BLACK ALPS

The inner guts of a connected glucose sensor for diabetes

Axelle Apvrille (Fortinet)
Travis Goodspeed

November 2019

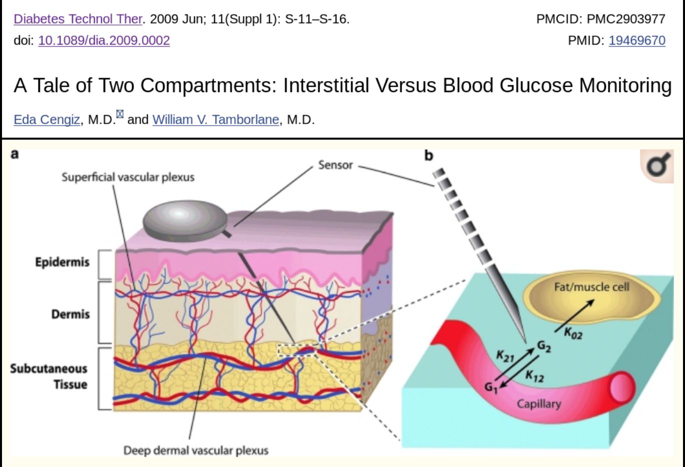
1. Introduction
    - Who are we?
    - Medical background
2. Hardware
    - Teardown
    - Read The Datasheet
3. NFC
    - FRAM: user data
    - FRAM: code and tables
    - Dumping the firmware: Raw Read
    - Lock/Unlock vuln
4. Sensor expiration date: how does it work?
5. Conclusion

Who are we?

Axelle Apvrille
Principal Security Researcher at **Fortinet**, @cryptax
Mobile malware, IoT, Ph0wn CTF

Travis Goodspeed
Digital watchmaker and Studebaker enthusiast, @travisgoodspeed
GoodFET, GoodWatch, PoC||GTFO

Flash Glucose Monitoring system



Diabetes Technol Ther. 2009 Jun; 11(Suppl 1): S-11–S-16. PMCID: PMC2903977

doi: 10.1089/dia.2009.0002 PMID: 19469670

A Tale of Two Compartments: Interstitial Versus Blood Glucose Monitoring

Eda Cengiz, M.D. and William V. Tamborlane, M.D.

@cryptax testing the sensor!

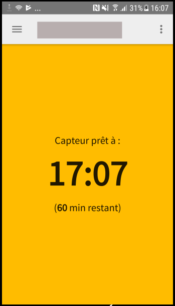
Screenshot from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2903977/

Sensor life cycle

Assemble pack

[redacted]
[redacted]

Apply sensor



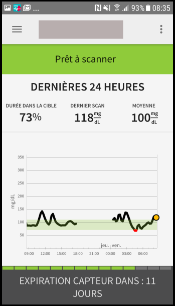
[redacted]

Capteur prêt à :

17:07

(60 min restant)

Activate it (60 min)



[redacted]

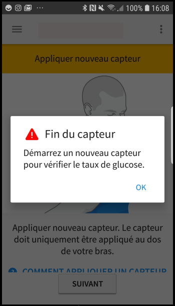
Prêt à scanner

DERNIÈRES 24 HEURES

DURÉE DANS LA CIBLE	DERNIER SCAN	MOYENNE
73%	118 mg/dL	100 mg/dL

EXPIRATION CAPTEUR DANS : 11 JOURS

Use it



[redacted]

Appliquer nouveau capteur

Fin du capteur

Démarrez un nouveau capteur pour vérifier le taux de glucose.

OK

Appliquer nouveau capteur. Le capteur doit uniquement être appliqué au dos de votre bras.

COMMENT APPLIQUER UN CAPTEUR

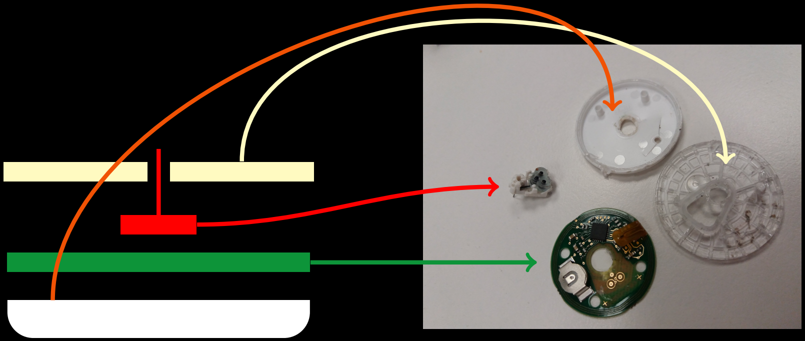
SUIVANT

Expires after 14 days

1. Introduction
   - Who are we?
   - Medical background
2. Hardware
   - Teardown
   - Read The Datasheet
3. NFC
   - FRAM: user data
   - FRAM: code and tables
   - Dumping the firmware: Raw Read
   - Lock/Unlock vuln
4. Sensor expiration date: how does it work?
5. Conclusion

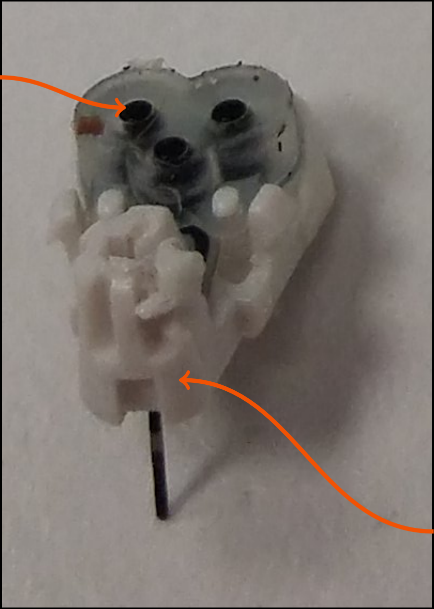
Sensor Teardown

Trick: unclip enzyme sensor part, then put a blade in the middle of the case



Other teardowns:
https://www.youtube.com/watch?v=40RXFhZp8hg
https://www.youtube.com/watch?v=sYIm97wjl0o

Enzyme sensor

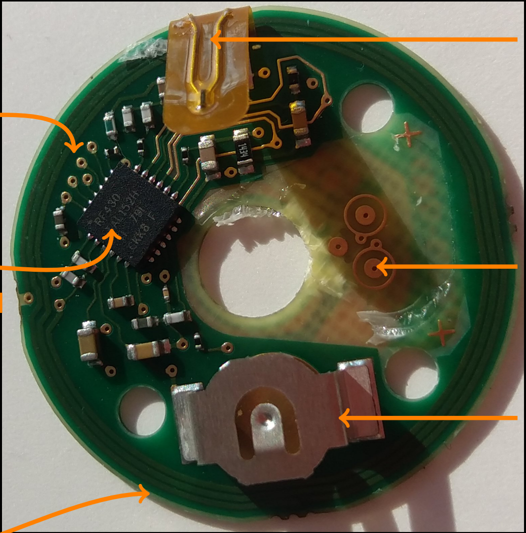


3 electrode contacts

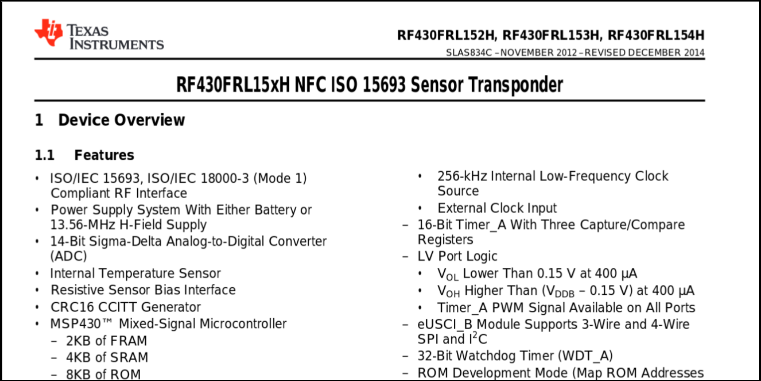
Covered with Glucose Oxydase (GOx)

5mm long, 0.4mm wide

PCB


Temperature sensor
JTAG
Texas Instruments RF430TAL152H
Enzyme sensor pins
Battery V337
NFC antenna

Datasheet



TEXAS INSTRUMENTS

RF430FRL152H, RF430FRL153H, RF430FRL154H

SLAS834C – NOVEMBER 2012 – REVISED DECEMBER 2014

RF430FRL15xH NFC ISO 15693 Sensor Transponder

1 Device Overview

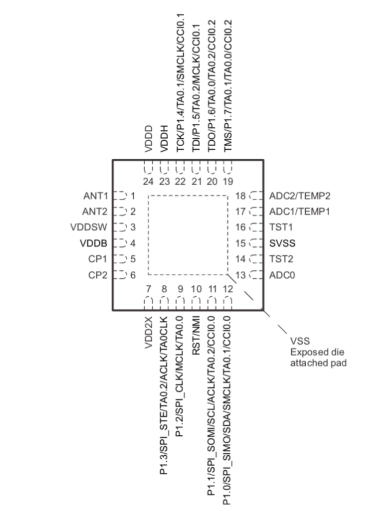
1.1 Features

- ISO/IEC 15693, ISO/IEC 18000-3 (Mode 1) Compliant RF Interface
- Power Supply System With Either Battery or 13.56-MHz H-Field Supply
- 14-Bit Sigma-Delta Analog-to-Digital Converter (ADC)
- Internal Temperature Sensor
- Resistive Sensor Bias Interface
- CRC16 CCITT Generator
- MSP430™ Mixed-Signal Microcontroller
  - 2KB of FRAM
  - 4KB of SRAM
  - 8KB of ROM
    - 256-kHz Internal Low-Frequency Clock Source
    - External Clock Input
  - 16-Bit Timer_A With Three Capture/Compare Registers
  - LV Port Logic
    - V_{OL} Lower Than 0.15 V at 400 µA
    - V_{OH} Higher Than (V_{DDB} – 0.15 V) at 400 µA
    - Timer_A PWM Signal Available on All Ports
  - eUSCI_B Module Supports 3-Wire and 4-Wire SPI and I^2C
  - 32-Bit Watchdog Timer (WDT_A)
  - ROM Development Mode (Map ROM Addresses

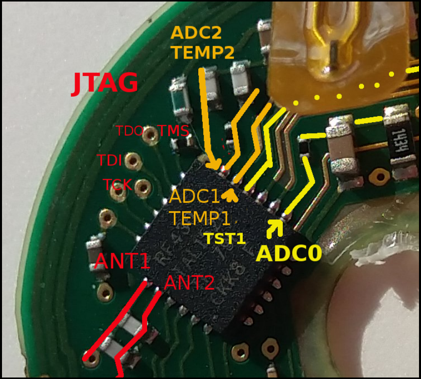
Screenshot of http://www.ti.com/lit/ds/symlink/rf430frl152h.pdf

- No public documentation for RF430 **TAL**, but *FRL*
- NFC ISO 15693 - *"Vicinity" cards*
- Uses **Ferroelectric** RAM (FRAM)

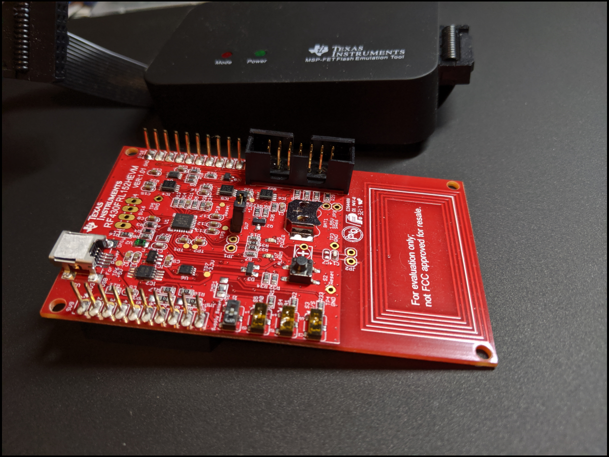
Pin assignment


VDDD
VDDH
TCK/P1.4/TA0.1/SMCLK/CCI0.1
TDI/P1.5/TA0.2/MCLK/CCI0.1
TDO/P1.6/TA0.0/TA0.2/CCI0.2
TMS/P1.7/TA0.1/TA0.0/CCI0.2
24 23 22 21 20 19
ANT1 1
ANT2 2
VDDSW 3
VDDB 4
CP1 5
CP2 6
18 ADC2/TEMP2
17 ADC1/TEMP1
16 TST1
15 SVSS
14 TST2
13 ADC0
7 8 9 10 11 12
VDD2X
P1.3/SPI_STE/TA0.2/ACLK/TA0CLK
P1.2/SPI_CLK/MCLK/TA0.0
RST/NMI
P1.1/SPI_SOMI/SCL/ACLK/TA0.2/CCI0.0
P1.0/SPI_SIMO/SDA/SMCLK/TA0.1/CCI0.0
VSS
Exposed die
attached pad

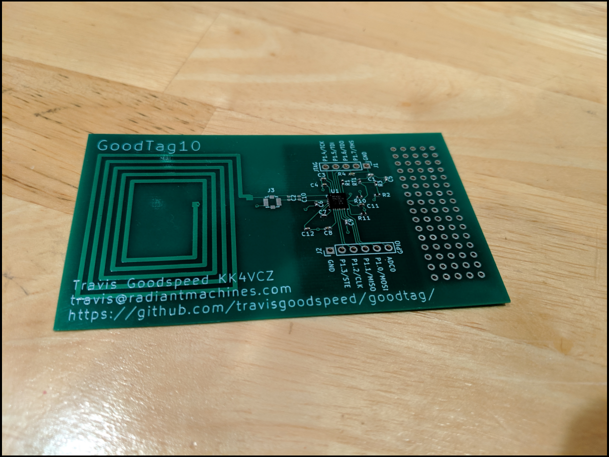
http://www.ti.com/lit/ds/symlink/rf430frl152h.pdf



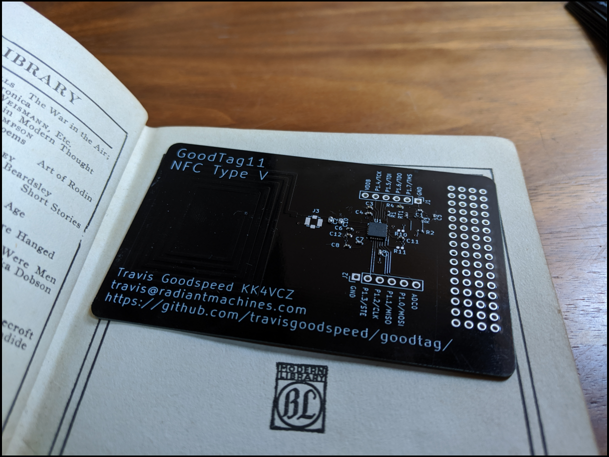
JTAG


Mode
Power
Texas Instruments
MSP-FET Flash Emulation Tool
Texas Instruments
RF430FRL152HEVM
For evaluation only;
not FCC approved for resale.

Custom Carrier Board

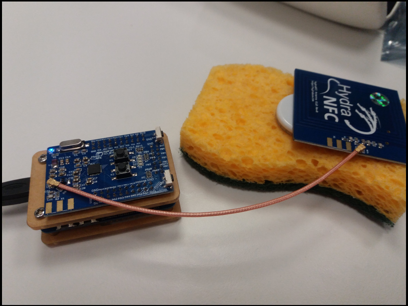


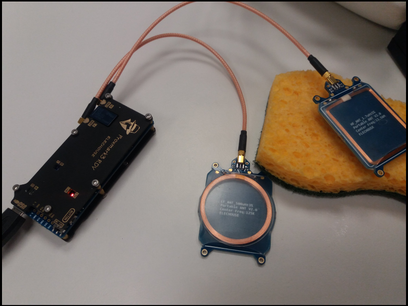
Custom Carrier Board



1. Introduction
   - Who are we?
   - Medical background
2. Hardware
   - Teardown
   - Read The Datasheet
3. NFC
   - FRAM: user data
   - FRAM: code and tables
   - Dumping the firmware: Raw Read
   - Lock/Unlock vuln
4. Sensor expiration date: how does it work?
5. Conclusion

NFC Reader





NXP PN 532, ST M24SR support ISO 14443, but **not 15693**.

Supported standard NFC commands

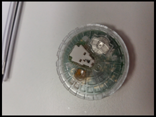
Command	Example
Get Inventory	26 **01** 00
Read Single Block	02 **20** BlockIndex
Write Single Block	42 **21** BlockIndex 8-byteData
Read Multiple Blocks (max 3)	02 **23** BlockIndex Number
Get System Info	02 **2B**

Reading NFC blocks

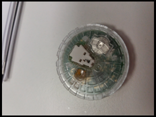
Dump memory

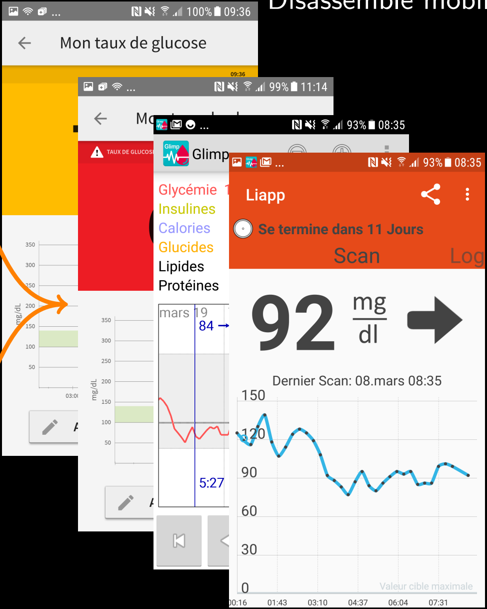
```
proxmark3> hf 15 dumpmemory
Reading memory from tag UID=E007A00003183AD2
Tag Info: Texas Instrument France
Block 00  75 B5 B0 12 01 00 00 00
Block 01  00 00 00 00 00 00 00 00
Block 02  00 00 00 00 00 00 00 00
Block 03  62 C2 00 00 00 00 00 00
Block 04  00 00 00 00 00 00 00 00
```

Understanding the memory layout



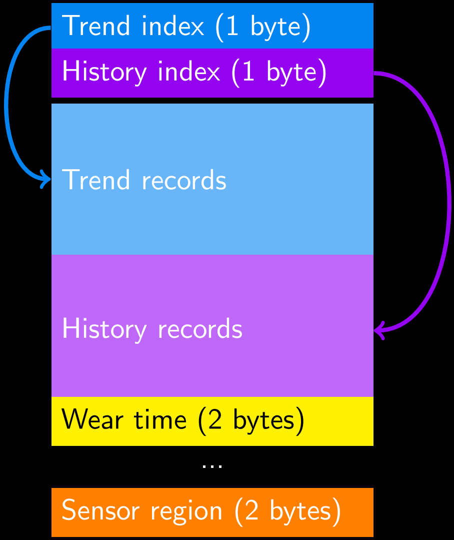
Numerous tests





Disassemble mobile apps

FRAM layout: user data

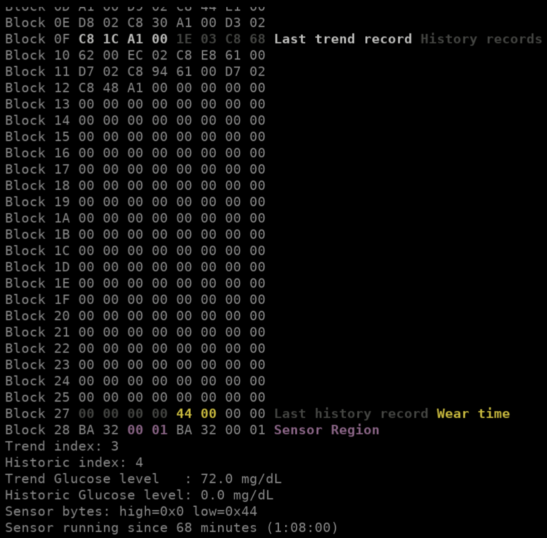


- 6-byte records
- 1 glucose measure per minute
- Wear time in minutes
- Region: 01 (Europe), 02 (US), 08 (Israel)...

Reading records

```
Trend record no. 0: 72.3 mg/dL
Trend record no. 1: 72.1 mg/dL
Trend record no. 2: 72.1 mg/dL
Trend record no. 3: 72.0 mg/dL
```

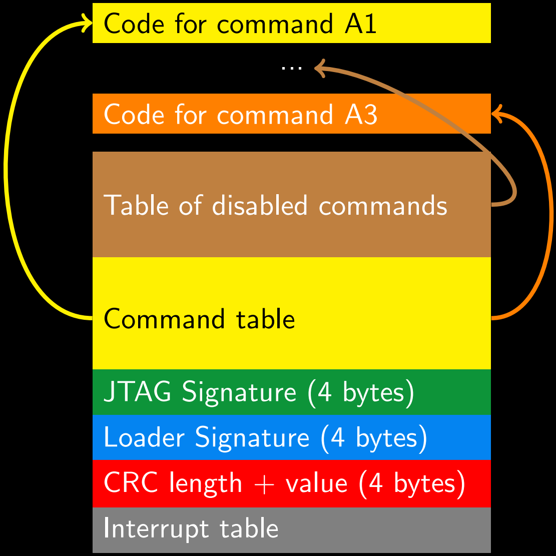
Live demo: Reading the FRAM



```
Block 0D A1 00 D5 02 C8 44 E1 00
Block 0E D8 02 C8 30 A1 00 D3 02
Block 0F C8 1C A1 00 1E 03 C8 68 Last trend record History records
Block 10 62 00 EC 02 C8 E8 61 00
Block 11 D7 02 C8 94 61 00 D7 02
Block 12 C8 48 A1 00 00 00 00 00
Block 13 00 00 00 00 00 00 00 00
Block 14 00 00 00 00 00 00 00 00
Block 15 00 00 00 00 00 00 00 00
Block 16 00 00 00 00 00 00 00 00
Block 17 00 00 00 00 00 00 00 00
Block 18 00 00 00 00 00 00 00 00
Block 19 00 00 00 00 00 00 00 00
Block 1A 00 00 00 00 00 00 00 00
Block 1B 00 00 00 00 00 00 00 00
Block 1C 00 00 00 00 00 00 00 00
Block 1D 00 00 00 00 00 00 00 00
Block 1E 00 00 00 00 00 00 00 00
Block 1F 00 00 00 00 00 00 00 00
Block 20 00 00 00 00 00 00 00 00
Block 21 00 00 00 00 00 00 00 00
Block 22 00 00 00 00 00 00 00 00
Block 23 00 00 00 00 00 00 00 00
Block 24 00 00 00 00 00 00 00 00
Block 25 00 00 00 00 00 00 00 00
Block 27 00 00 00 00 44 00 00 00 Last history record Wear time
Block 28 BA 32 00 01 BA 32 00 01 Sensor Region
Trend index: 3
Historic index: 4
Trend Glucose level   : 72.0 mg/dL
Historic Glucose level: 0.0 mg/dL
Sensor bytes: high=0x0 low=0x44
Sensor running since 68 minutes (1:08:00)
```

FRAM layout: code and tables

- Command table begins and ends with `AB AB`
- Each command entry is `aa aa cc cc`:
  - `aa aa`: address
  - `cc cc`: command identifier e.g. `E2 00`
- JTAG signature: `00 00 00 00` (unlocked)
- NFC Commands E0 - E2 are disabled
- New NFC commands: A0 - A4



Code for command A1

...

Code for command A3

Table of disabled commands

Command table

JTAG Signature (4 bytes)

Loader Signature (4 bytes)

CRC length + value (4 bytes)

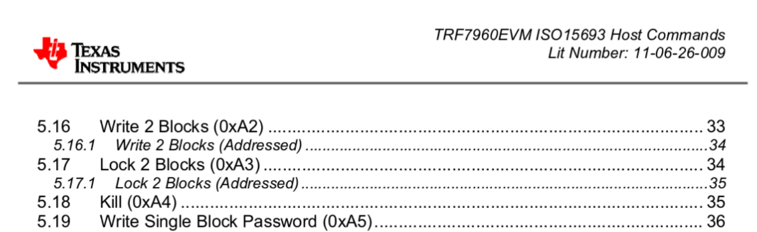
Interrupt table

MSP430

Remember that the RF430 is a microcontroller.
It runs software, and we'd like to read that software.

Custom NFC commands

http://www.ti.com/lit/an/sloa141/sloa141.pdf



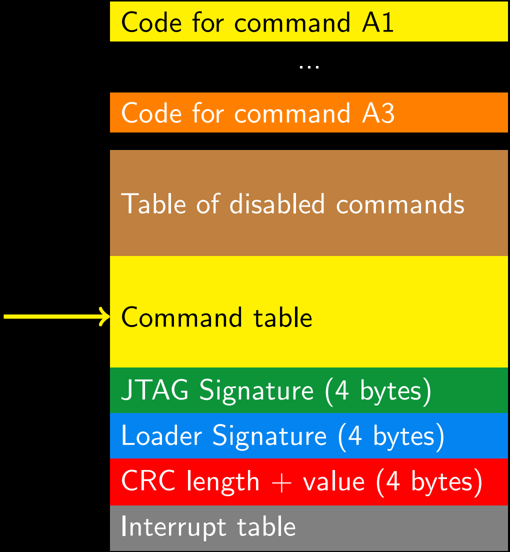
TEXAS INSTRUMENTS

TRF7960EVM ISO15693 Host Commands
Lit Number: 11-06-26-009

This sensor has different custom commands + yet additional ones!

Specific NFC commands

They are declared here



Code for command A1

...

Code for command A3

Table of disabled commands

Command table

JTAG Signature (4 bytes)

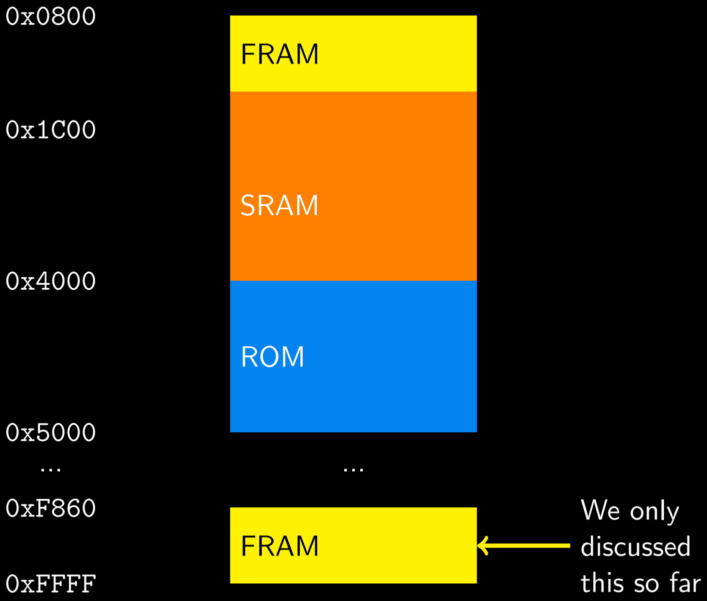
Loader Signature (4 bytes)

CRC length + value (4 bytes)

Interrupt table

Activate, Get Patch Info, Lock/Unlock tag, Raw Read...

Other parts of memory



0x0800 FRAM

0x1C00 SRAM

0x4000 ROM

0x5000

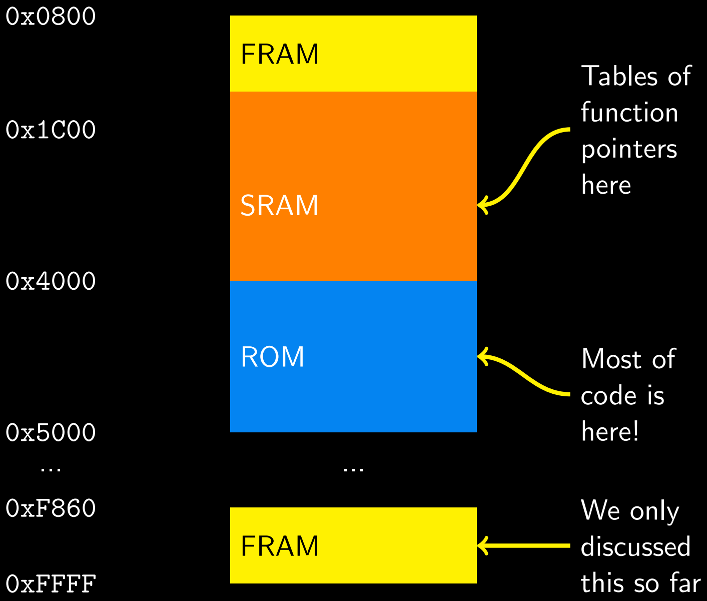
… …

0xF860 FRAM

0xFFFF

We only discussed this so far

Other parts of memory



0x0800

FRAM

0x1C00

SRAM

0x4000

ROM

0x5000

...

...

0xF860

FRAM

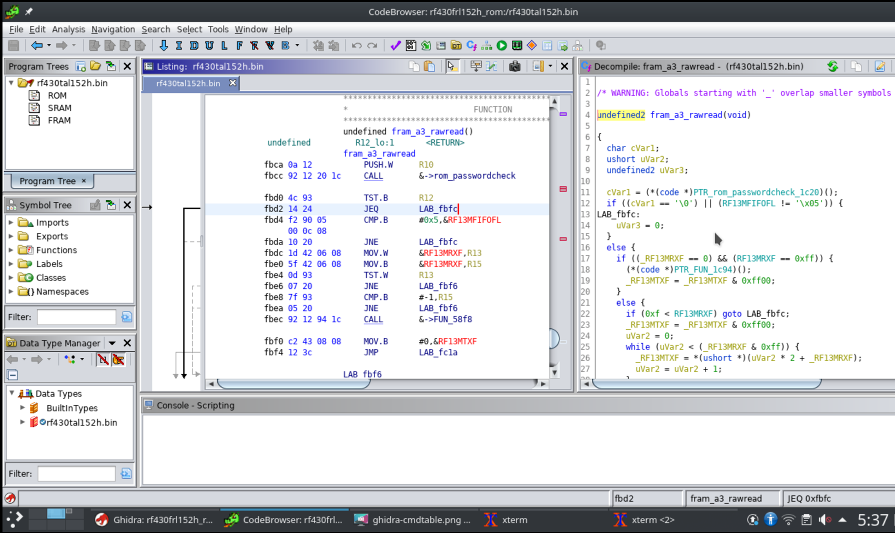
0xFFFF

Tables of function pointers here

Most of code is here!

We only discussed this so far

A3 Raw Read



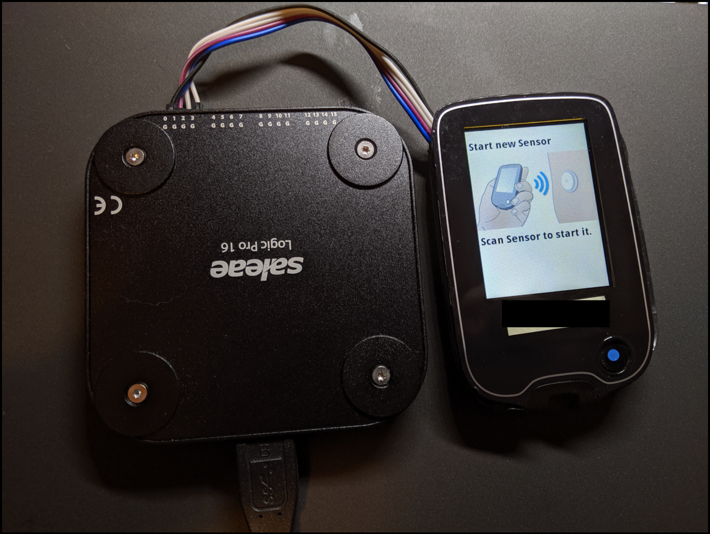
```
                          *************************************************
                          *                              FUNCTION
                          *************************************************
                          undefined fram_a3_rawread()
         undefined          R12_lo:1       <RETURN>
                          fram_a3_rawread
        fbca 0a 12             PUSH.W      R10
        fbcc 92 12 20 1c       CALL        &->rom_passwordcheck

        fbd0 4c 93             TST.B       R12
        fbd2 14 24             JEQ         LAB_fbfc
        fbd4 f2 90 05          CMP.B       #0x5,&RF13MFIFOFL
             00 0c 08
        fbda 10 20             JNE         LAB_fbfc
        fbdc 1d 42 06 08       MOV.W       &RF13MRXF,R13
        fbe0 5f 42 06 08       MOV.B       &RF13MRXF,R15
        fbe4 0d 93             TST.W       R13
        fbe6 07 20             JNE         LAB_fbf6
        fbe8 7f 93             CMP.B       #-1,R15
        fbea 05 20             JNE         LAB_fbf6
        fbec 92 12 94 1c       CALL        &->FUN_58f8

        fbf0 c2 43 08 08       MOV.B       #0,&RF13MTXF
        fbf4 12 3c             JMP         LAB_fc1a

                          LAB_fbf6
```

```

/* WARNING: Globals starting with '_' overlap smaller symbols

undefined2 fram_a3_rawread(void)

{
  char cVar1;
  ushort uVar2;
  undefined2 uVar3;

  cVar1 = (*(code *)PTR_rom_passwordcheck_1c20)();
  if ((cVar1 == '\0') || (RF13MFIFOFL != '\x05')) {
LAB_fbfc:
    uVar3 = 0;
  }
  else {
    if ((_RF13MRXF == 0) && (RF13MRXF == 0xff)) {
      (*(code *)PTR_FUN_1c94)();
      _RF13MTXF = _RF13MTXF & 0xff00;
    }
    else {
      if (0xf < RF13MRXF) goto LAB_fbfc;
      _RF13MTXF = _RF13MTXF & 0xff00;
      uVar2 = 0;
      while (uVar2 < (_RF13MRXF & 0xff)) {
        _RF13MTXF = *(ushort *)(uVar2 * 2 + _RF13MRXF);
        uVar2 = uVar2 + 1;
```

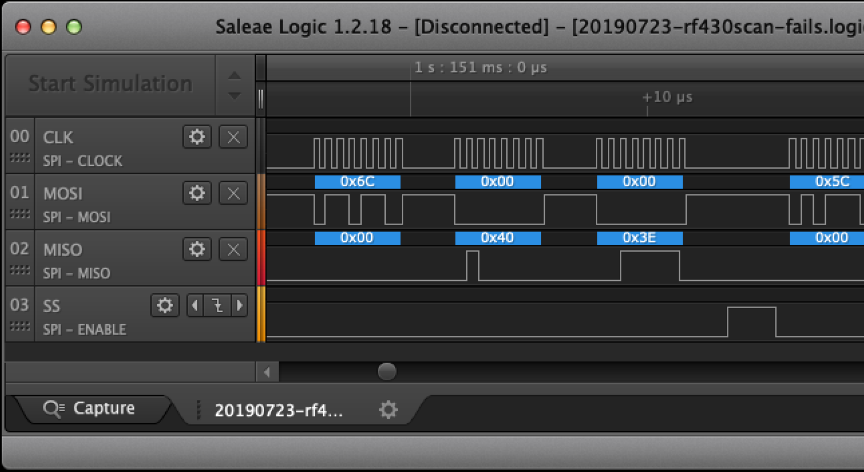
A3 Raw Read

Parameters:
4-byte password.
2-byte raw address.
1-byte length.

Sniffing the Password



Sniffing the Password



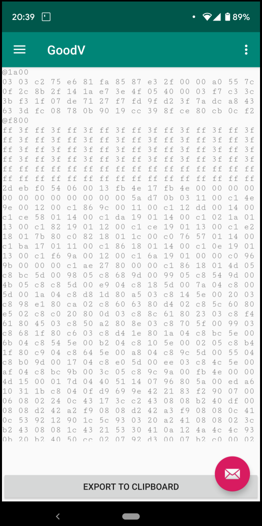
Sniffing the Password

The password is revealed in when the sensor is initialized, as all custom commands share the same password.

Custom NFC Commands

Command	Example
Initialize	02 **A0** 07 DEADBEEF
Info	02 **A1** 07
Lock	02 **A2** 07 DEADBEEF
Raw Read	02 **A3** 07 DEADBEEF F0FF 06
Unlock	02 **A4** 07 DEADBEEF

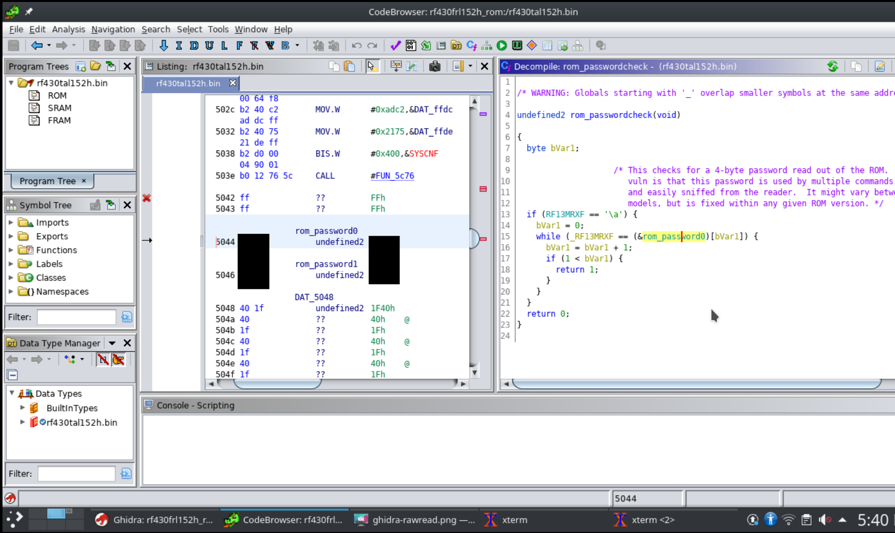
GoodV – An App for the RF430



```
@1a00
03 03 c2 75 e6 81 fa 85 87 e3 2f 00 00 a0 55 7c
0f 2c 8b 2f 14 1a e7 3e 4f 05 40 00 03 f7 c3 3c
3b f3 1f 07 de 71 27 f7 fd 9f d2 3f 7a dc a8 43
63 3d fc 08 78 0b 90 19 cc 39 8f ce 80 cb 0c f2
@f800
ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f
ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f
ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f
ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f ff 3f
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
2d eb f0 54 06 00 13 fb 4e 17 fb 4e 00 00 00 00
00 00 00 00 00 00 00 00 5a d7 0b 03 11 00 c1 4e
9e 00 12 00 c1 86 9c 00 11 00 c1 12 dd 00 14 00
c1 ce 58 01 14 00 c1 da 19 01 14 00 c1 02 1a 01
13 00 c1 82 19 01 12 00 c1 ce 19 01 13 00 c1 e2
18 01 7b 80 c0 82 18 01 1c 00 c0 76 57 01 14 00
c1 ba 17 01 11 00 c1 86 18 01 14 00 c1 0e 19 01
13 00 c1 f6 9a 00 12 00 c1 6a 19 01 00 00 c0 96
9b 00 00 00 c1 ae 27 80 00 00 c1 86 18 01 4d 05
c8 bc 5d 00 98 05 c8 68 9d 00 99 05 c8 54 9d 00
4b 05 c8 c8 5d 00 e9 04 c8 18 5d 00 7a 04 c8 00
5d 00 1a 04 c8 d8 1d 80 a5 03 c8 14 5e 00 20 03
c8 98 e1 80 ca 02 c8 60 63 80 d4 02 c8 5c 60 80
e5 02 c8 c0 20 80 0d 03 c8 8c 61 80 23 03 c8 f4
61 80 45 03 c8 50 a2 80 8e 03 c8 70 5f 00 99 03
c8 68 1f 80 c6 03 c8 d4 1e 80 1a 04 c8 bc 5e 00
6b 04 c8 54 5e 00 b2 04 c8 10 5e 00 02 05 c8 b4
1f 80 c9 04 c8 64 5e 00 a8 04 c8 9c 5d 00 55 04
c8 b0 9d 00 17 04 c8 e0 5d 00 ee 03 c8 4c 5e 00
af 04 c8 bc 9b 00 3c 05 c8 9c 9a 00 fb 4e 00 00
4d 15 00 01 7d 04 40 51 14 07 96 80 5a 00 ed a6
10 31 1b c8 04 0f d9 69 9e 42 21 83 f2 90 07 00
06 08 02 24 0c 43 17 3c c2 43 08 08 b2 40 df 00
08 08 d2 42 a2 f9 08 08 d2 42 a3 f9 08 08 0c 41
0c 53 92 12 90 1c 5c 93 03 20 a2 41 08 08 02 3c
b2 43 08 08 1c 43 21 53 30 41 0a 12 4a 4c 4c 93
0b 20 b2 40 50 cc 02 07 92 d3 00 07 b2 c0 00 02
```

Android app can read raw memory, giving full dumps of the ROM for reverse engineering in GHIDRA.

Password Check



```
          00 64 f8
    502c b2 40 c2        MOV.W      #0xadc2,&DAT_ffdc
          ad dc ff
    5032 b2 40 75        MOV.W      #0x2175,&DAT_ffde
          21 de ff
    5038 b2 d0 00        BIS.W      #0x400,&SYSCNF
          04 90 01
    503e b0 12 76 5c     CALL       #FUN_5c76

    5042 ff              ??         FFh
    5043 ff              ??         FFh

                      rom_password0
    5044 [redacted]      undefined2 [redacted]

                      rom_password1
    5046                 undefined2

                      DAT_5048
    5048 40 1f           undefined2 1F40h
    504a 40              ??         40h    @
    504b 1f              ??         1Fh
    504c 40              ??         40h    @
    504d 1f              ??         1Fh
    504e 40              ??         40h    @
    504f 1f              ??         1Fh
```

```
/* WARNING: Globals starting with '_' overlap smaller symbols at the same addre

undefined2 rom_passwordcheck(void)

{
  byte bVar1;

                    /* This checks for a 4-byte password read out of the ROM.
                       vuln is that this password is used by multiple commands
                       and easily sniffed from the reader.  It might vary betw
                       models, but is fixed within any given ROM version. */
  if (RF13MRXF == '\a') {
    bVar1 = 0;
    while (_RF13MRXF == (&rom_password0)[bVar1]) {
      bVar1 = bVar1 + 1;
      if (1 < bVar1) {
        return 1;
      }
    }
  }
  return 0;
}
```

FRAM Command Table

`A0` initializes the sensor.

`A1` identifies the sensor.

`A2` write-protects all of FRAM.

`A3` reads from a raw address.

`A4` unlocks all blocks.

`E0`, `E1`, and `E2` are not yet understood.

Writing the FRAM

Normally, the sensor is locked

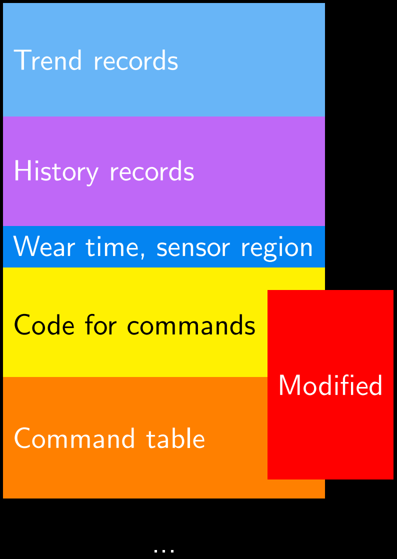
```
proxmark3> hf 15 cmd write u 03 62 C2 00 00 00 00 00 00
Tag returned Error 18: The specified block is locked and
its content cannot be changed.
```

Unlock the sensor

```
proxmark3> hf 15 cmd raw -c 02 A4 07 DE AD BE EF
received 3 octets
00 78 F0
proxmark3> hf 15 cmd write u 03 AA BB CC DD 00 00 00 00 00
OK
proxmark3> hf 15 cmd read u 03
AA BB CC DD 00 00 00 00 00
```

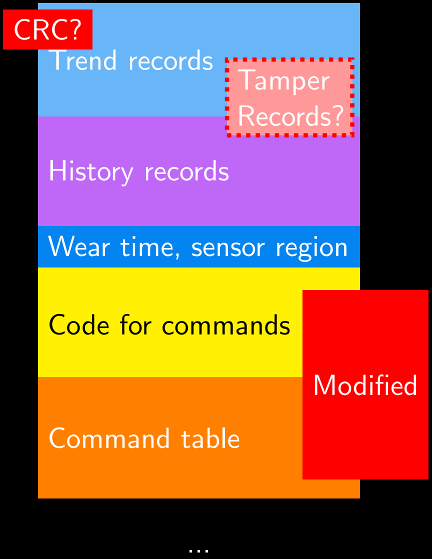
Importance and mitigations

- We can **tamper** with the memory
- E.g modify firmware!
- Limitation: a few blocks are **not writable** (0x00-0x03, 0xef)



Importance and mitigations

- We can **tamper** with the memory
- E.g modify firmware!
- Limitation: a few blocks are **not writable** (0x00-0x03, 0xef)
- We **cannot modify glucose measures or wear time** *yet*: they are protected by a **checksum**: we are uncertain about the algo *yet* and its location



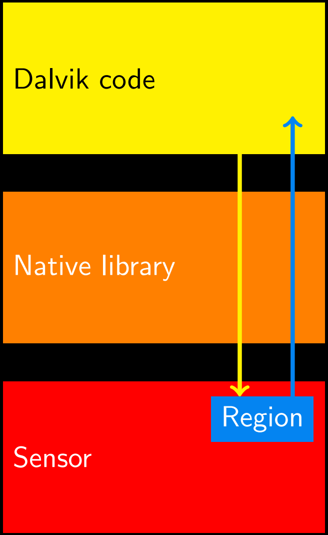
Medical threat or not?

- Requires NFC **proximity**
- Vendor **fixed this in new model**, released in August/October 2018. However some pharmacies are still currently **shipping old versions**.
- Diabetic patients usually know **how they feel** at a given glucose level
- The sensor **does not inject insulin**
- Hospitals use **blood glucose tests**
- An attacker can probably **mess up** things, but unlikely to be *lethal*. This is not *Homeland* TV series!

Complicated

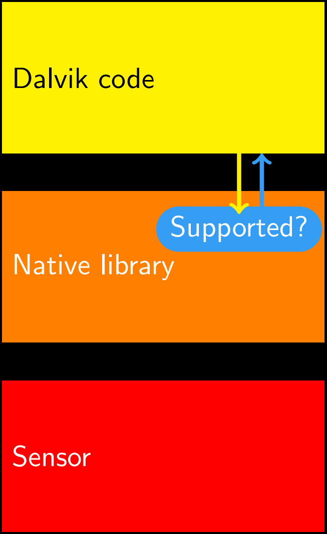
1. Introduction
   - Who are we?
   - Medical background
2. Hardware
   - Teardown
   - Read The Datasheet
3. NFC
   - FRAM: user data
   - FRAM: code and tables
   - Dumping the firmware: Raw Read
   - Lock/Unlock vuln
4. **Sensor expiration date: how does it work?**
5. Conclusion

Expiration date: the 14-day limit



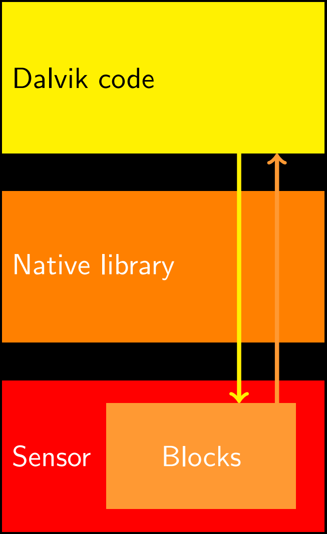
1. Get sensor Info: custom NFC command. Returns **region**.

Expiration date: the 14-day limit



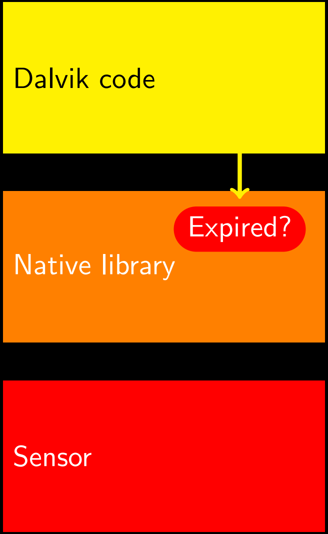
1. Get sensor Info: custom NFC command. Returns **region**.
2. Is sensor supported? Check app region matches sensor. Implemented in native layer.

Expiration date: the 14-day limit



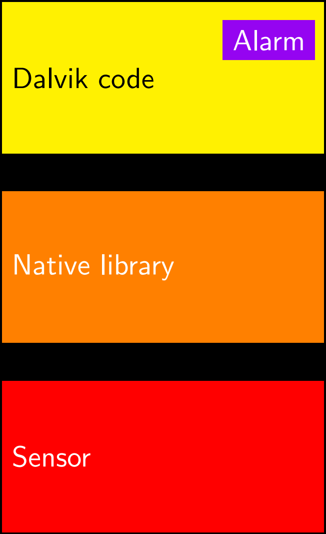
1. Get sensor Info: custom NFC command. Returns **region**.
2. Is sensor supported? Check app region matches sensor. Implemented in native layer.
3. Read Multiple Blocks: **blocks** 0x00 to 0x2a.

Expiration date: the 14-day limit



1. Get sensor Info: custom NFC command. Returns **region**.
2. Is sensor supported? Check app region matches sensor. Implemented in native layer.
3. Read Multiple Blocks: **blocks** 0x00 to 0x2a.
4. Supply **block dump** and check **expiration** date. Implemented in native layer.

Expiration date: the 14-day limit


Alarm

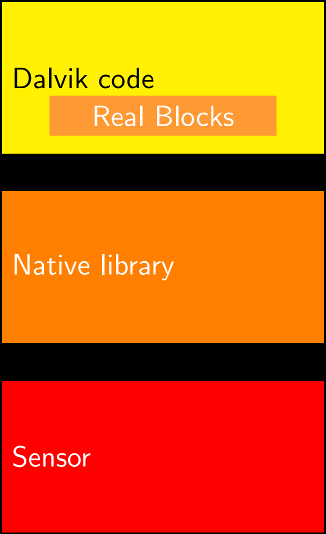
Dalvik code

Native library

Sensor

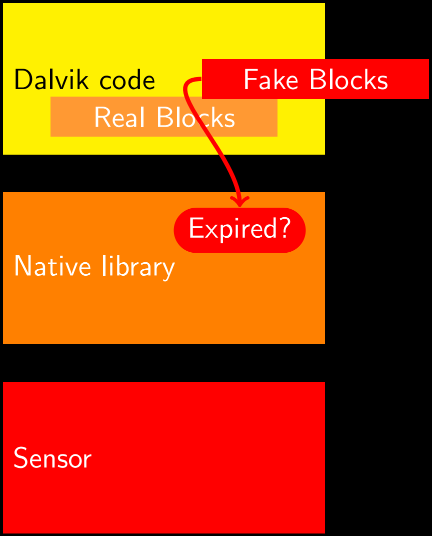
1. Get sensor Info: custom NFC command. Returns **region**.
2. Is sensor supported? Check app region matches sensor. Implemented in native layer.
3. Read Multiple Blocks: **blocks** 0x00 to 0x2a.
4. Supply **block dump** and check **expiration** date. Implemented in native layer.
5. Add sensor to database and set **alarm** for expiration date.

Hook expiration check



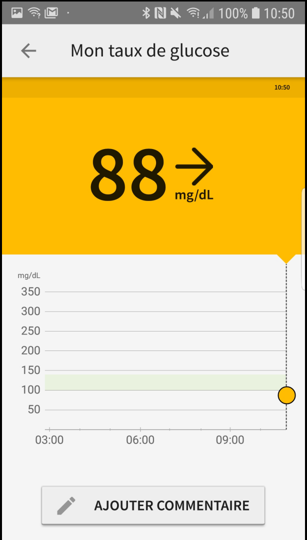
- The native library is **obfuscated**
- We replace the blocks with blocks from a **new, unexpired sensor**.
- **It works**!

Hook expiration check



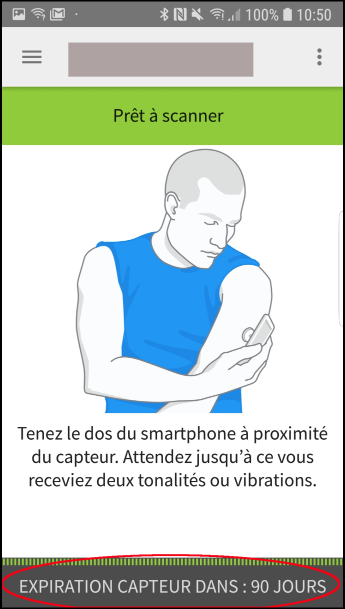
- The native library is **obfuscated**
- We replace the blocks with blocks from a **new, unexpired sensor**.
- **It works**!

Frida hook demo



```
[*] Inside getPatchTimeValues(): parserType=10
Warm up minutes = 60
Wear minutes = 20160
Patched wear minutes = 129600
[*] returned: true
[*] Inside processScan(): type=1095774808 warm
dump=4904b07...
patched dump=f418b0320...
processScan returned: SUCCESS
```

Oops! 90 days?! :)



[redacted]

Prêt à scanner

Tenez le dos du smartphone à proximité du capteur. Attendez jusqu’à ce vous receviez deux tonalités ou vibrations.

EXPIRATION CAPTEUR DANS : 90 JOURS

1. Introduction
   - Who are we?
   - Medical background
2. Hardware
   - Teardown
   - Read The Datasheet
3. NFC
   - FRAM: user data
   - FRAM: code and tables
   - Dumping the firmware: Raw Read
   - Lock/Unlock vuln
4. Sensor expiration date: how does it work?
5. Conclusion

Conclusion

- Nice IoT design and implementation
- Write vulnerability (fixed in v2)
- Check expiration from sensor data
- Interesting to **hack** your sensors (beware)
- Highest security threat *is not the sensor* but **a compromised smartphone**! Be safe!

Thank You

Contact us: @cryptax @travisgoodspeed

Thanks to

Anonymous diabetic contacts :) and
@PagetPhil @TuxDePoinsisse @aurelsec @trufae
@_j3lena_ @Baldanos @r00tbsd @doegox
@herrmann1001 BigEZ

```
BA19/badge-30aea4d3a90b/vote Track 1:5
BA19/badge-30aea47855d6/vote Track 1:5
BA19/badge-30aea4ee73a2/vote Track 1:5
BA19/badge-30aea4b40aec/vote Track 1:5
BA19/badge-30aea4fc564c/vote Track 1:5
```