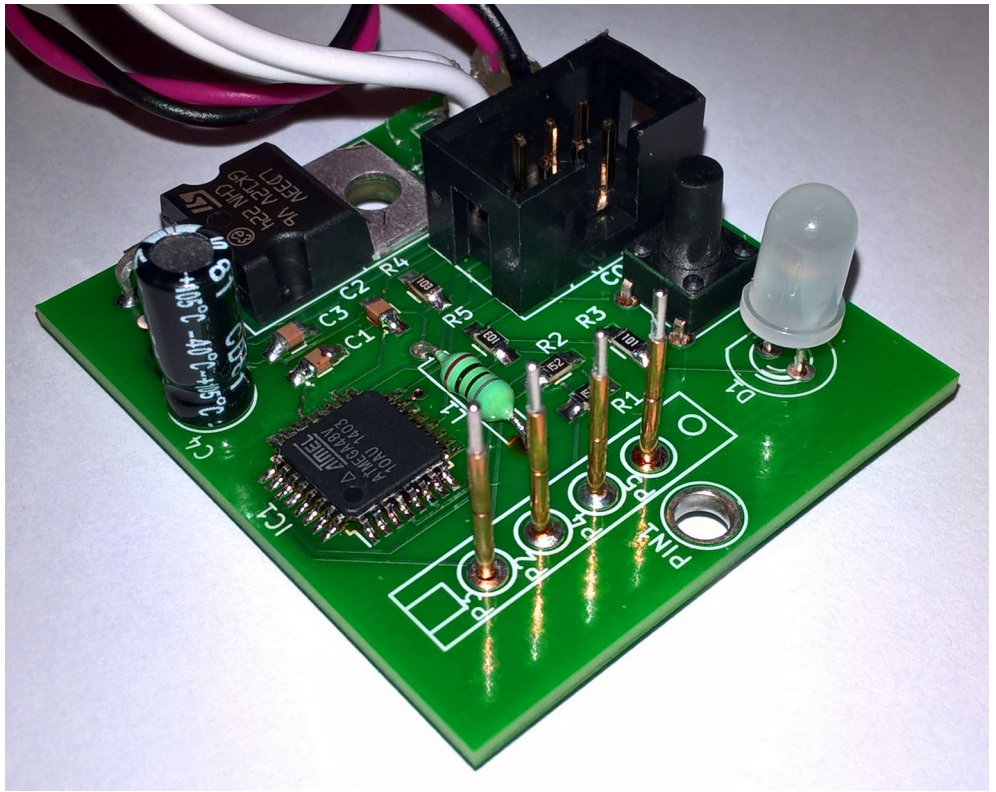
- Now we can create a data map…

	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07
0x00	M	CV	B	Region	Cartridge type			0x00
0x08	IT	0x00	EDP code (407166)					
0x10	Cartridge serial number							
0x18	0x00				Control number			
0x20	Printer serial number							
0x28					PT	OW	0x00	

⋮

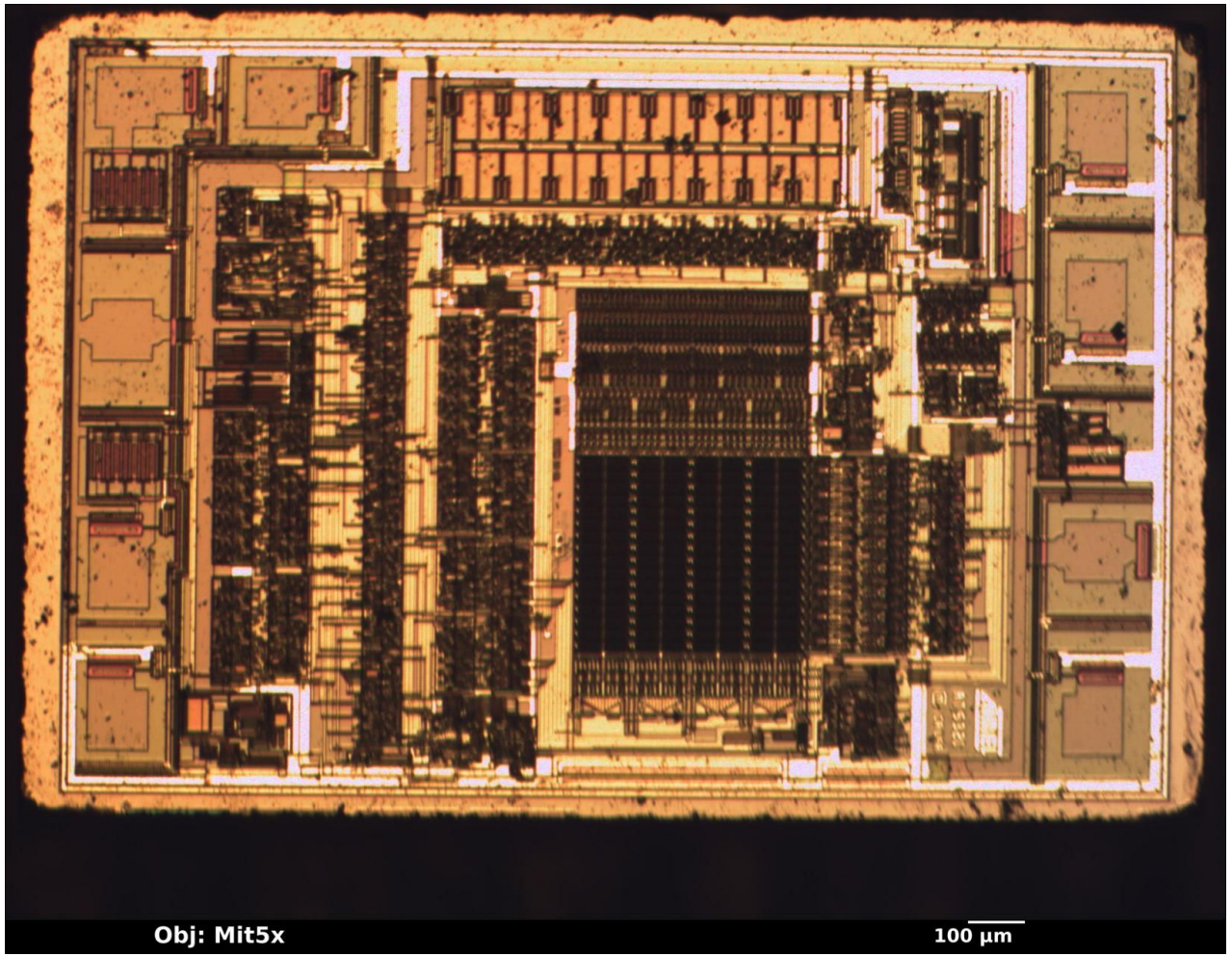
Analysis with examples

- ...and a resetter!



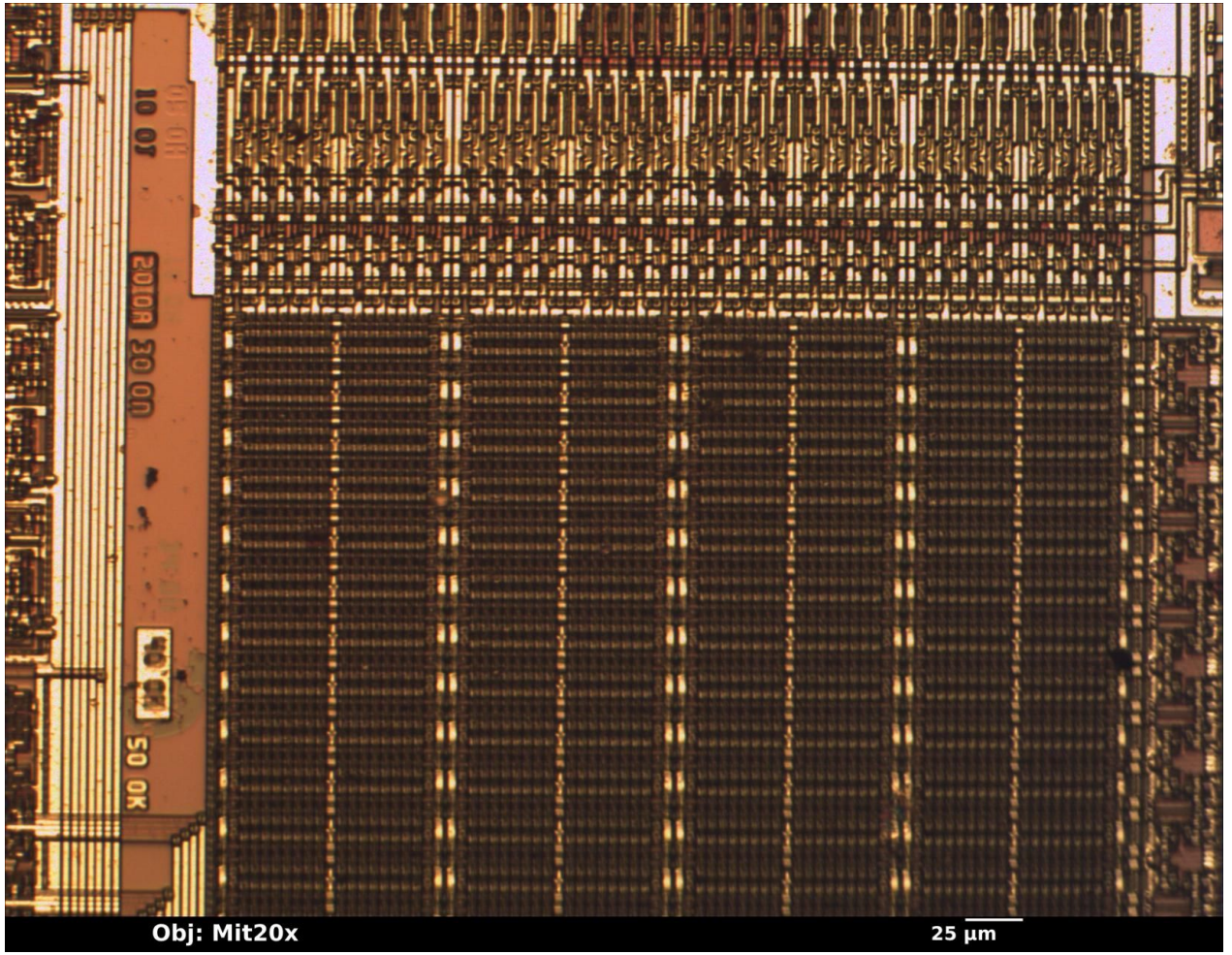
Analysis with examples

- Silicon of ordinary 24C02 EEPROM
- *Source:*
  https://siliconpr0n.org/archive/doku.php?id=mcmaster:atmel:648-24c02



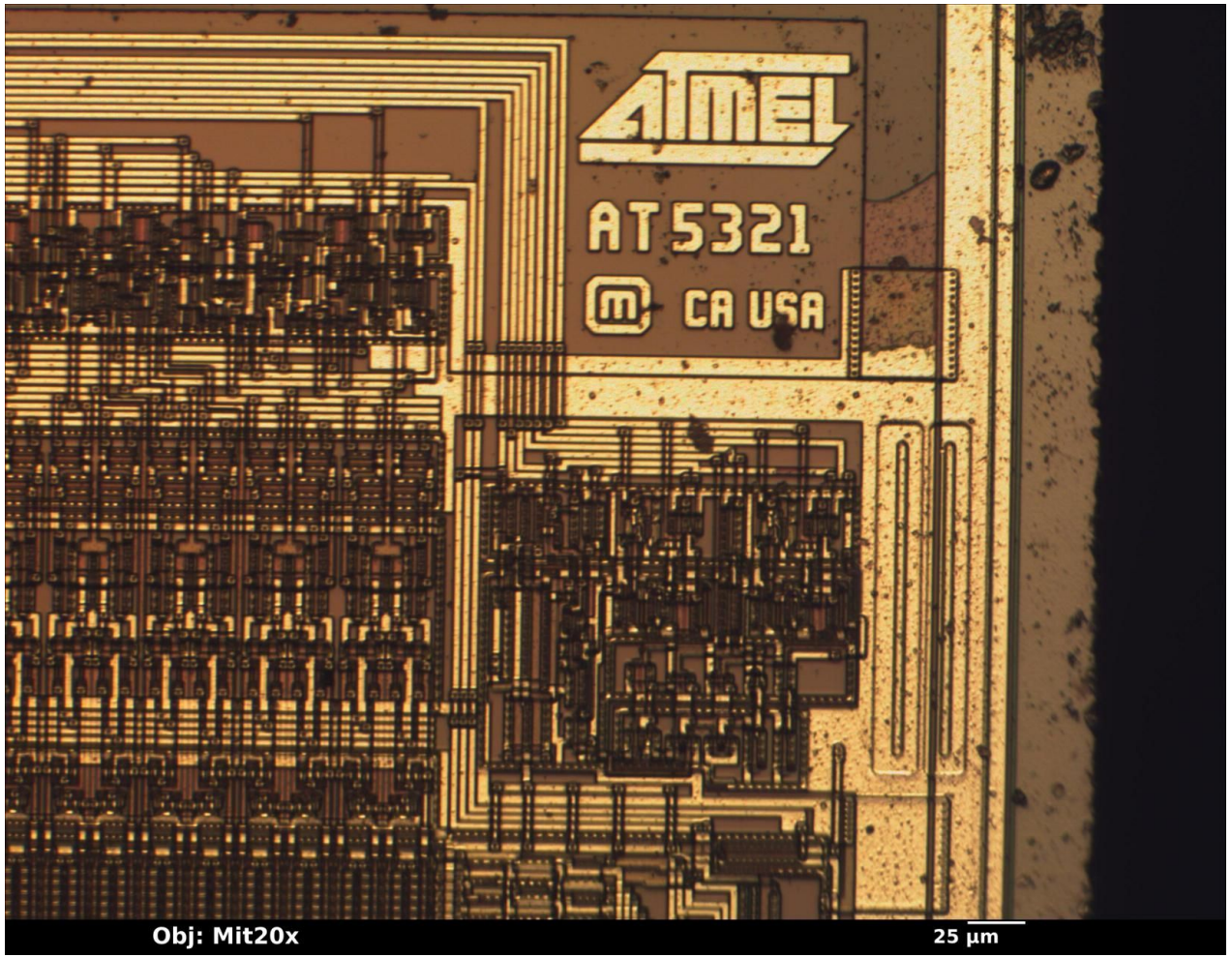
Analysis with examples

- Silicon of ordinary 24C02 EEPROM
- *Source:*
  https://siliconpr0n.org/archive/doku.php?id=mcmaster:atmel:648-24c02



Analysis with examples

- Silicon of ordinary 24C02 EEPROM
- *Source:*
  https://siliconpr0n.org/archive/doku.php?id=mcmaster:atmel:648-24c02



Analysis with examples

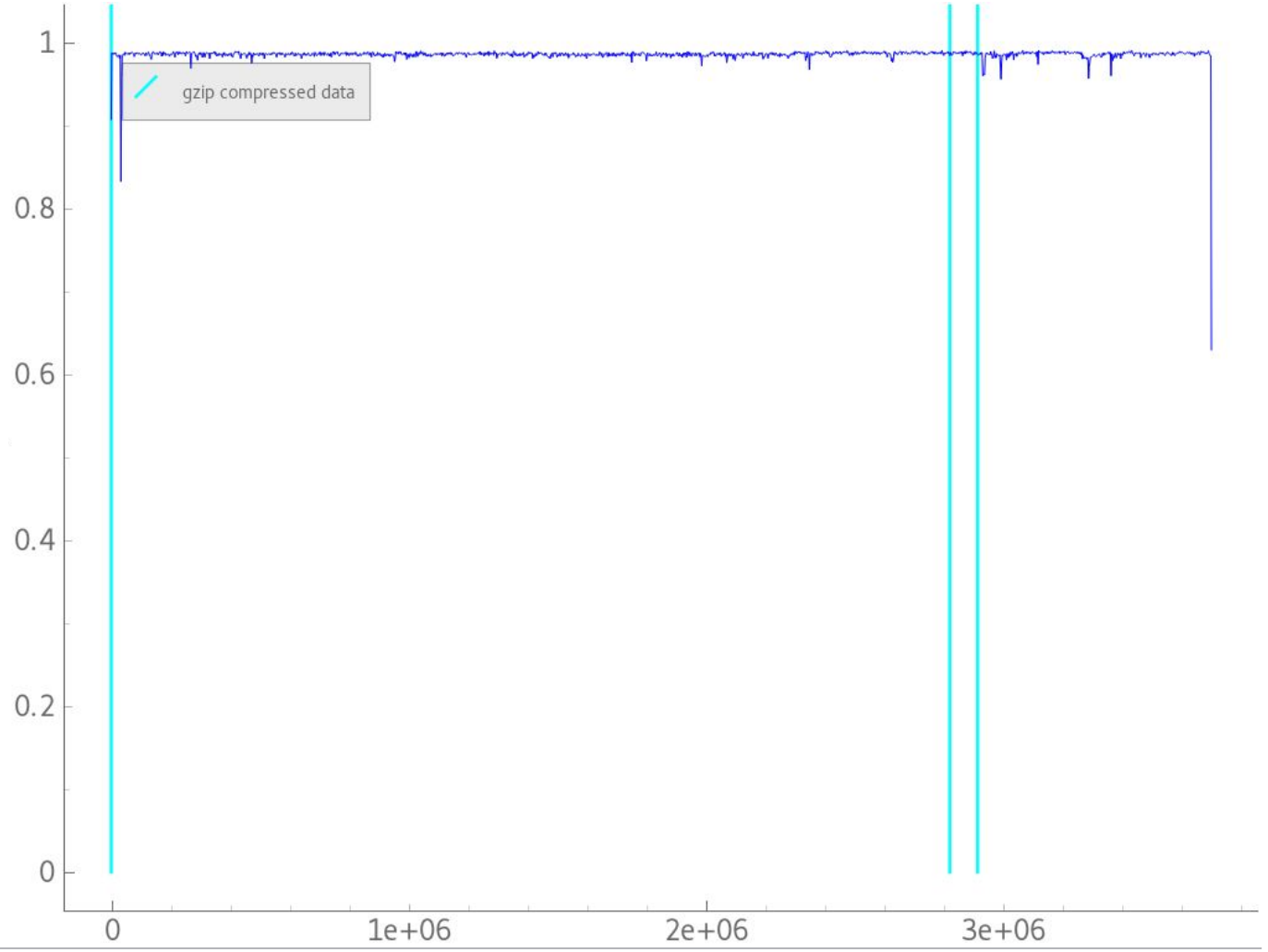
DRM 3

Analysis with examples

- This time, although the printer was a completely different type (gel), the chip was the same.
- For this reason, the analysis went the same as before.
- However, in this case, a firmware update was available - why not take advantage and see if we can find something useful there?

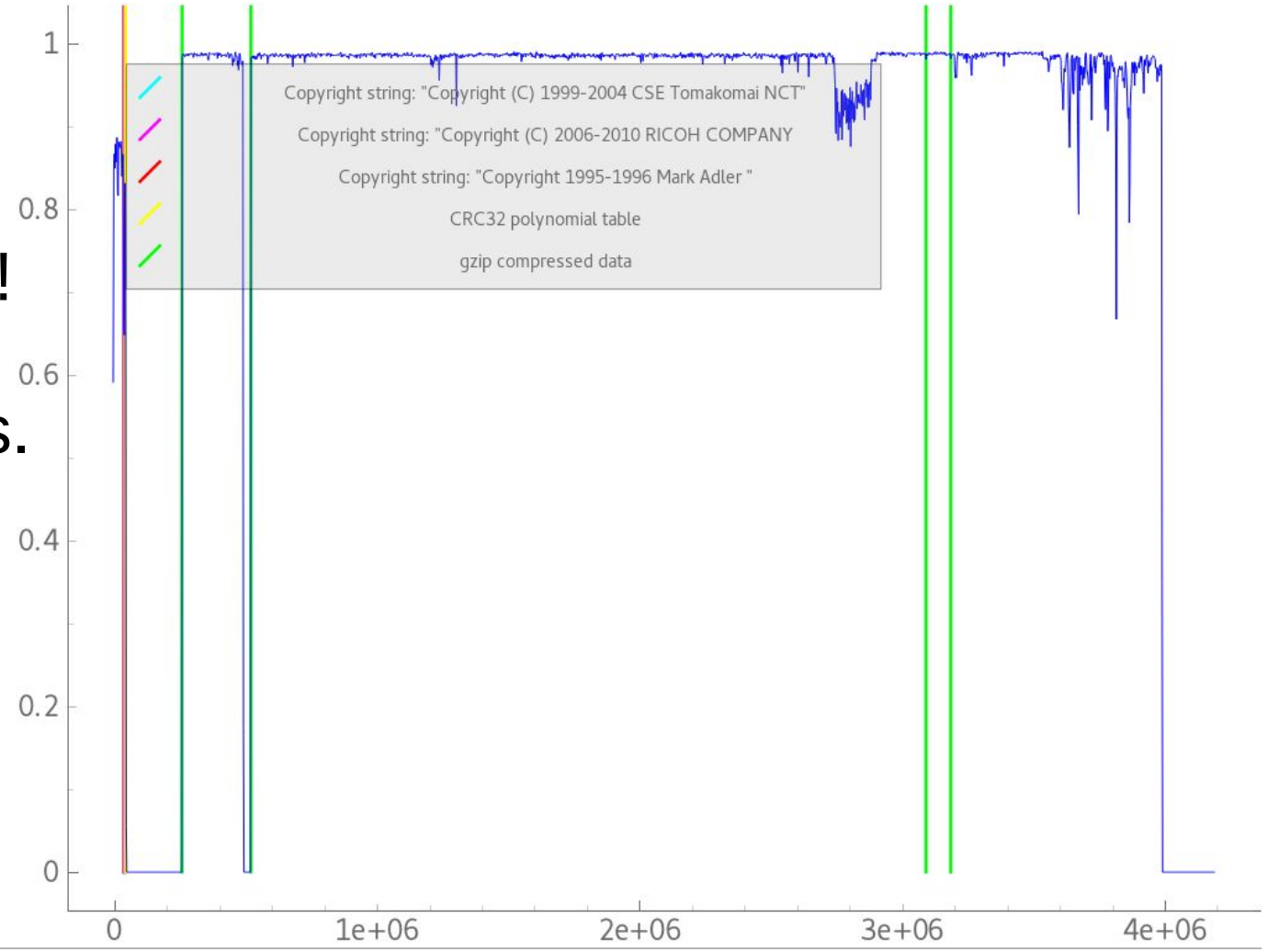
Analysis with examples

- binwalk to the rescue!
- Stage 1:
  We cut out and extract the first gzip archive.



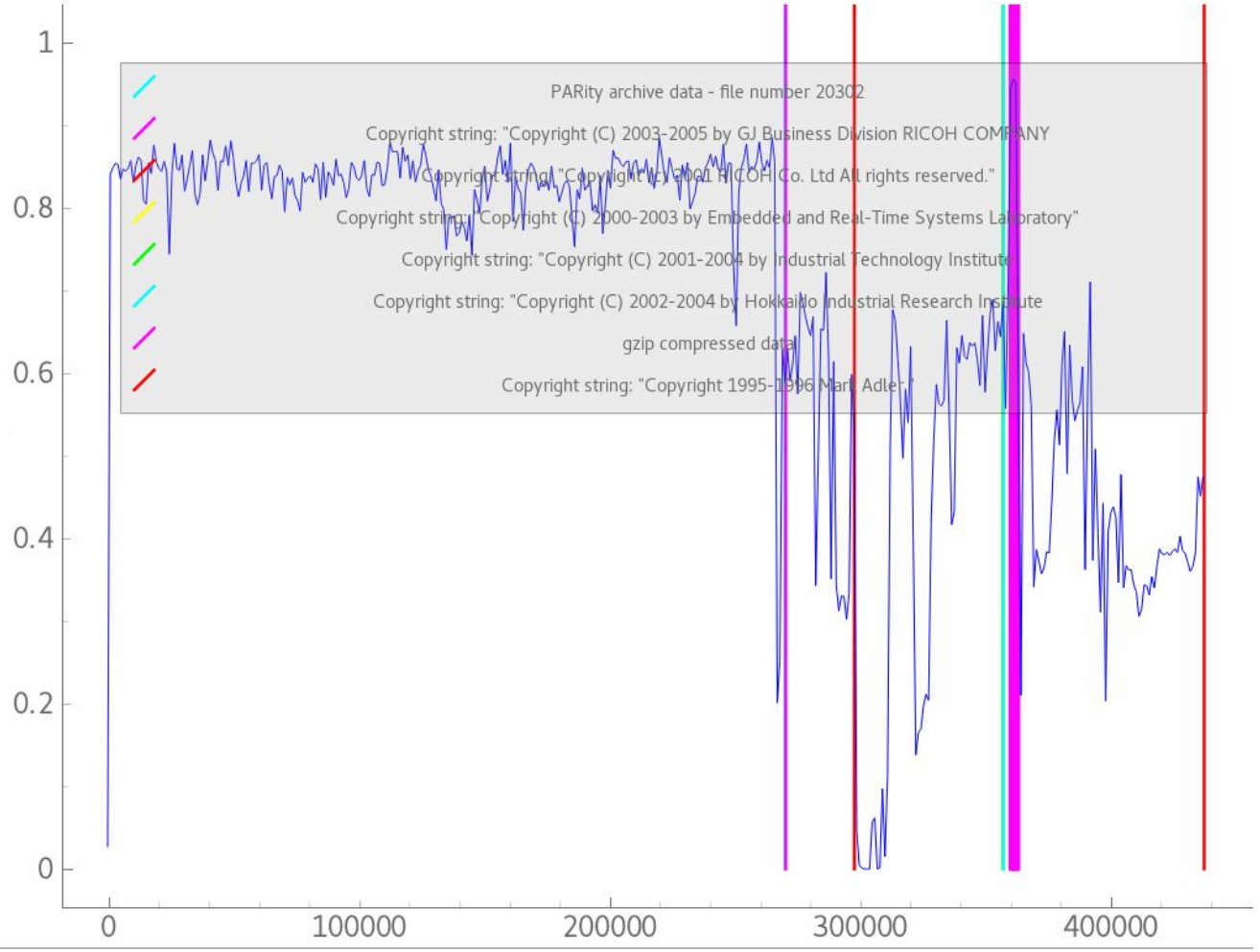
Analysis with examples

- binwalk to the rescue!
- Stage 2:
  But wait - there's more!
- We cut out and extract two more gzip archives.



Analysis with examples

- binwalk to the rescue!
- Stage 3:
  The first file appears to contain code related to the update process, and the second contains the printer's web service interface and "operating system" files, again packaged in a gzip archive.



Analysis with examples

- The second file contained what we were looking for.
- Now a tedious analysis could be performed, combined with analyzing the printer's PCB and trying to use the debugging port to pull out the information we were looking for.
- On the other hand, why not try a classic solution? Linux includes the *strings* tool, which could come in handy here.

Analysis with examples

- And indeed, using *strings* was enough to pull out the names of the data fields embedded in the firmware.
- Unfortunately, while this was a quick and useful solution, it does not provide us with an important piece of information - what are the addresses of these fields in the chip's memory?
- However, since this chip was the same as the previous one, some of the fields matched, some could be easily determined by analyzing the data as in previous cases and for some the information obtained from the firmware was enough.

Analysis with examples

DRM 4

Analysis with examples

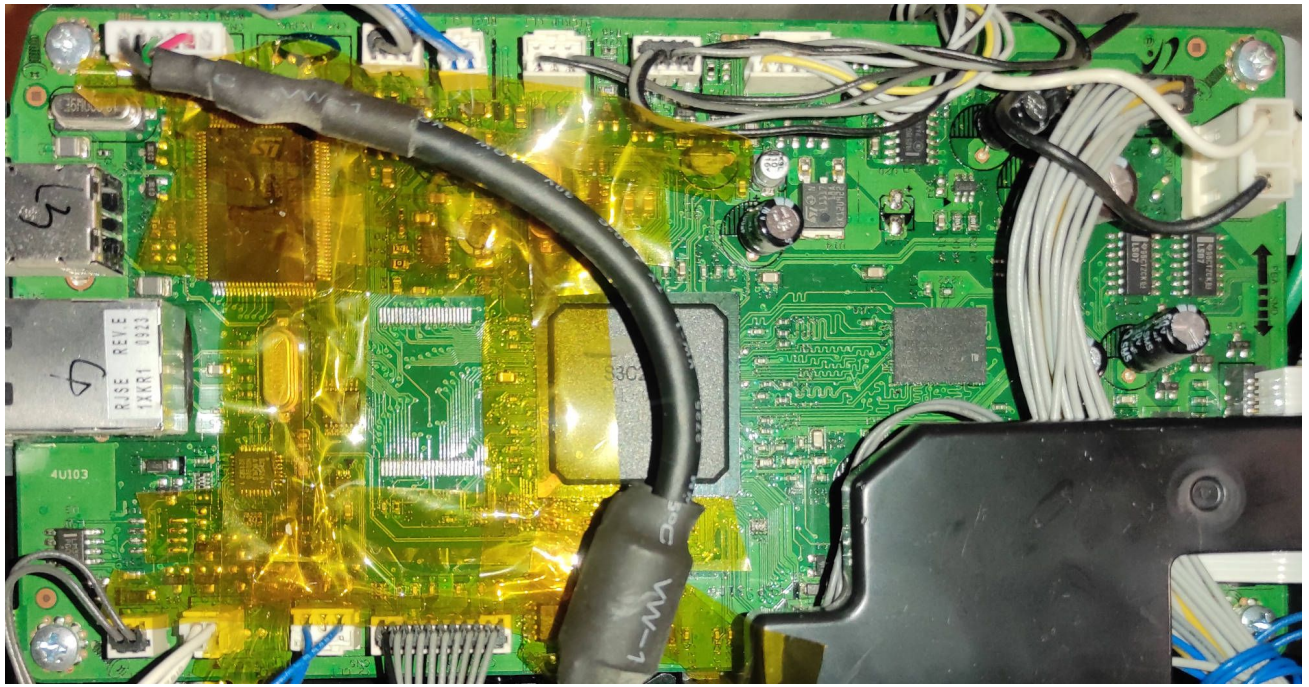
- But what happens when we come across an ASIC transmitting encrypted data via a standard interface (such as I2C)?
- *Oh boy, this is where the fun begins!*
- What we know:
  - There is some sort of integrated circuit. It may be a standard one with changed markings or without markings, but we don't know that.
  - The communication interface is known (I2C) and we are able, as before, to capture the data.
  - The captured data looks random.

Analysis with examples

- Since we don't have access to tools that would allow us to perform invasive IC analysis (not to mention the time and money required), we are left with firmware analysis.
- However, no need to worry - it won't be *that* bad! Well, maybe a little.
- At the very beginning we run into the first problem - our printer is no longer supported by the manufacturer, so there are no firmware updates to download for it.
- If we can't take the easy route, we need to go on an adventure!

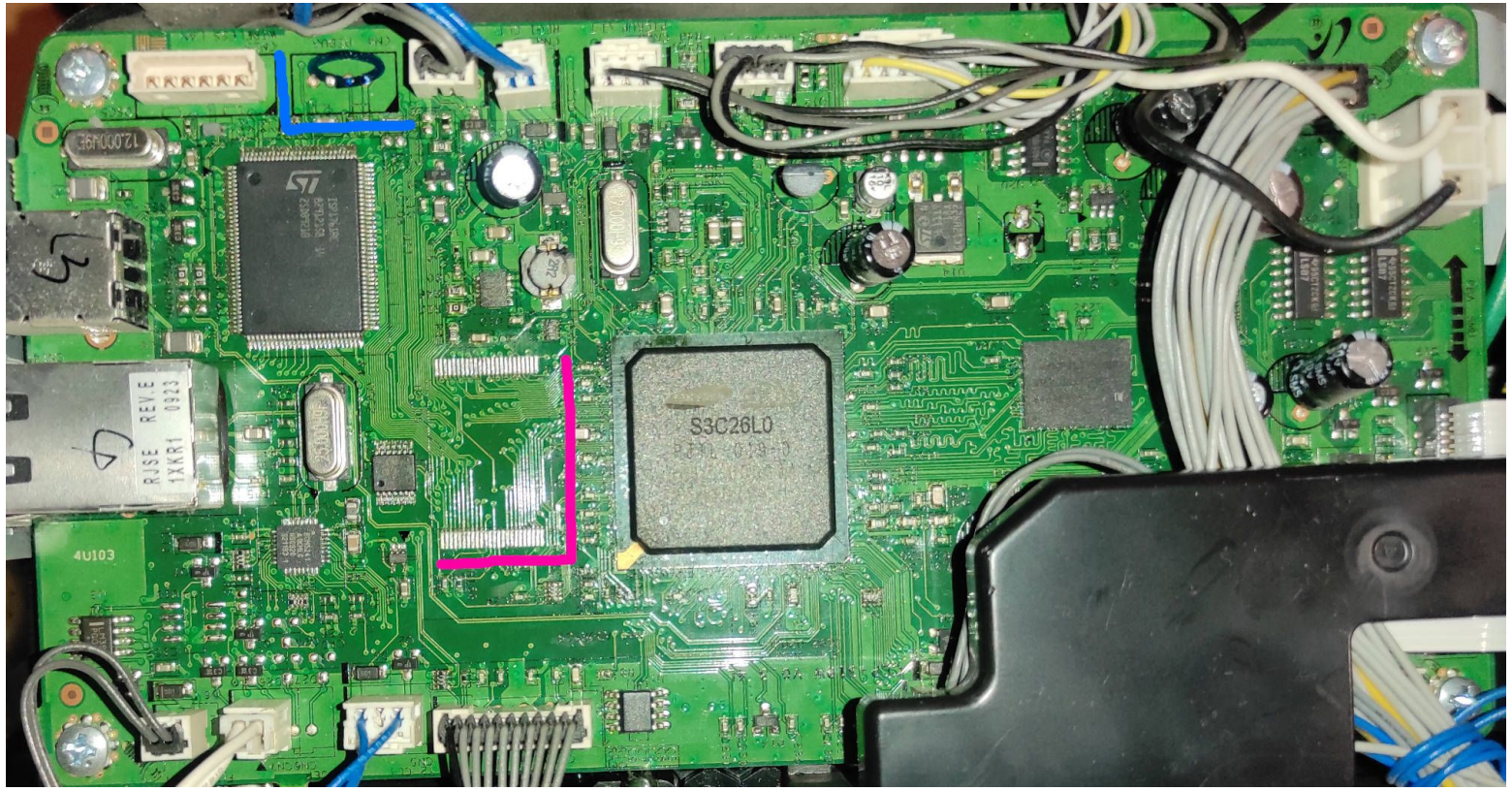
Analysis with examples

- To obtain the firmware, we need to read it from the printer's flash memory. To do this, we desolder the memory with hot air and read it with a programmer.



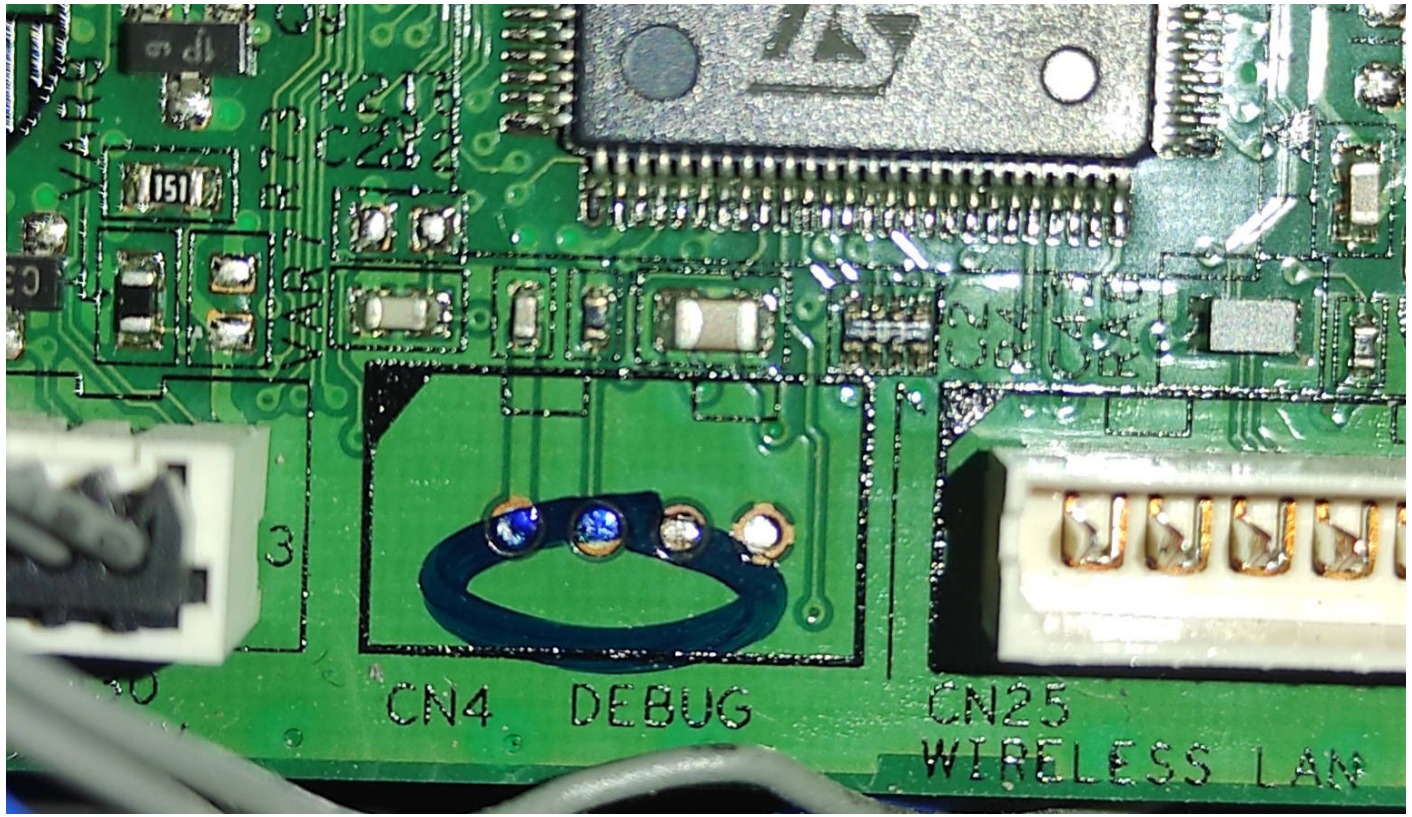
Analysis with examples

- However, should we always do so? Well, we don't have to. But it's too late for that.



Analysis with examples

- And what do we have here?



Analysis with examples

- Now that we've dumped the firmware, let's take a peek at what's inside.

Analysis with examples

- The analysis shows that there are three areas of high entropy (probably compressed data), and we have a fragment at the beginning that may be a starting point for us in the analysis.
- However, is it really? Let's check it out!

Analysis with examples

- *What does strings say?*

```
< <<<<<<<<<<OBTOH AEET RNO>>>>>>>>>>>>>>>
< <<<<<<<<<<OBTOH AEET RFOFFFF>F>>>>>>>>>>>>>>>
ujpmt  ox0x%
oR mhCceSkmuv laeui  savil dx0x%
oR mhCceSkmuv laeui  son tavil dx0x%
rPsees dek.y..
oDnwoldaM do e  rPse stSpoK ye
kSpiD wolnao
rPse sna yek yote exucetM notiroP orrgmaw tiih n 1es.c..
D RD>1
D RD>2
Z$PI
F ni deKnrle
K reen lnUiZ poDen!!
```

Analysis with examples

- It appears that the byte order is swapped in pairs.
- This time a simple Python script will come in handy:

```
def swap_bytes(input_file_path, output_file_path):
    with open(input_file_path, 'rb') as input_file:
        original_bytes = input_file.read()

    swapped_bytes = bytearray()
    for i in range(0, len(original_bytes), 2):
        if i + 1 < len(original_bytes):
            swapped_bytes.append(original_bytes[i + 1])
            swapped_bytes.append(original_bytes[i])

    with open(output_file_path, 'wb') as output_file:
        output_file.write(swapped_bytes)

swap_bytes("source.bin", "swapped.bin")
```

Analysis with examples

- *What does strings say?*

```
 <<<<<<<<<<<<BOOT HEATER ON>>>>>>>>>>>>>>>>
 <<<<<<<<<<<<BOOT HEATER OFFFFFF>>>>>>>>>>>>>>>>
jump to 0x%x
Rom CheckSum value is valid 0x%x
Rom CheckSum value is not valid 0x%x
Pressed key...
Download Mode   Press Stop Key
Skip Download
Press any key to execute Monitor Program within 1 sec...
 DDR1>
 DDR2>
$ZIP
 Find Kernel
 Kernel UnZip Done!!
```

Analysis with examples

- Now we can start analyzing part of the extracted firmware.
- However, before we do that, let's connect to the console. This will allow us to find clues that will be useful in later analysis.

```
------------------------------------------------------------
START-UP MODE : Monitor Program
  Boot into pROBE+ like stand-alone mode
  [Type 'help' to see command info.]
  Version : B0.236 05-07-2008 (Millet WLAN No use IFS)
------------------------------------------------------------
Do you want to download from external port?[N] : N

pROBE+> ?
```

Analysis with examples

```
-----ROM monitor command format-----------
dm   start_addr <byte_count>
dm.b start_addr <byte_count>
dm.w start_addr <byte_count>
dm.l start_addr <byte_count>
ESC  repeat memory dump
fm   start_addr  byte_count byte_value
fm.b start_addr  byte_count byte_value
fm.w start_addr  word_count word_value
fm.l start_addr  long_count long_value
pm   addr        byte_value
pm.b addr        byte_value
pm.w addr        word_value
pm.l addr        long_value
ul   byte_count  : upload image
cmp  src1_addr src2_addr size : compare memory

ml    : Automatically download and execute the ram area binary
fl    : upgrade flash image
gz.s start_area : gunzip image at target area for DDR test
gz.a  : Automatically gunzip image at all ram area for DDR test
go   jmp_addr
dl   load_addr
```

Analysis with examples

```
--------------
ddr cmd arg1 arg2 arg3
      read size (0 :64M,1:128M,2:256M,3:512M(All memory)
      write size (0 :64M,1:128M,2:256M,3:512M(All memory)
      wrc   size dimm
            -size    -> 0 :64M,1:128M,2:256M,3:512M(each memory)
            -dimm -> 0 :dimm0,1:dimm1
      wid   size dimm
            -size    -> 0 :64M,1:128M,2:256M,3:512M(each memory)
            -dimm -> 0 :dimm0,1:dimm1
      read   size dimm
            -size    -> 0 :64M,1:128M,2:256M,3:512M(each memory)
            -dimm -> 0 :dimm0,1:dimm1
      write   size dimm
            -size    -> 0 :64M,1:128M,2:256M,3:512M(each memory)
            -dimm -> 0 :dimm0,1:dimm1
```

Analysis with examples

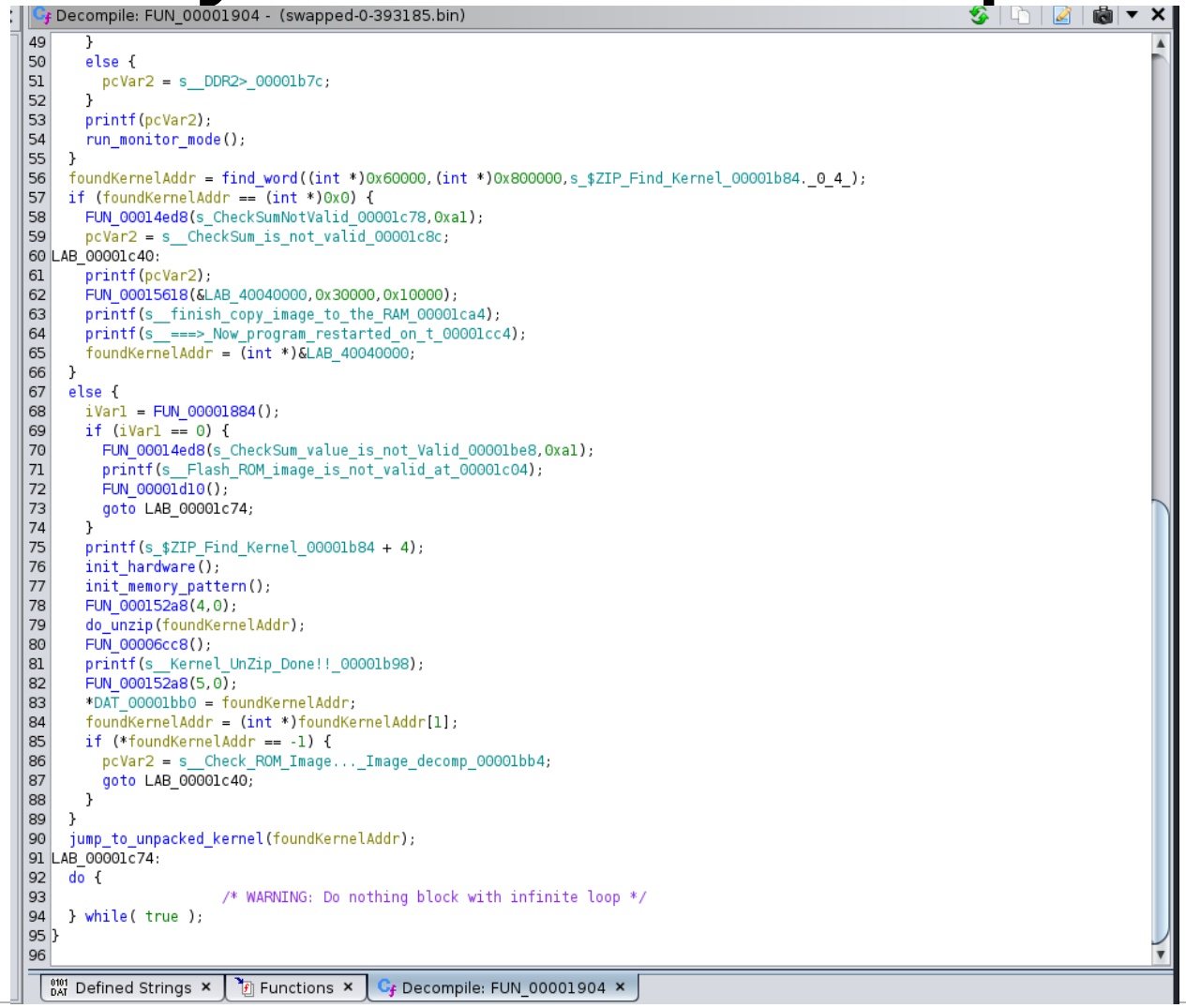
```
pROBE+> gz.a

 Start Address is 0x42000000
 Enable MMU
 Enable DCache

 Find Kernel
ZIP IMG ROM Start Address : 0x00060000
 Try Num : 1.....................................*
 Zip End

================================================
 Start Address is 0x42300000
ZIP IMG ROM Start Address : 0x00060000
 Try Num : 2.....................................*
 Zip End

pROBE+> dm 60000 10
00060000   24 5a 49 50 40 14 00 00  00 24 00 00 00 15 2e 1c    $ZIP@....$......

pROBE+> ml

Ready to download from Parallel/Usb/Serial (0x40140000)
[ISP1761] Cable is not connected
jump to 0x40140000
```

Analysis with examples

- So we open the firmware in Ghidra, start looking for clues and find...

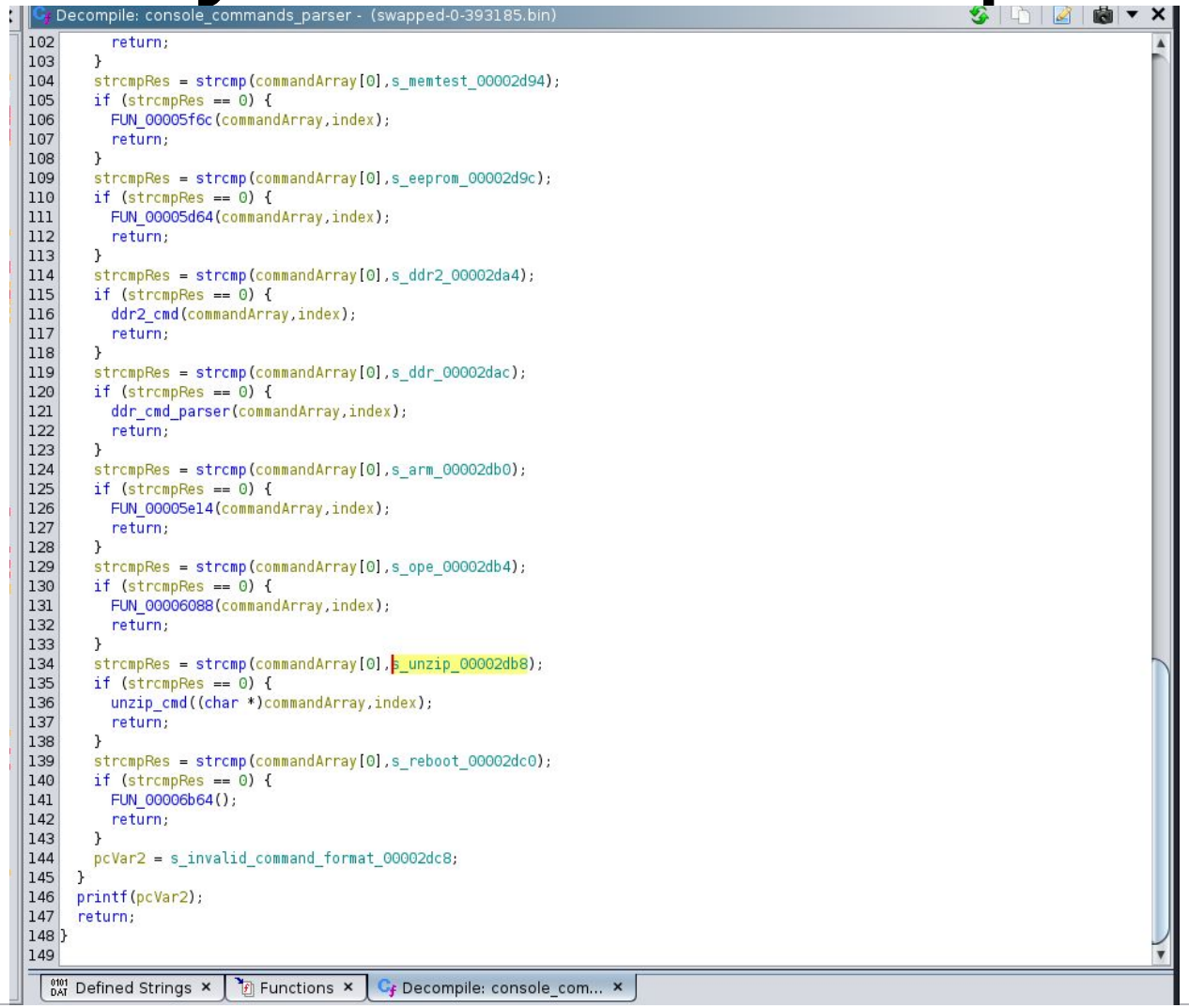
Analysis with examples

- Initialization code



Analysis with examples

- And undocumented features!



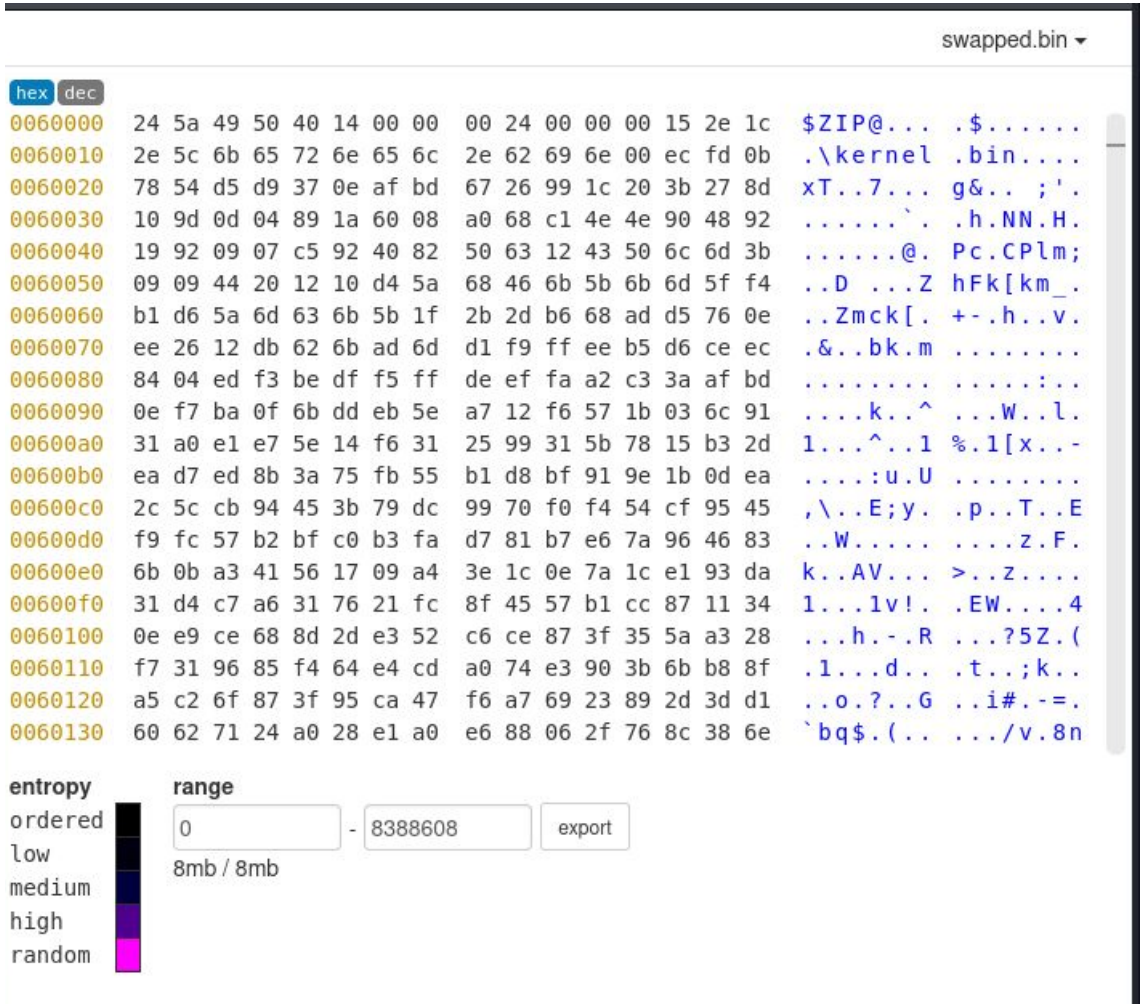

```
102       return;
103     }
104     strcmpRes = strcmp(commandArray[0],s_memtest_00002d94);
105     if (strcmpRes == 0) {
106       FUN_00005f6c(commandArray,index);
107       return;
108     }
109     strcmpRes = strcmp(commandArray[0],s_eeprom_00002d9c);
110     if (strcmpRes == 0) {
111       FUN_00005d64(commandArray,index);
112       return;
113     }
114     strcmpRes = strcmp(commandArray[0],s_ddr2_00002da4);
115     if (strcmpRes == 0) {
116       ddr2_cmd(commandArray,index);
117       return;
118     }
119     strcmpRes = strcmp(commandArray[0],s_ddr_00002dac);
120     if (strcmpRes == 0) {
121       ddr_cmd_parser(commandArray,index);
122       return;
123     }
124     strcmpRes = strcmp(commandArray[0],s_arm_00002db0);
125     if (strcmpRes == 0) {
126       FUN_00005e14(commandArray,index);
127       return;
128     }
129     strcmpRes = strcmp(commandArray[0],s_ope_00002db4);
130     if (strcmpRes == 0) {
131       FUN_00006088(commandArray,index);
132       return;
133     }
134     strcmpRes = strcmp(commandArray[0],s_unzip_00002db8);
135     if (strcmpRes == 0) {
136       unzip_cmd((char *)commandArray,index);
137       return;
138     }
139     strcmpRes = strcmp(commandArray[0],s_reboot_00002dc0);
140     if (strcmpRes == 0) {
141       FUN_00006b64();
142       return;
143     }
144     pcVar2 = s_invalid_command_format_00002dc8;
145   }
146   printf(pcVar2);
147   return;
148 }
149
```

Analysis with examples

- So if those areas of high entropy in memory are ZIP archives, why not just cut them out of the firmware and unzip them?

Analysis with examples

- *Seemingly it is a ZIP, but kind of not really.*



Analysis with examples

- What can we do now?
- We are left with putting something together from the tools available in the console.
- Let's check them out:
  undocumented features – not working
  reading from RAM – not working

Analysis with examples

- So is there nothing that can be done and you will have to manually analyze the compression algorithm used?
- *Fortunately, no.*
- It turns out that one of the commands (gz.s) can extract the packed image for us to a selected address in RAM.
- *Unfortunately, this is immediately followed by a printer reset.*
- But fortunately, the memory is not cleared at startup, so the unpacked image is still there!
- In addition, the command to read flash memory also accepts addresses in RAM, so we can read the contents of RAM with it.

Analysis with examples

```
pROBE+> dm.l 42000000 10
42000000   ffffffff ffffffff  ffffffff ffffffff    ................

pROBE+> gz.s 42000000

 Find Kernel
 Enable MMU
 Enable DCache
ZIP IMG ROM Start Address : 0x00060000
 Try Num : 1.......................................*
 Zip End
 Disable MMU
 Disable DCache

-------------------------------------------------------------
START-UP MODE : Monitor Program
  Boot into pROBE+ like stand-alone mode
  [Type 'help' to see command info.]
  Version : B0.236 05-07-2008 (Millet WLAN No use IFS)
-------------------------------------------------------------
Do you want to download from external port?[N] : n

pROBE+> dm 42000000 10
42000000   ea 06 8f 4e e5 9f 00 34  e5 9f 10 34 e5 9f 30 34    ...N...4...4..04
```

Analysis with examples

- Now we can dump the RAM, but we do not know how much data was in the unpacked archive.
- So we will download with the excess, just to be sure.
- The code analysis looks like the previous case, but here there is much more code.
- Theoretically, we could try to emulate this processor, but it is part of the SoC, moreover, we will not be able to emulate what we are interested in, that is, chips with DRM.
- What to do in this situation?
- *strings! strings! strings!*

Analysis with examples

- With keywords such as "crum" and "crypto" we find what we were looking for – "S3CC921". This is the designation of the DRM chip used.
- However, there are no functions in sight that look like they are related to "encryption," which most often means "a lot of XOR."
- So let's see what they write on the Internet.
- It turns out that someone has already made a program to read and write these chips.

Analysis with examples

Decompile: FUN_421c38ec - (memdmp_kernel_0x42000000.bin)

```
1
2  void FUN_421c38ec(uint *param_1,int param_2,int param_3)
3
4  {
5    uint uVar1;
6    int iVar2;
7    uint uVar3;
8    uint uVar4;
9    uint uVar5;
10   uint uVar6;
11   int iVar7;
12
13   uVar5 = (*param_1 ^ param_1[1] >> 4) & 0xf0f0f0f;
14   uVar1 = *param_1 ^ uVar5;
15   uVar6 = param_1[1] ^ uVar5 << 4;
16   uVar5 = (uVar6 ^ uVar1 >> 0x10) & 0xffff;
17   uVar6 = uVar6 ^ uVar5;
18   uVar1 = uVar1 ^ uVar5 << 0x10;
19   uVar5 = (uVar1 ^ uVar6 >> 2) & 0x33333333;
20   uVar1 = uVar1 ^ uVar5;
21   uVar6 = uVar6 ^ uVar5 << 2;
22   uVar5 = (uVar6 ^ uVar1 >> 8) & 0xff00ff;
23   uVar6 = uVar6 ^ uVar5;
24   uVar1 = uVar1 ^ uVar5 << 8;
25   uVar5 = (uVar1 ^ uVar6 >> 1) & 0x55555555;
26   uVar1 = uVar1 ^ uVar5;
27   uVar6 = uVar6 ^ uVar5 << 1;
28   uVar5 = uVar6 >> 0x1d | uVar6 << 3;
29   uVar1 = uVar1 >> 0x1d | uVar1 << 3;
30   if (param_3 == 0) {
31     iVar2 = 0x1e;
32     do {
33       uVar4 = *(uint *)(param_2 + iVar2 * 4) ^ uVar1;
34       iVar7 = param_2 + iVar2 * 4;
35       uVar3 = *(uint *)(iVar7 + 4) ^ uVar1;
36       uVar6 = uVar3 >> 4;
37       iVar2 = iVar2 + -8;
38       uVar5 = *(uint *)(DAT_421c3ec4 + (uVar4 >> 2 & 0x3f) * 4) ^
39               *(uint *)(DAT_421c3ec4 + (uVar4 >> 10 & 0x3f) * 4 + 0x200) ^
40               *(uint *)(DAT_421c3ec4 + (uVar4 >> 0x12 & 0x3f) * 4 + 0x400) ^
41               *(uint *)(DAT_421c3ec4 + (uVar4 >> 0x1a) * 4 + 0x600) ^
42               *(uint *)(DAT_421c3ec4 + (uVar6 & 0xfc) + 0x100) ^
43               *(uint *)(DAT_421c3ec4 + ((uVar6 & 0xfc00) >> 10) * 4 + 0x300) ^
44               *(uint *)(DAT_421c3ec4 + ((uVar6 & 0xfc0000) >> 0x12) * 4 + 0x500) ^
45               *(uint *)(DAT_421c3ec4 + ((uVar6 | uVar3 << 0x1c) >> 0x1a) * 4 + 0x700) ^ uVar5;
46       uVar3 = *(uint *)(iVar7 + -8) ^ uVar5;
47       uVar4 = *(uint *)(iVar7 + -4) ^ uVar5;
48       uVar6 = uVar4 >> 4;
```

Defined Strings × | Functions × | Decompile: FUN_421c38ec ×

421c38ec | FUN_421c38ec

Decompile: crypto2 - (ps3cc921.exe)

```
1
2  void crypto2(uint param_1)
3
4  {
5    uint *puVar1;
6    uint uVar2;
7    uint uVar3;
8    int unaff_EBX;
9    uint uVar4;
10   uint *unaff_ESI;
11   uint uVar5;
12   uint local_28;
13
14   uVar2 = (unaff_ESI[1] >> 4 ^ *unaff_ESI) & 0xf0f0f0f;
15   uVar4 = *unaff_ESI ^ uVar2;
16   uVar2 = uVar2 << 4 ^ unaff_ESI[1];
17   uVar3 = (uVar4 >> 0x10 ^ uVar2) & 0xffff;
18   uVar2 = uVar2 ^ uVar3;
19   uVar4 = uVar4 ^ uVar3 << 0x10;
20   uVar3 = (uVar2 >> 2 ^ uVar4) & 0x33333333;
21   uVar4 = uVar4 ^ uVar3;
22   uVar2 = uVar2 ^ uVar3 << 2;
23   uVar3 = (uVar4 >> 8 ^ uVar2) & 0xff00ff;
24   uVar2 = uVar2 ^ uVar3;
25   uVar4 = uVar3 << 8 ^ uVar4;
26   uVar3 = (uVar2 >> 1 ^ uVar4) & 0x55555555;
27   uVar4 = uVar4 ^ uVar3;
28   uVar2 = uVar3 << 1 ^ uVar2;
29   uVar3 = uVar2 >> 0x1d | uVar2 << 3;
30   uVar2 = uVar4 >> 0x1d | uVar4 << 3;
31   if ((param_1 & 0xf) == 0) {
32     local_28 = 0x1e;
33     do {
34       puVar1 = (uint *)(local_28 * 4 + unaff_EBX);
35       uVar5 = *puVar1 ^ uVar2;
36       uVar4 = (puVar1[1] ^ uVar2) >> 4;
37       uVar3 = uVar3 ^ *(uint *)(&DAT_0040784c + ((uVar4 | (puVar1[1] ^ uVar2) << 0x1c) >> 0x1a) * 4)
38                     ^ *(uint *)(&DAT_0040744c + ((uVar4 & 0xfc00) >> 8)) ^
39                       *(uint *)(&DAT_0040724c + (uVar4 & 0xfc)) ^
40                       *(uint *)(&DAT_0040774c + (uVar5 >> 0x1a) * 4) ^
41                       *(uint *)(&DAT_0040714c + (uVar5 & 0xfc)) ^
42                       *(uint *)(&DAT_0040734c + (uVar5 >> 8 & 0xfc)) ^
43                       *(uint *)(&DAT_0040754c + (uVar5 >> 0x10 & 0xfc)) ^
44                       *(uint *)(&DAT_0040764c + ((uVar4 & 0xfc0000) >> 0x10));
45       uVar5 = puVar1[-2] ^ uVar3;
46       uVar4 = (puVar1[-1] ^ uVar3) >> 4;
47       uVar2 = uVar2 ^ *(uint *)(&DAT_0040764c + ((uVar4 & 0xfc0000) >> 0x10)) ^
48                       *(uint *)(&DAT_0040744c + ((uVar4 & 0xfc00) >> 8)) ^
```

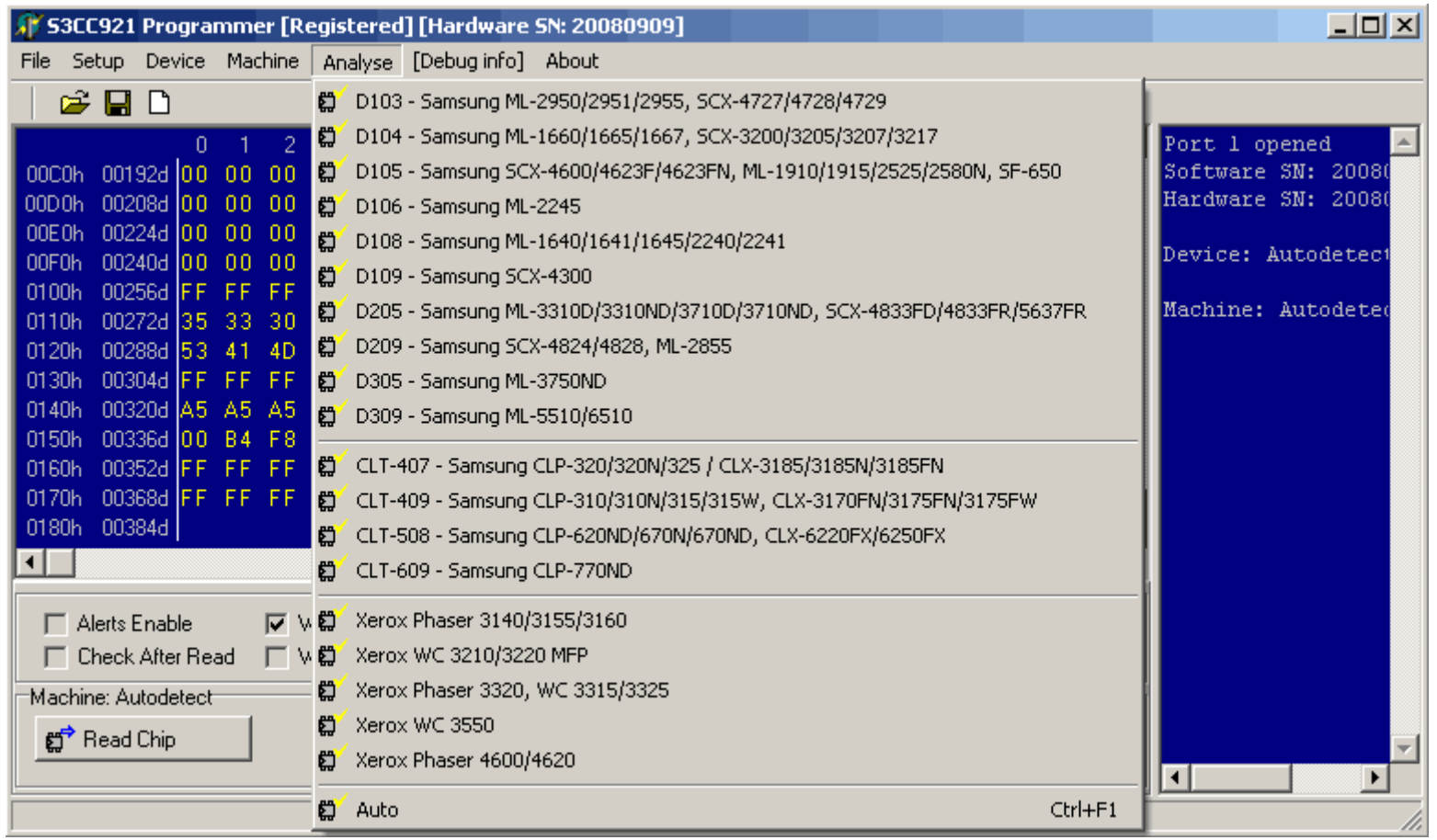
Defined Strings × | Functions × | Decompile: crypto2 ×

0040483e | crypto2 | PUSH EBX

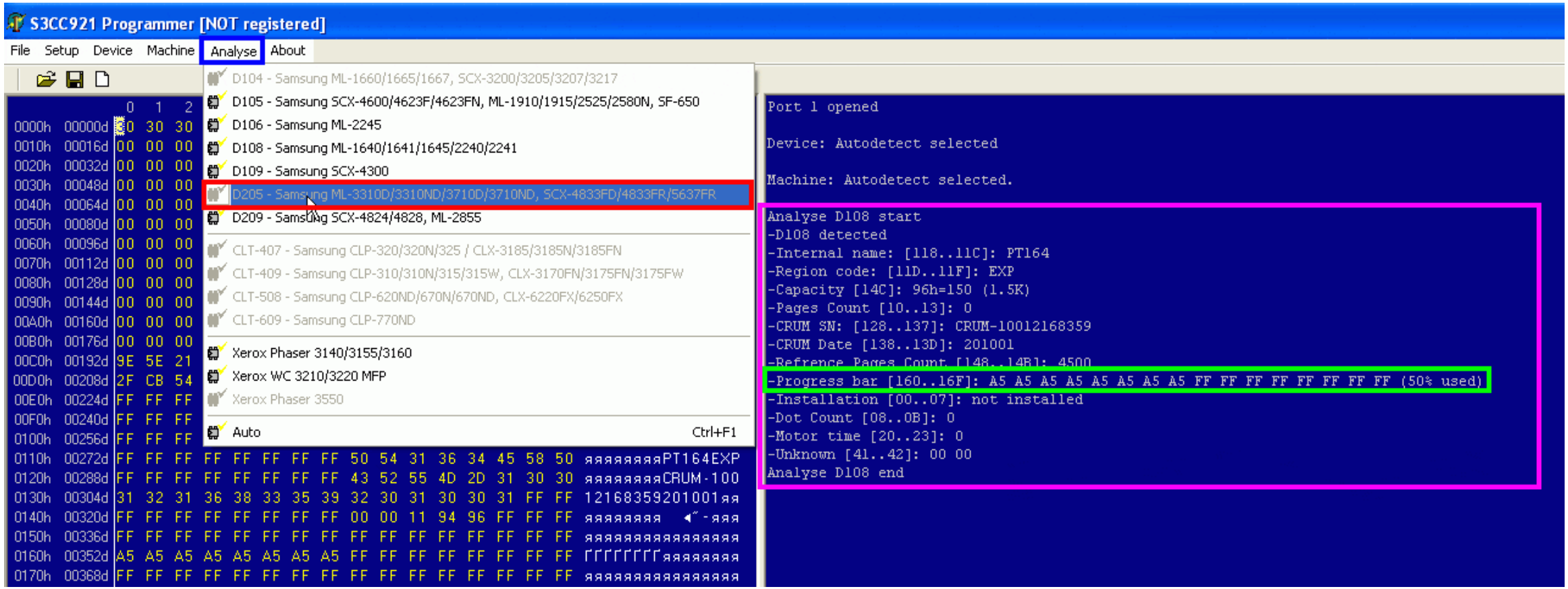
Analysis with examples

- Its code even looks quite similar to the one found in the printer's firmware.
- This will make it easier to analyze how this DRM works.
- Overall, it can be said that this DRM relies more on "security by obscurity" (and XORs) than on standard cryptographic algorithms.
- In addition, on the Internet you can find interesting sources and communities (mostly Russian) centered around creating printer firmware patches ("eternal chips") or chip reset tools.

Analysis with examples



Analysis with examples



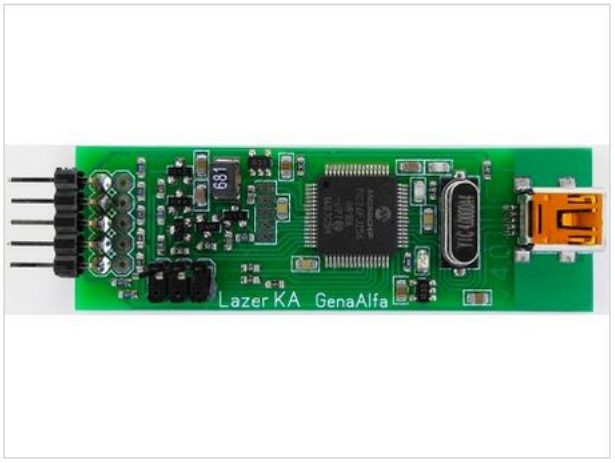
Analysis with examples

Resetkits

ENG 0.00 EUR

SEARCH PRODUCTS

PRODUCTS FILES NEWS ARTICLES QUESTIONS AND ANSWERS ABOUT US CONTACTS LOG IN



Programmer CrumProg

Resetkits

~~115.00~~ 100.00 EUR

Quantity: 1 Add to cart

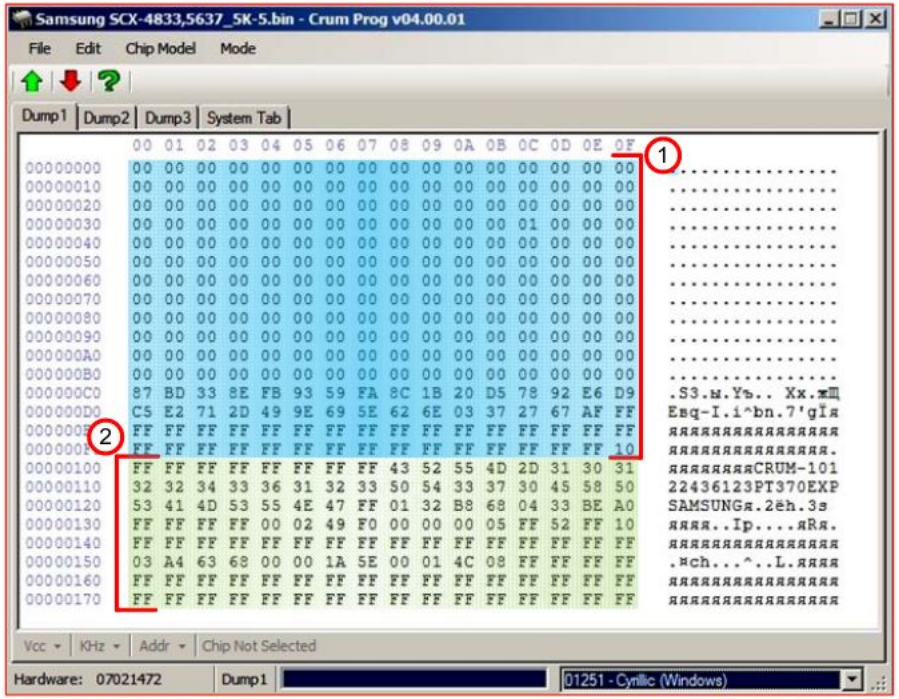
In stock

DESCRIPTION

The programmer of the memory chips is oriented to work with a memory of chips installed in the consumable printer cartridges.
The programmer is connected and is powered via USB 2.0 / 1.1 connection and has no additional external power supply. The programmer works with standard USB drivers, HID class, which are available in each system. Additional driver installation is not required.
The quantity of programmable chips is unlimited and has no restrictions. A payment is made only one time - at the time of the purchase, and the programmer does not require additional costs for the further using.
The programmer is equipped with a bootloader that allows to update the firmware of the programmer when an update is released. Updates are placed on our FTP server in a free download and do not require additional payment.
The developers are constantly working on increasing a list of supported chips and on improving the functionality of the software of the programmer. A frequency of issuing the updates can be seen in a document History.rtf, which is found in the software package.
The programmer is equipped with several power supplies, which cover a range of supply voltages from 3.3 to 20 volts required for a different types of memory chips.
A software and hardware output protection of the Programmer is implemented. It minimizes the chance of

Analysis with examples

S3CC912, S3CC921



Chip address space is divided into two zones - EEPROM (1) and OTP (2). **EEPROM** zone is located at the addresses from 00h to FFh. In the original chips this area can be changed as you like, and in this zone there are main counters that must be reset. Multiple reading and writing of data is possible. It should be noted that the memory area from C0 to FF still has not been used in printers, and it contains different data in different chips, so we advise to leave this zone unchanged. Perhaps in this zone there are the keys or data that will be used by next versions of printer firmware.

Analysis with examples

- Tools:
  https://www.saleae.com/
  https://www.dreamsourcelab.com/

  https://binvis.io
  https://ghidra-sre.org/

Analysis with examples

- Links:

OCEŃ PRELEKCJĘ

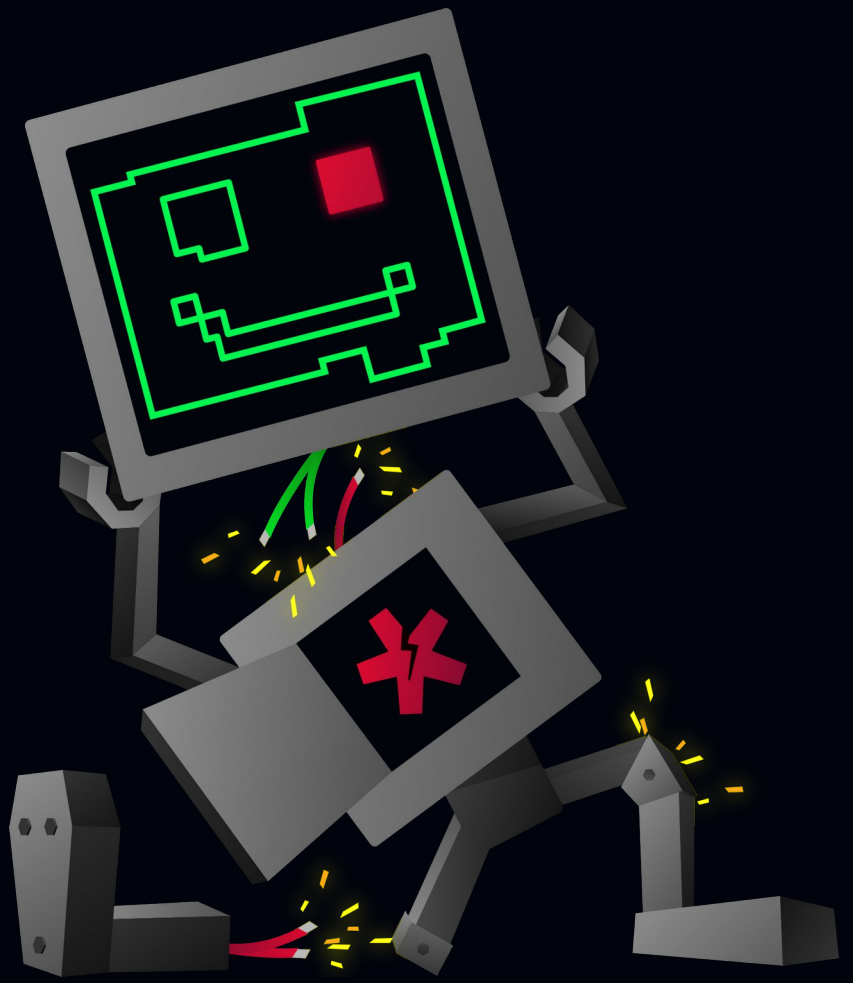
Unveiling the Secrets of Hardware DRM in Printers

Wojciech Cybowski

https://thehacksummit.com/user.html#!/lecture/THS23-0996/rate

That’s all folks!

THE H@CK SUMMIT



Dziękuję za uwagę!

Zapraszam do zadawania pytań oraz oceny wystąpienia pod nagraniem.

thehacksummit.com | 19-20 października 2023 | PGE Narodowy + Online

ORGANIZATORZY:

ACADEMIC PARTNERS

DOWÓDZTWO KOMPONENTU WOJSK OBRONY CYBERPRZESTRZENI