



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



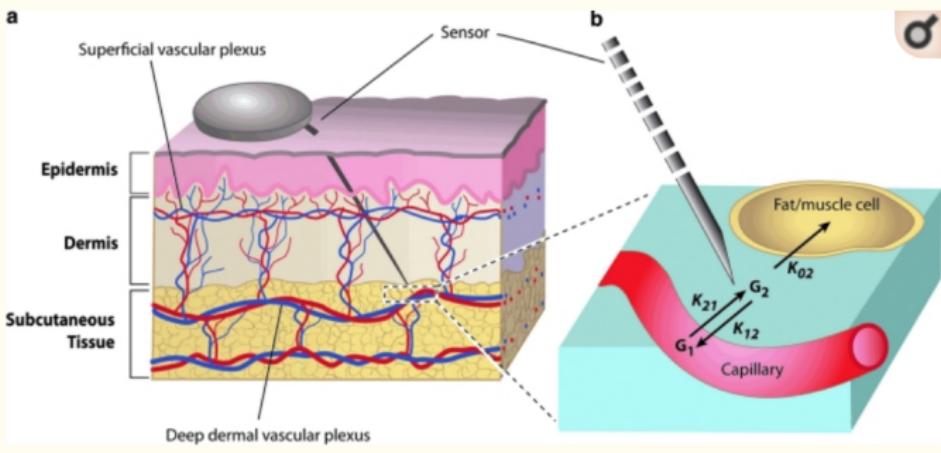
Flash Glucose Monitoring system

Diabetes Technol Ther. 2009 Jun; 11(Suppl 1): S-11–S-16.
doi: [10.1089/dia.2009.0002](https://doi.org/10.1089/dia.2009.0002)

PMCID: PMC2903977
PMID: [19469670](https://pubmed.ncbi.nlm.nih.gov/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/>



The inner guts of a connected glucose sensor for diabetes - A. Apvrille, T. Goodspeed

Sensor life cycle

Assemble pack



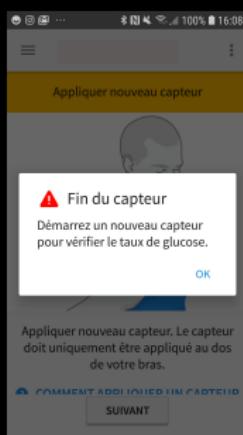
Apply sensor



Activate it (60 min)



Use it



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

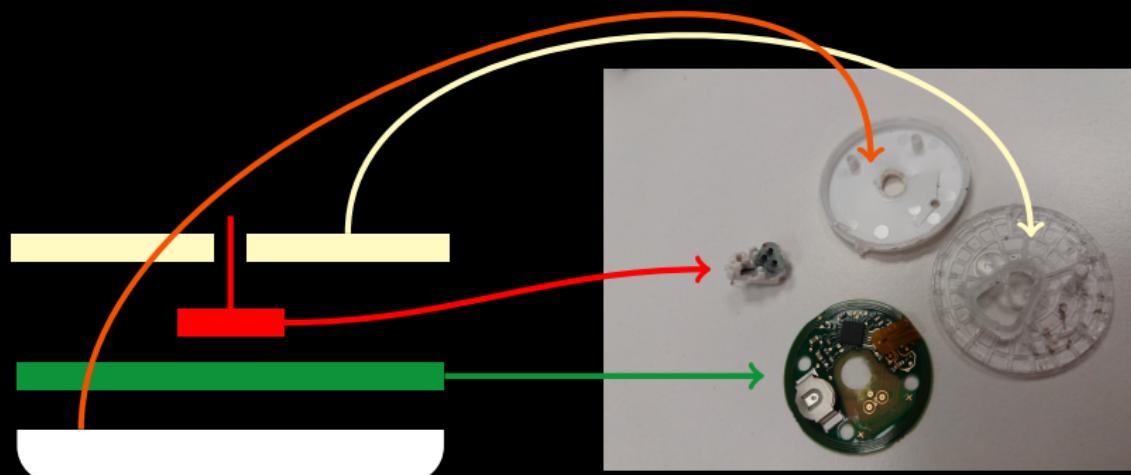
4 Sensor expiration date: how does it work?

5 Conclusion



Sensor Teardown

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



Other teardowns:

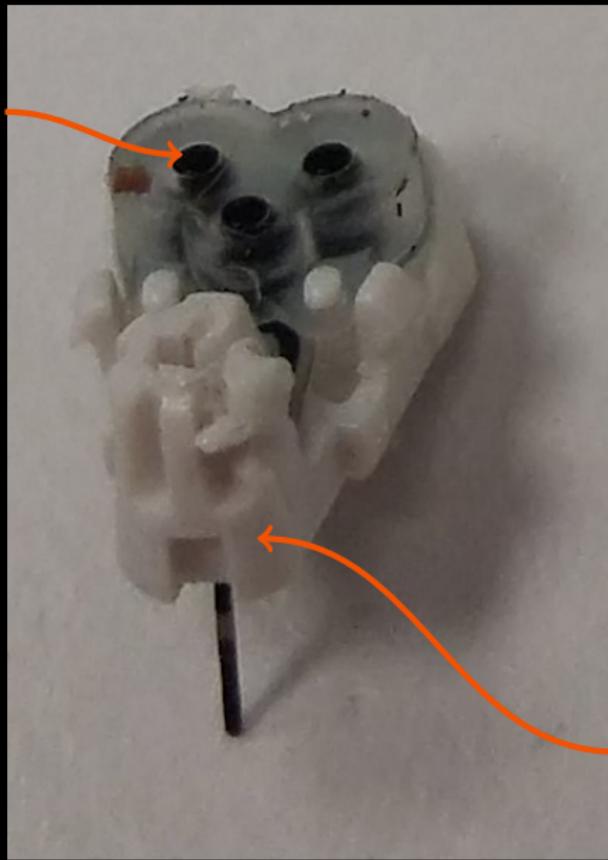
<https://www.youtube.com/watch?v=40RXFhZp8hg>

<https://www.youtube.com/watch?v=sYIm97wj10o>



Enzyme sensor

3 electrode contacts

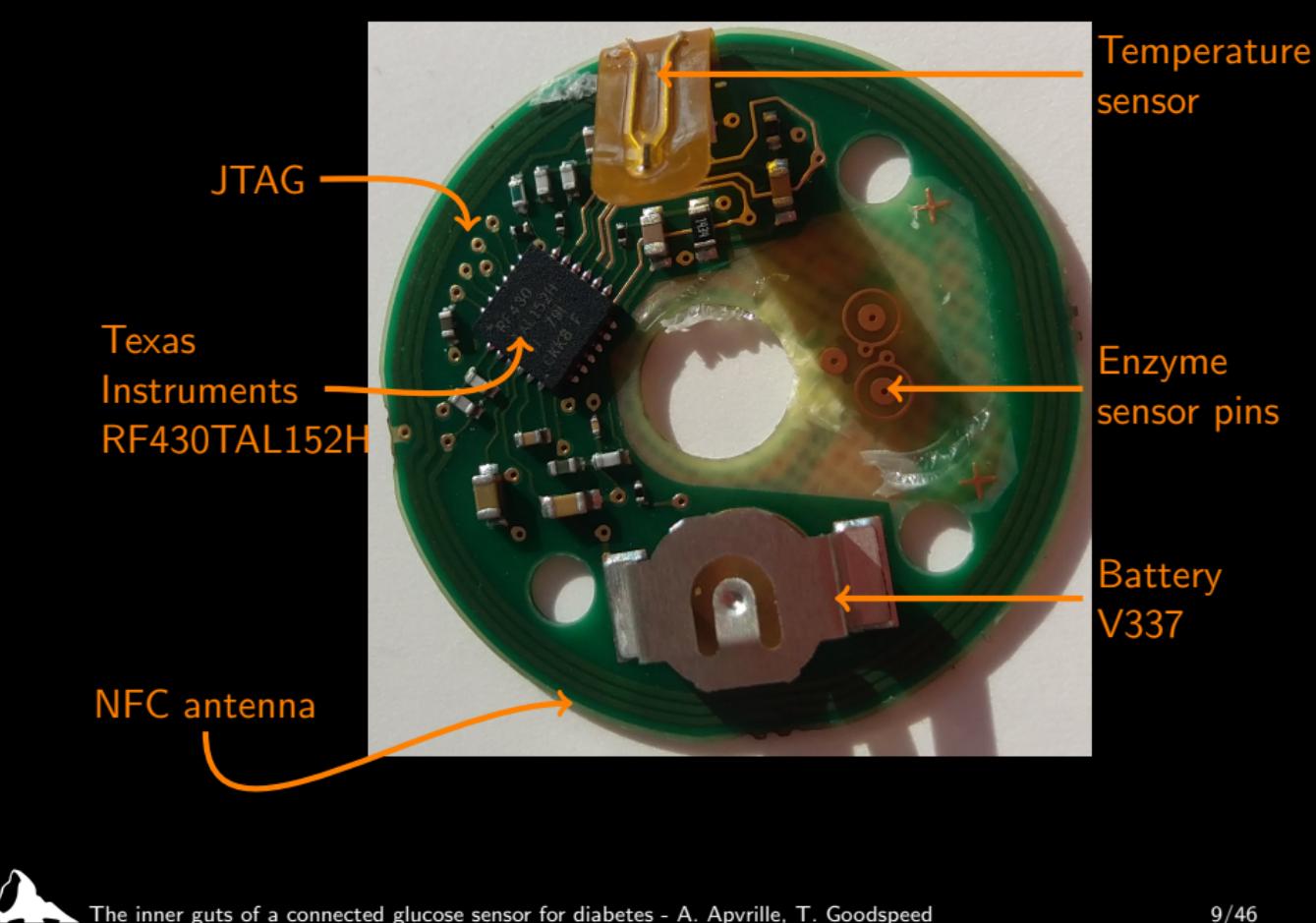


Covered with Glu-
cose Oxydase
(GOx)

5mm long,
0.4mm wide



PCB



Datasheet



RF430FRL152H, RF430FRL153H, RF430FRL154H

SLAS834C – NOVEMBER 2012 – REVISED DECEMBER 2014

RF430FRL15xH NFC ISO 15693 Sensor Transponder

1 Device Overview

1.1 Features

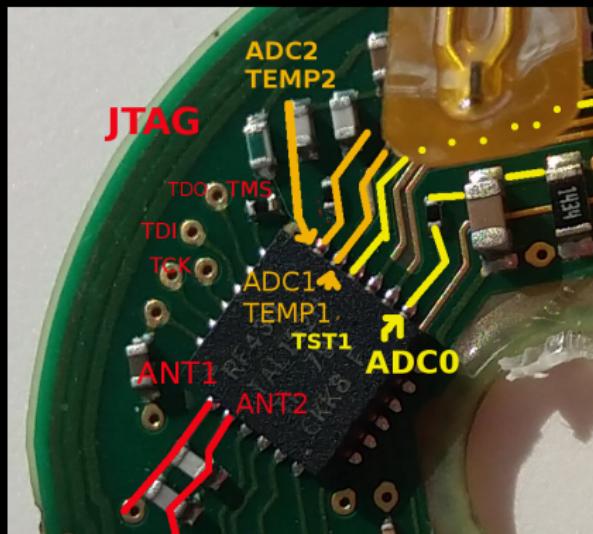
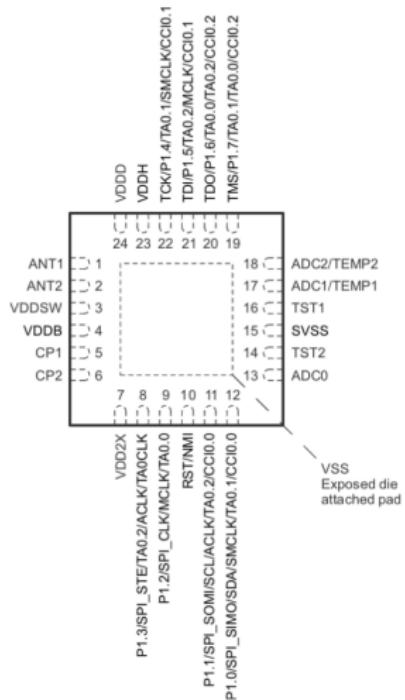
- ISO/IEC 15693, ISO/IEC 18000-3 (Mode 1) Compliant RF Interface
- Power Supply System With Either Battery or 13.56-MHz H-Field Supply
- 14-Bit Sigma-Delta Analog-to-Digital Converter (ADC)
- Internal Temperature Sensor
- Resistive Sensor Bias Interface
- CRC16 CCITT Generator
- MSP430™ Mixed-Signal 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 μ A
 - V_{OH} Higher Than ($V_{DDB} - 0.15$ V) at 400 μ A
 - Timer_A PWM Signal Available on All Ports
- eUSCI_B Module Supports 3-Wire and 4-Wire SPI and I²C
- 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



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

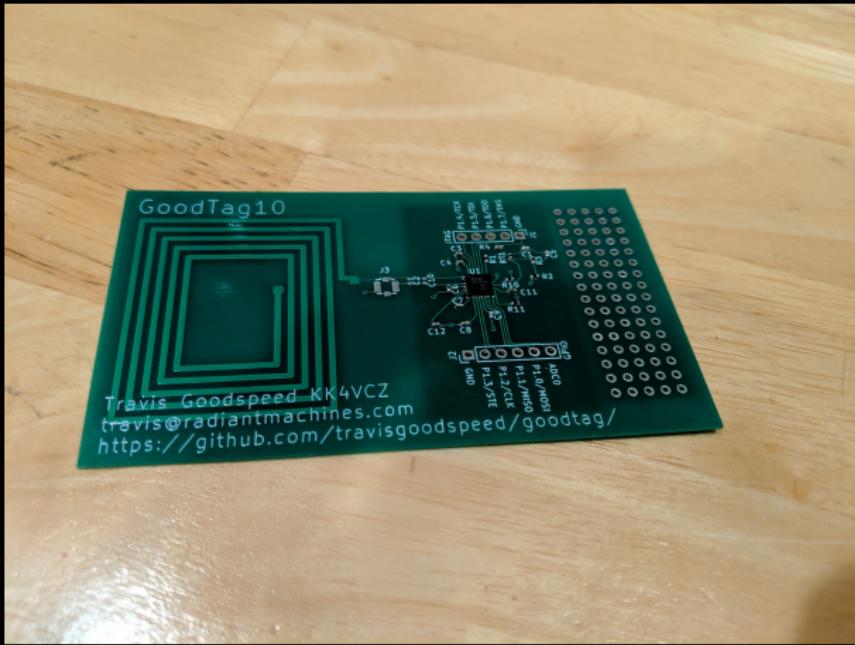


JTAG



The inner guts of a connected glucose sensor for diabetes - A. Apvrille, T. Goodspeed

Custom Carrier Board



Custom Carrier Board



The inner guts of a connected glucose sensor for diabetes - A. Apvrille, T. Goodspeed

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

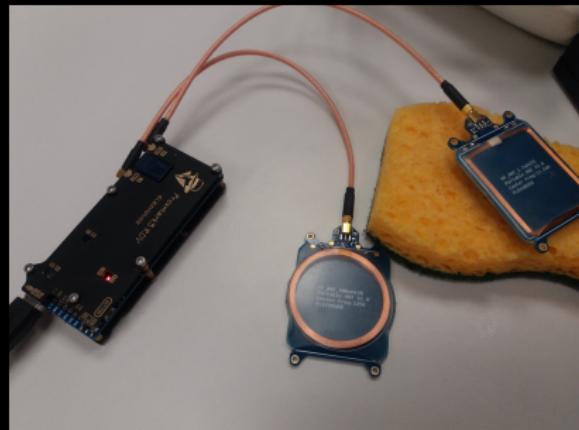
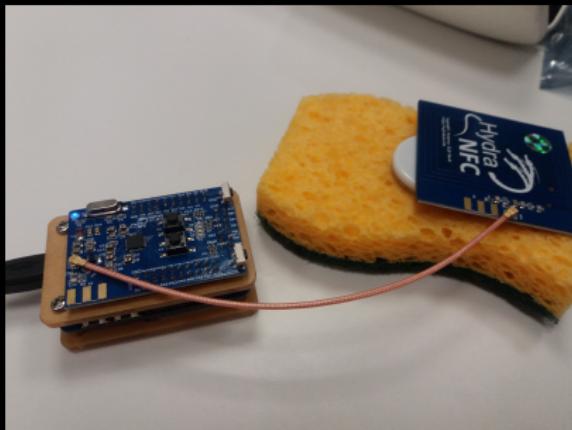
Lock/Unlock vuln

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
```

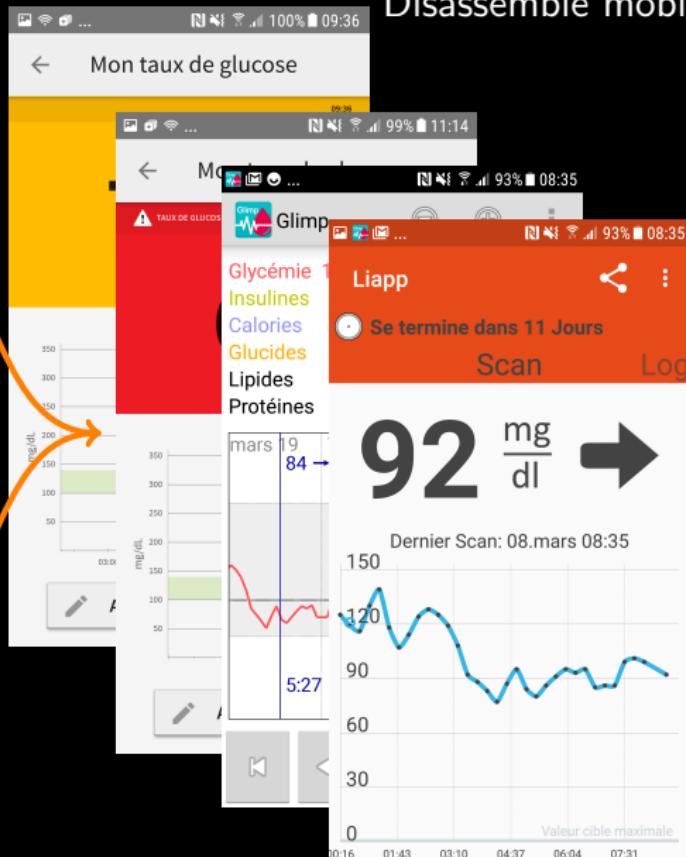


Understanding the memory layout

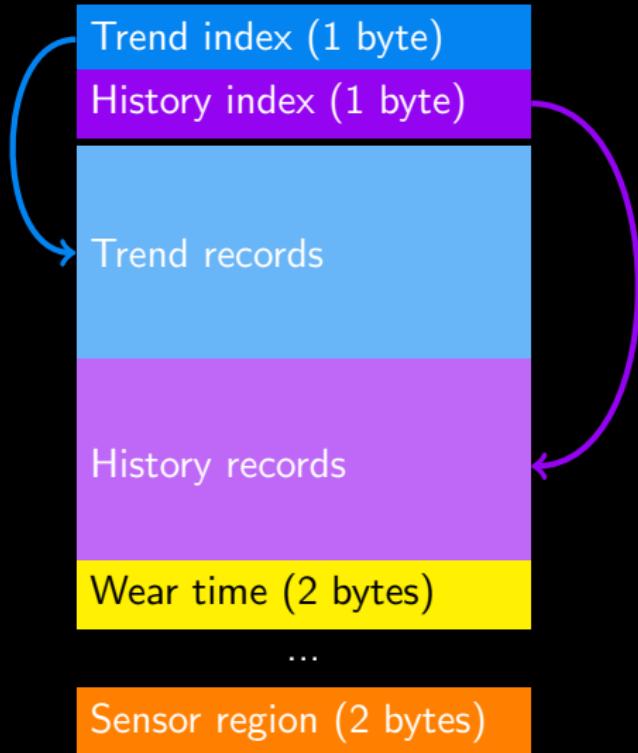
Disassemble mobile apps



Numerous tests



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



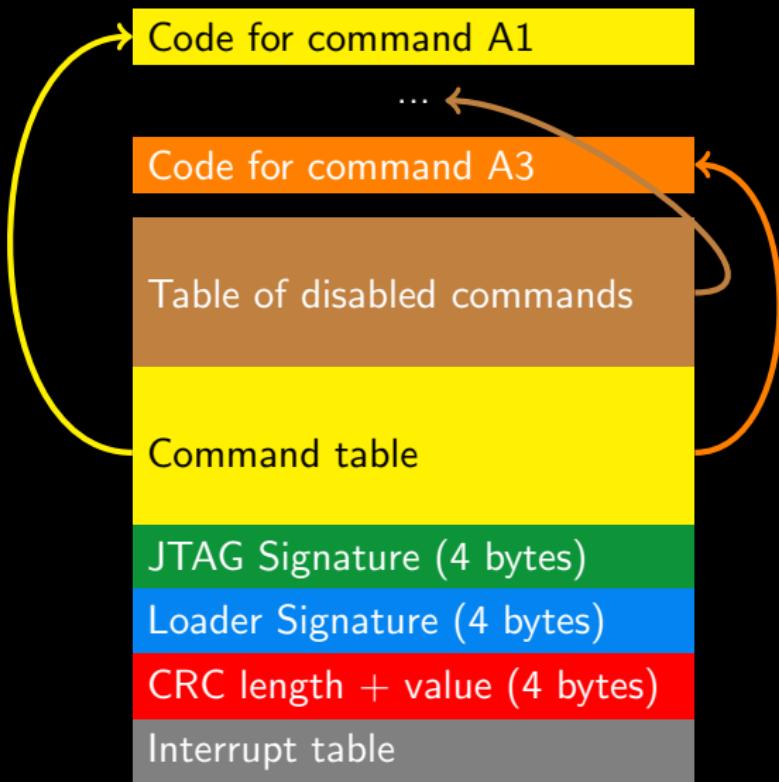
Live demo: Reading the FRAM

```
Block 00 00 00 00 00 00 00 00 00 00 00 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 00  
Block 13 00 00 00 00 00 00 00 00 00  
Block 14 00 00 00 00 00 00 00 00 00  
Block 15 00 00 00 00 00 00 00 00 00  
Block 16 00 00 00 00 00 00 00 00 00  
Block 17 00 00 00 00 00 00 00 00 00  
Block 18 00 00 00 00 00 00 00 00 00  
Block 19 00 00 00 00 00 00 00 00 00  
Block 1A 00 00 00 00 00 00 00 00 00  
Block 1B 00 00 00 00 00 00 00 00 00  
Block 1C 00 00 00 00 00 00 00 00 00  
Block 1D 00 00 00 00 00 00 00 00 00  
Block 1E 00 00 00 00 00 00 00 00 00  
Block 1F 00 00 00 00 00 00 00 00 00  
Block 20 00 00 00 00 00 00 00 00 00  
Block 21 00 00 00 00 00 00 00 00 00  
Block 22 00 00 00 00 00 00 00 00 00  
Block 23 00 00 00 00 00 00 00 00 00  
Block 24 00 00 00 00 00 00 00 00 00  
Block 25 00 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



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

5.16	Write 2 Blocks (0xA2)	33
5.16.1	<i>Write 2 Blocks (Addressed)</i>	34
5.17	Lock 2 Blocks (0xA3)	34
5.17.1	<i>Lock 2 Blocks (Addressed)</i>	35
5.18	Kill (0xA4)	35
5.19	Write Single Block Password (0xA5)	36

This sensor has different custom
commands + yet additional ones!



Specific NFC commands

They are declared here



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

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



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



Tables of
function
pointers
here

0x1C00

SRAM

0x4000

ROM

0x5000

...

Most of
code is
here!

0xF860

...

0xFFFF



We only
discussed
this so far



A3 Raw Read

The screenshot shows the Ghidra IDE interface with the following windows:

- Program Trees**: Shows a tree for rf430tal152h.bin with ROM, SRAM, and FRAM nodes.
- Symbol Tree**: Shows symbols like Imports, Exports, Functions, Labels, Classes, and Namespaces.
- Data Type Manager**: Shows Data Types, BuiltInTypes, and a specific entry for rf430tal152h.bin.
- Listing: rf430tal152h.bin**: Displays assembly code for the function fram_a3_rawread. The assembly code includes instructions like PUSH.W, CALL, TST.B, JEQ, CMP.B, JNE, MOV.W, MOV.B, TST.W, and CMP.B. Labels LAB_fbfc, LAB_fb6, and LAB_fcl are also present.
- Decompile: fram_a3_rawread - (rf430tal152h.bin)**: Shows the decompiled C-like pseudocode for the function. It includes global declarations for cVar1, uVar2, and uVar3, and a loop that increments uVar2 by 2 and checks if it is less than _RF13MRXF.
- Console - Scripting**: A command-line interface for running scripts.
- Status Bar**: Shows tabs for Ghidra, CodeBrowser, ghidra-cmdtable.png, and xterm, along with the current time (5:37).



A3 Raw Read

Parameters:

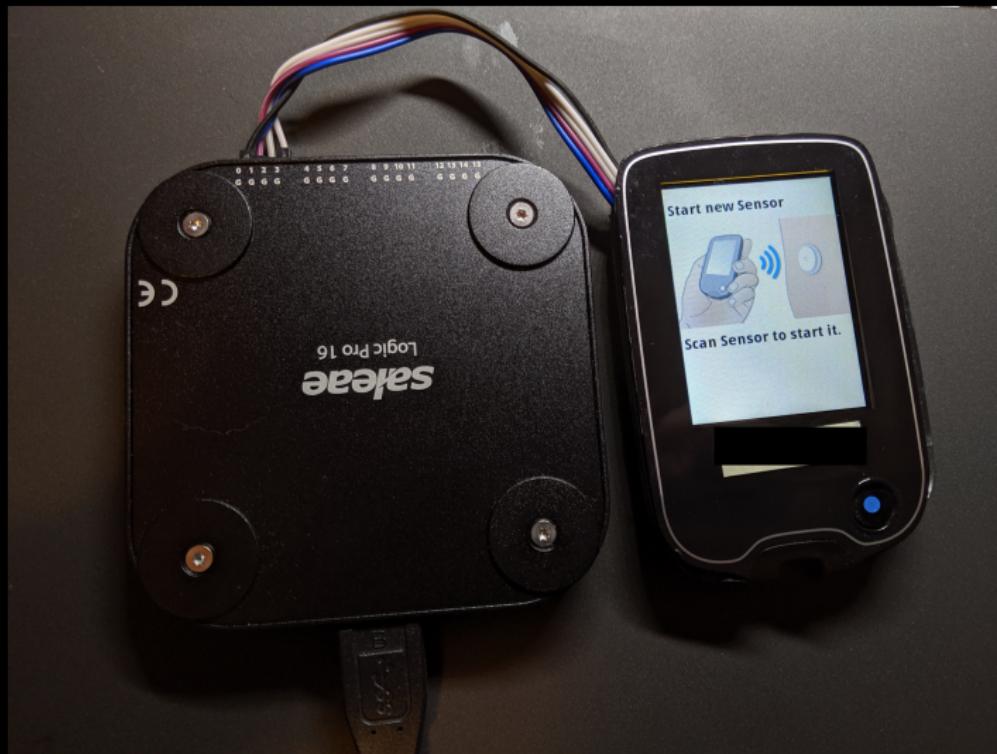
4-byte password.

2-byte raw address.

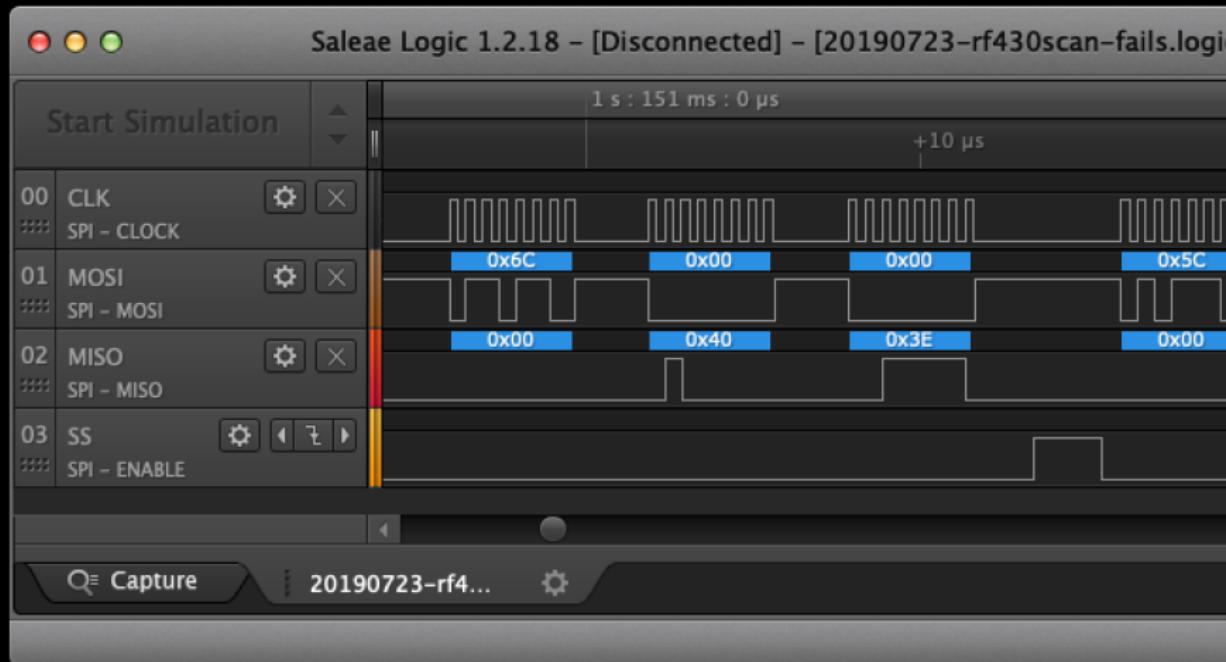
1-byte length.



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



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



Password Check

The screenshot shows the Ghidra IDE interface with the following windows:

- Program Trees**: Shows the file structure with `rf430tal152h.bin` containing ROM, SRAM, and FRAM sections.
- Listing: rf430tal152h.bin**: Displays assembly code for the `rf430tal152h.bin` file. A specific section of the code is highlighted:

```
00 64 f8    MOV.W    #0xadcc2, &DAT_ffdc
502c b2 40 c2 ad dc ff
5032 b2 40 75 MOV.W    #0x2175, &DAT_ffde
21 de ff
5038 b2 d0 00 BIS.W    #0x400, &SYSCHF
04 90 01
503e b0 12 76 5c CALL     #FUN_5c76

5042 ff      ??        FFh
5043 ff      ??        FFh
```
- Decompile: rom_passwordcheck - (rf430tal152h.bin)**: Shows the decompiled C-like pseudocode for the `rom_passwordcheck` function:

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

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 between
     * 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;
}
```
- Console - Scripting**: An empty console window.

The status bar at the bottom shows the current memory address as `5044`, the file name as `Ghidra: rf430frl152h...`, and the time as `5:40`.



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)

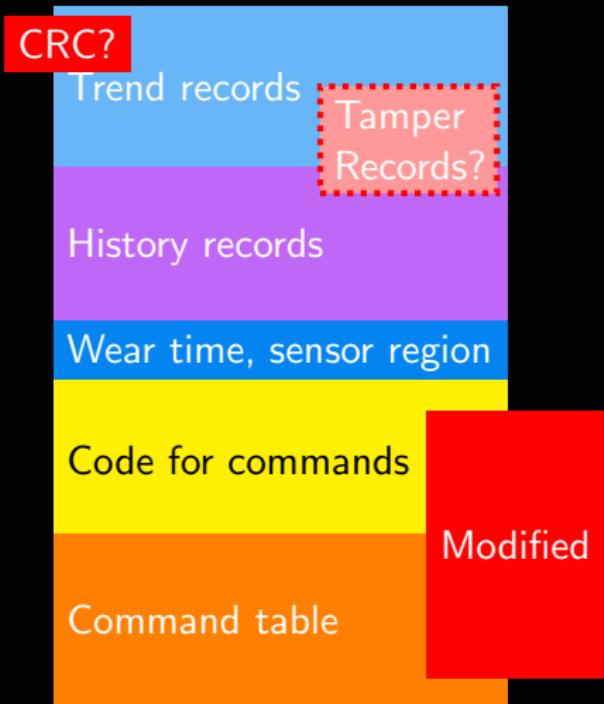


...



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

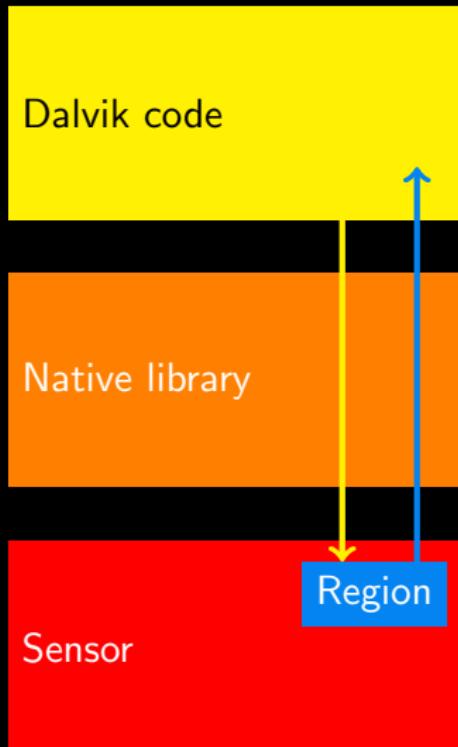
Lock/Unlock vuln

④ Sensor expiration date: how does it work?

⑤ Conclusion



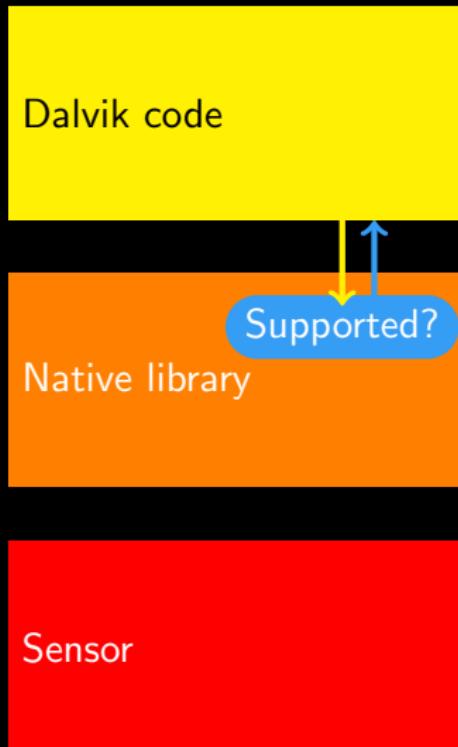
Expiration date: the 14-day limit



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



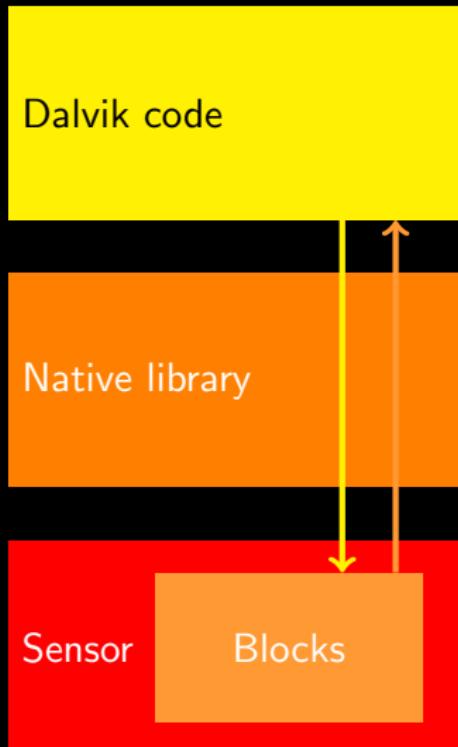
Expiration date: the 14-day limit



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



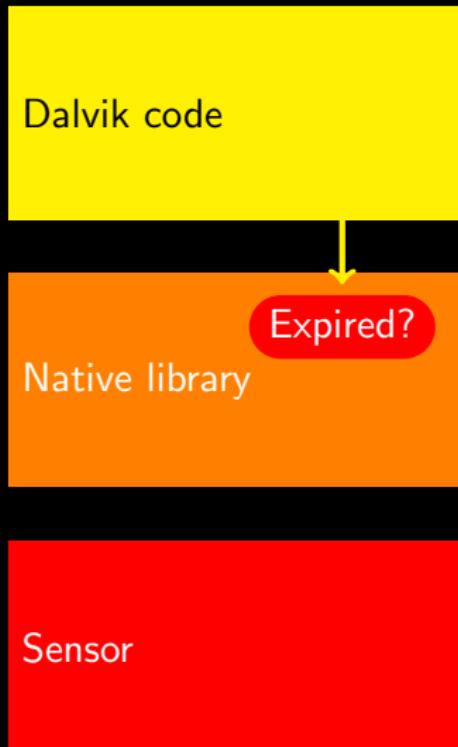
Expiration date: the 14-day limit



- 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

Dalvik code

Alarm

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

Dalvik code

Real Blocks

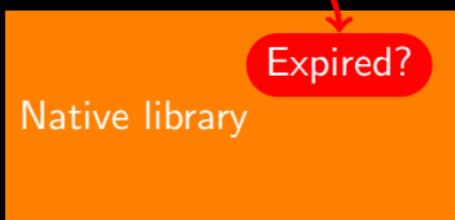
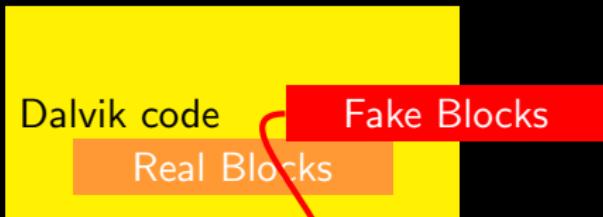
Native library

Sensor

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



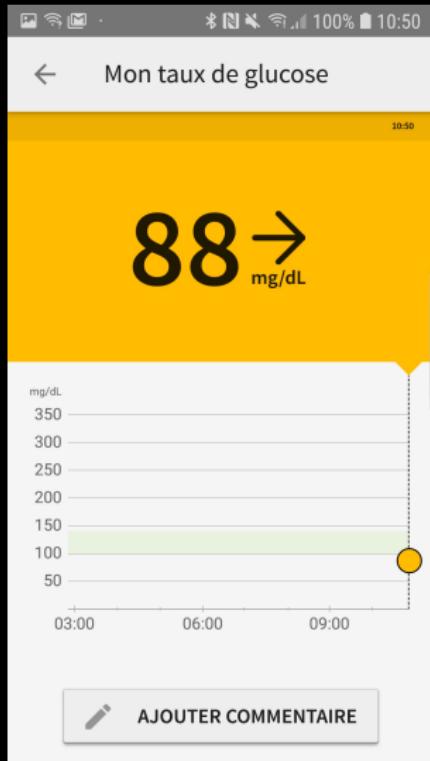
Hook expiration check



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



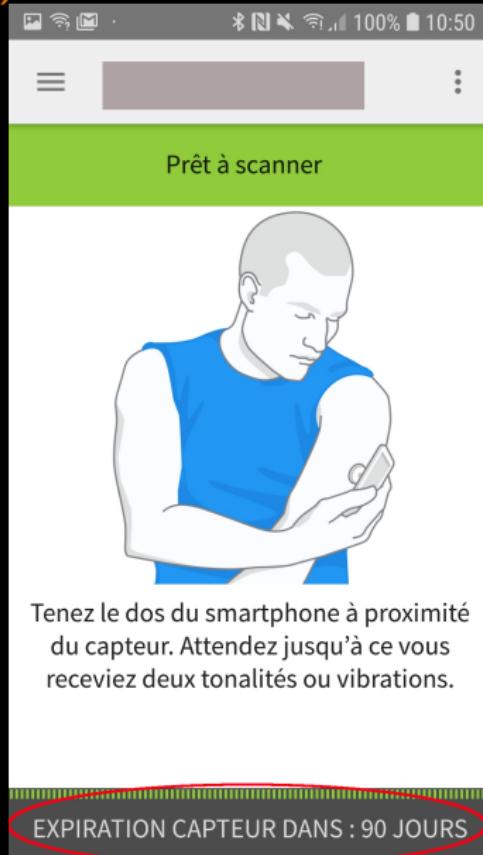
Frida hook demo



```
[*] 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?! :)



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

