riscure

hardwear.io
Hardware Security Conference and Training

Divide & Conquer revisited: FI as a SW EXP primitive

Federico Menarini
Rafael Boix Carpi

Outline

- Intro
  - Example case study: STM32WB55 chip
  - Requirements for a SW exploit
- FI as a SW attack primitive
  - FI: yearly reminder/disclaimer & quick recap
  - What does FI do from a SW point of view?
  - Problems/limitations of complex SW or FI attacks
  - Divide and conquer: stretching the power of ‘simple’ attacks
- Wrap-up, mitigations & conclusions

Intro to case study

Case study: intro

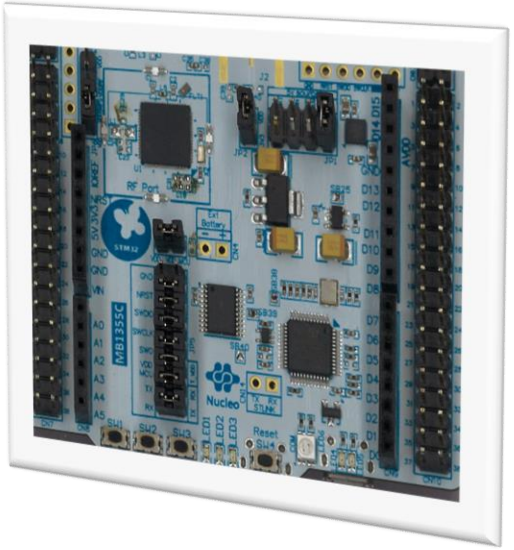
ST STM32WB55: low cost IOT chip with a ton of features

Dual core

- ARM Cortex M4 @ 64MHz + Cortex M0 @ 32 MHz
- M4: General purpose core (non-secure domain)
- M0: Communication + security (secure domain)

Supports Bluetooth and Zigbee

Security settings stored in “option bytes” in flash



Research goals

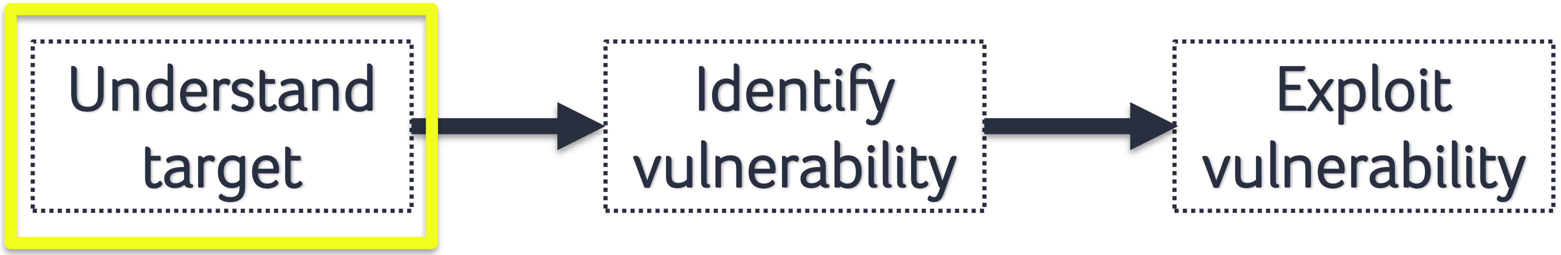
Our research goal targeted the typical assets in the following scenario:

“If we start from a fully locked chip, can we get anything from it?”

Assets we planned to target:

- **Secure core (M0 core)**
  - Runtime control? Obtain debug privileges?
  - Arbitrary code execution?
  - Dump secret AES crypto keys / secure area / wireless firm / ...?
- **Non-secure core (M4 core)**
  - Dump non-secure flash contents?
  - Bypass debug lockout?

Usual process for breaking embedded systems



To understand the target, we need the **firmware** and **datasheets**

Datasheet study – flash security

From the chip datasheet:

RDP level	Value	Behavior
RDP 0	0xAA	Full access
RDP 1	Not 0xAA or 0xCC	Can read part of RAM + registers, but no FLASH access or single stepping
RDP 2	0xCC	No access - debug locked (irreversible state)

RDP2 is final – no downgrade possible

RDP1 can be reprogrammed to RDP0 after erasing the flash

Debug access to M0 always blocked, even with RDP 0

Datasheet study – flash security

SRAM and Flash partitioned between M0 and M4

- M0 can always access everything
- But M4 cannot access M0 resources

Secure core firmware / code execution only reserved to ST

Images are encrypted and signed

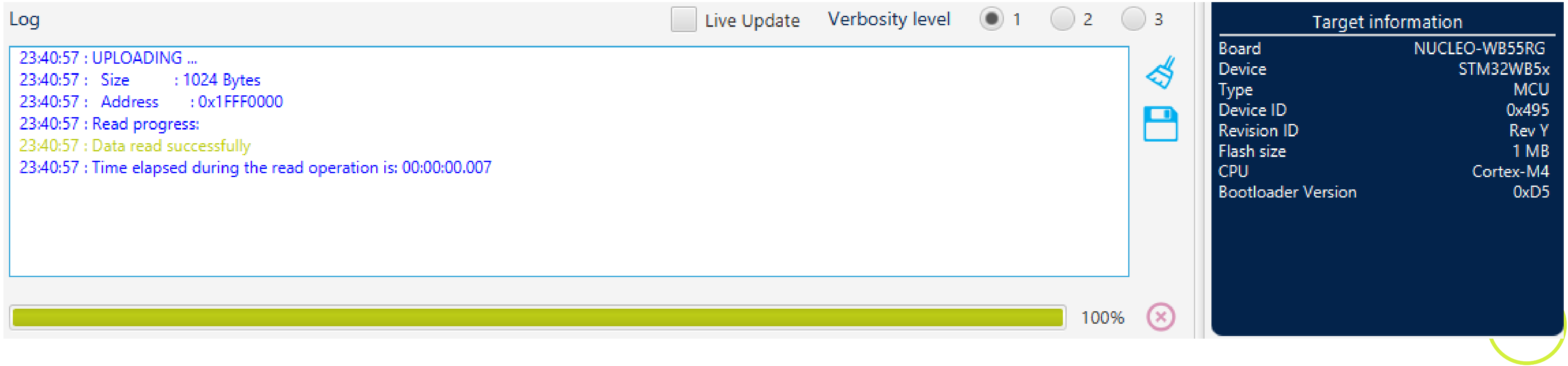
- Programmed through a loader running in M4+M0

Target overview – secure core (M0)

- Security can be configured through secure option bytes
  - Locked down in production devices
  - Secure option bytes only writable by the M0
- Most code does wireless comms (according to datasheet)
- M0 core also in charge of security, e.g. secure key storage
  - Keys cannot be deleted/modified
  - Command to load keyslot directly into AES engine
    - M4 core can then use the engine without seeing the keys

Dumping the boot ROM

- According to datasheet, Boot ROM mapped at 0x1FFF0000
- Performed simple attack to read it out:
  - Connect JTAG and read memory at 0x1FFF0000



```
23:40:57 : UPLOADING ...
23:40:57 :   Size        : 1024 Bytes
23:40:57 :   Address     : 0x1FFF0000
23:40:57 : Read progress:
23:40:57 : Data read successfully
23:40:57 : Time elapsed during the read operation is: 00:00:00.007
```

Target information

Board	NUCLEO-WB55RG
Device	STM32WB5x
Type	MCU
Device ID	0x495
Revision ID	Rev Y
Flash size	1 MB
CPU	Cortex-M4
Bootloader Version	0xD5

Quick analysis of the boot ROM

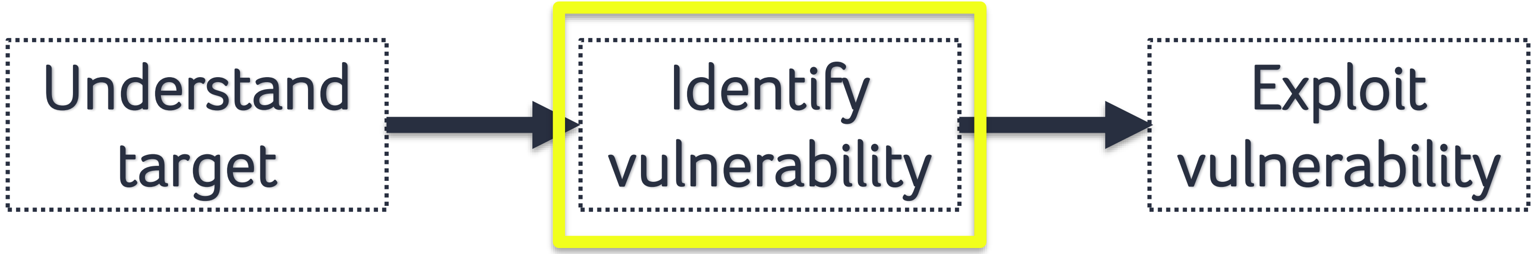
- Behavior seems to match public documents
- No hidden functionality
- No obvious logical flaws
- No check for RDP values
  - Probably checked in hardware at boot

BootROM looks simple yet quite robust…

Analyzing the non-secure core

- Cannot access secure domain via non-secure peripherals
- Non-secure core (M4) can interact with secure domain via a mailbox system
- Wireless stack is executed by the secure core
  - But we don't have its firmware
  - ST firmware updates are encrypted
    - No RE possible
    - No debug access to the secure core memory
    - Security domain HW seems properly isolated

Usual process for breaking embedded systems



Let’s see our options for exploiting something…

If we want to exploit SW vulns, we need…

1. A way to **obtain runtime control**
   - But we don’t want to do blind exploitation (M0 core is hidden to us)

2. A **predictable place** to put code
   - But M4 core cannot modify the M0 memory <u>and the chip will be locked</u>

3. A place in memory that is **executable**
   - Flash memory is executable
   - But only shared interface is a mailbox to send instructions to the M0

Possible SW attack path for different assets

Fully locked chip: no code in user area via debug access

“Fuzzing, ROP and blind exploits have no secrets for me”

- Typical approach: find a vulnerability in the wireless stack
- **Jedi master level:** blind exploitation of the vulnerability (if it is exploitable)
- **Longer way:**
  - Leak info through some vuln
  - Partially RE the wireless stack behavior and find more vulns
  - Craft an exploit chain (where do we put the exploit? ROP?)
  - Trigger the whole process and pray it works
- **PROFIT!?**

Possible FI attack path for different assets

"Glitch-your-way-through-everything!"

1 - Glitch the RDP level in order to unlock chip & get debug access

- Load some payload into the user memory & pointers to the payload

2 - Glitch a pointer to the payload into the PC of the secure core M0

- Glitching values into PC → classic FI attack on ARM 32bit – check out: https://www.riscure.com/uploads/2017/09/Controlling-PC-on-ARM-using-Fault-Injection.pdf

PROFIT!?

Fault Injection as a Software attack primitive

Glitching 101

A **glitch** is an event that leads to data/flow corruption

- Usually caused by a physical hardware fault
- Can be even triggered only by software under certain conditions (e.g. PlunderVolt)

Glitching == "physical fuzzing"

img src: https://i.redd.it/vb2y4mav2to11.jpg

riscure

What does FI do from a SW point of view

FI can corrupt almost any data value; some examples:

- Any register in the CPU (Program Counter is a critical one)
- Configuration registers (e.g. copy of OTP configuration bytes)
- In registers, during data transfers, in volatile or non-volatile memory…

FI corrupts the hardware itself

- Things that are unreachable by software can be corrupted by FI
- "Magical behavior" can happen: instruction corruption, skipping, …
- Given the proper conditions: FI attacks introduce new SW vulns!

Issues with the two different approaches

“Glitch-your-way-through-everything!”
Approach requires two glitches
We are implicitly assuming that the hardware is glitchable: *what if it isn’t?*
- What if e.g. we cannot glitch stuff into the M0 core?

“Fuzzing, ROP and blind exploits have no secrets for me”
We cannot load code from the user area due to the chip lock status
Attack path only works if there are exploitable vulnerabilities
Vulnerability discovery & exploitation is going to be far from trivial
An OTA SW update of the wireless stack will break the SW exploit

What we know about glitching and SW exp...

Performing a single glitch is usually easy...

- <u>But multiple glitches is typically difficult</u>
- Especially with non-time-constant software running between glitches

Exploiting several vulnerabilities is difficult...

- But a <u>single, simple exploit is often easy</u>

We painted a non-trivial scenario for SW exploitation:

What if we could make our life easier by changing the scenario...

... by using a single glitch...

...by using a simple exploit...

...and combining both?

SOME RULES CAN BE ~~BENT~~
EXPLOITED
GLITCHED
OTHERS CAN BE ~~BROKEN~~
memegenerator.net

Divide and conquer: revisited

If we can divide a *complex* FI or SW attack chain into a *simple* FI attack that enables a *simpler* SW exploiting scenario, we will more easily conquer our goal

Ask yourself as an attacker:

which SW exploitation challenge can we alter/modify/remove with a glitch?

riscure

First things first: same old disclaimer **again**

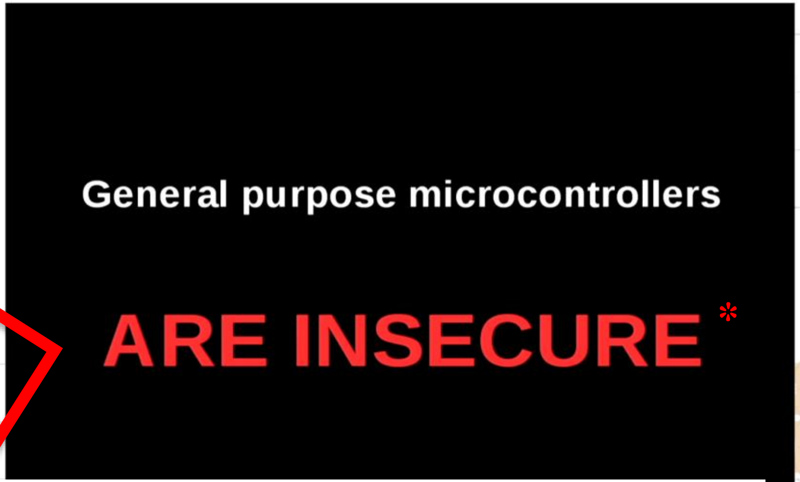
We will see combined FI attacks on a specific MCU model...

...but FI is a problem for general purpose chips **from all vendors**

FYI: Some vendors have FI-resistant chip series

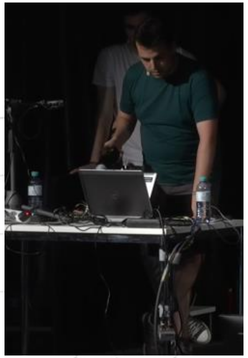
This message is not new: already in 2015…

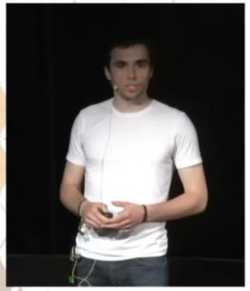
*against SCA&FI attacks



General purpose microcontrollers

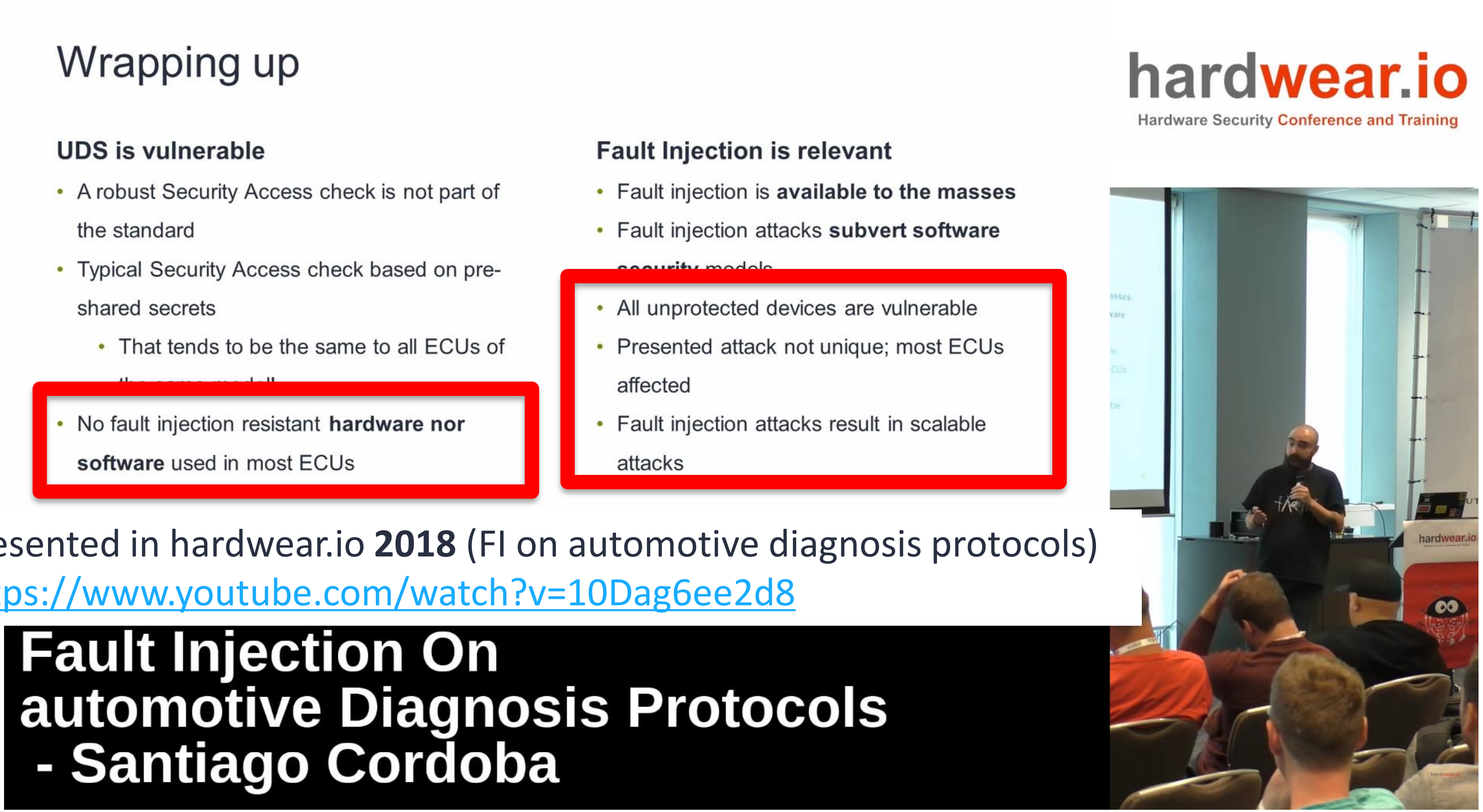
ARE INSECURE *





Presented in CCCcamp **2015** (HW attacks: hacking chips on the (very) cheap)
https://media.ccc.de/v/camp2015-6711-hardware_attacks_hacking_chips_on_the_very_cheap

Mentioned several times here also…



Wrapping up

UDS is vulnerable

- A robust Security Access check is not part of the standard
- Typical Security Access check based on pre-shared secrets
  - That tends to be the same to all ECUs of the same model!
- No fault injection resistant **hardware nor software** used in most ECUs

Fault Injection is relevant

- Fault injection is **available to the masses**
- Fault injection attacks **subvert software security models**
- All unprotected devices are vulnerable
- Presented attack not unique; most ECUs affected
- Fault injection attacks result in scalable attacks

Fault Injection On automotive Diagnosis Protocols - Santiago Cordoba

Presented in hardwear.io **2018** (FI on automotive diagnosis protocols)
https://www.youtube.com/watch?v=10Dag6ee2d8

Divide and conquer attack

Let’s glitch the rules with a combined attack

Most annoying SW exploit restriction: cannot load code or debug

Step 1: glitch the chip to unlock it
Let’s spam a single glitch and hope it unlocks the chip

Step 2: load code from non-secure domain by exploiting the secure/non-secure interface
We can load exploit code into the non-secure domain if step 1 succeeds
Let’s try to abuse the secure/non-secure interface

Step 1: let's try unlocking the chip with FI

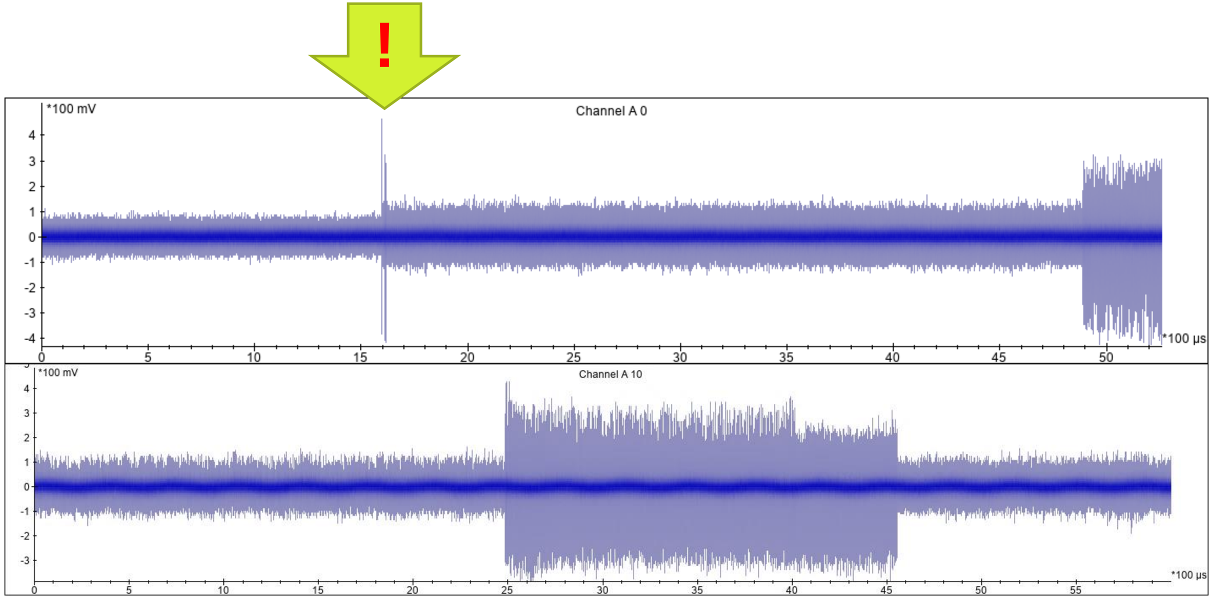
- Lockdown level setting in configuration bytes register
  - 0xAA: unlocked
  - 0xCC: fully locked (JTAG disabled, irreversible)
  - Anything else: partial locked
- FI plan:
  - Corrupting configuration bytes → partial lock → enables JTAG again

  1. Force reload of configuration bytes on open sample: look for interesting SCA pattern
  2. Glitch option bytes register while being reloaded
  3. If previous attempt works: repeat glitch at boot time (after powerup)

FI profiling

Powerup reset: spike

Warm reset: no spike



- Readout of option bytes is only done after powerup according to datasheet
- The same pattern is found if you trigger a reload by software

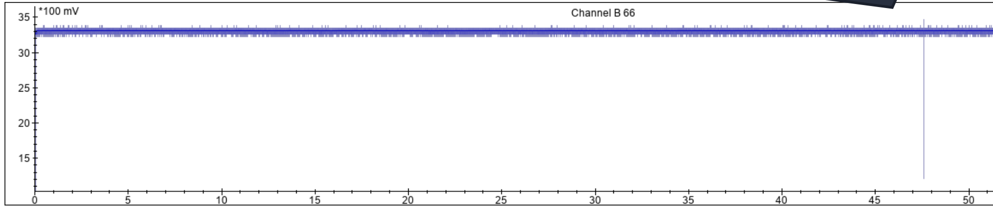
→This is our FI point

FI attack: attempt 1

VCC glitching, nominal VCC

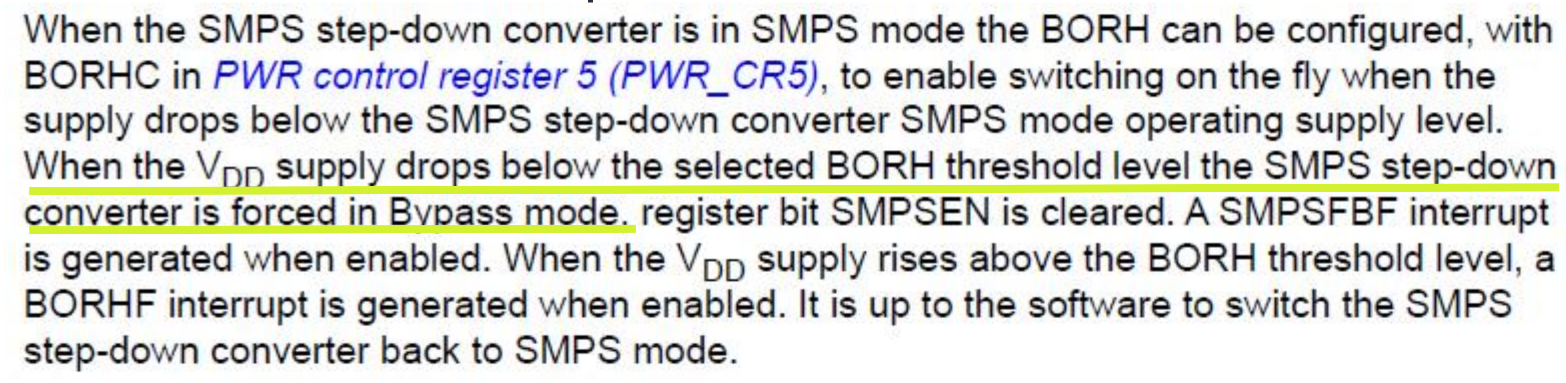
- Removed decoupling capacitors
- Several FI profiling rounds
- Single glitch up to (-4V, 1000ns)
- The glitch is *very* visible in the power consumption traces

Result: No glitching *at all* due to internal filtering by on-die power supply



Developer recommendation: use the SMPS→Voltage FI attacks harder!

FI attack: attempt 2


When the SMPS step-down converter is in SMPS mode the BORH can be configured, with BORHC in *PWR control register 5 (PWR_CR5)*, to enable switching on the fly when the supply drops below the SMPS step-down converter SMPS mode operating supply level. When the V_{DD} supply drops below the selected BORH threshold level the SMPS step-down converter is forced in Bypass mode, register bit SMPSEN is cleared. A SMPSFBF interrupt is generated when enabled. When the V_{DD} supply rises above the BORH threshold level, a BORHF interrupt is generated when enabled. It is up to the software to switch the SMPS step-down converter back to SMPS mode.

Attempt 2: VCC glitching with very low VCC supply voltage

- Result: Glitches

Attempt 3: VCC glitching with hardware modification to bypass on-die PSU

- Result: More glitches

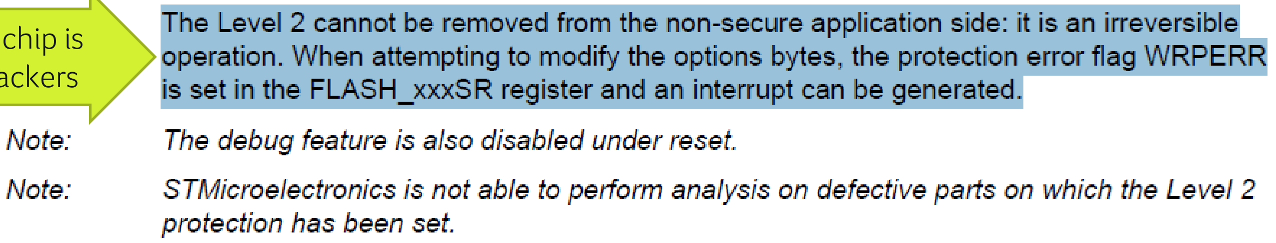
Attempt 4: EMFI with no target modification (just to try other FI method)

- Result: More glitches

FI: attack results & conclusion

Permanent downgrade possible from full lockdown to partial lock/unlocked with FI by glitching RDP to 0xFF

*except if the chip is against FI attackers



The Level 2 cannot be removed from the non-secure application side: it is an irreversible operation. When attempting to modify the options bytes, the protection error flag WRPERR is set in the FLASH_xxxSR register and an interrupt can be generated.

Note: The debug feature is also disabled under reset.

Note: STMicroelectronics is not able to perform analysis on defective parts on which the Level 2 protection has been set.

img from STM32WB55 datasheet

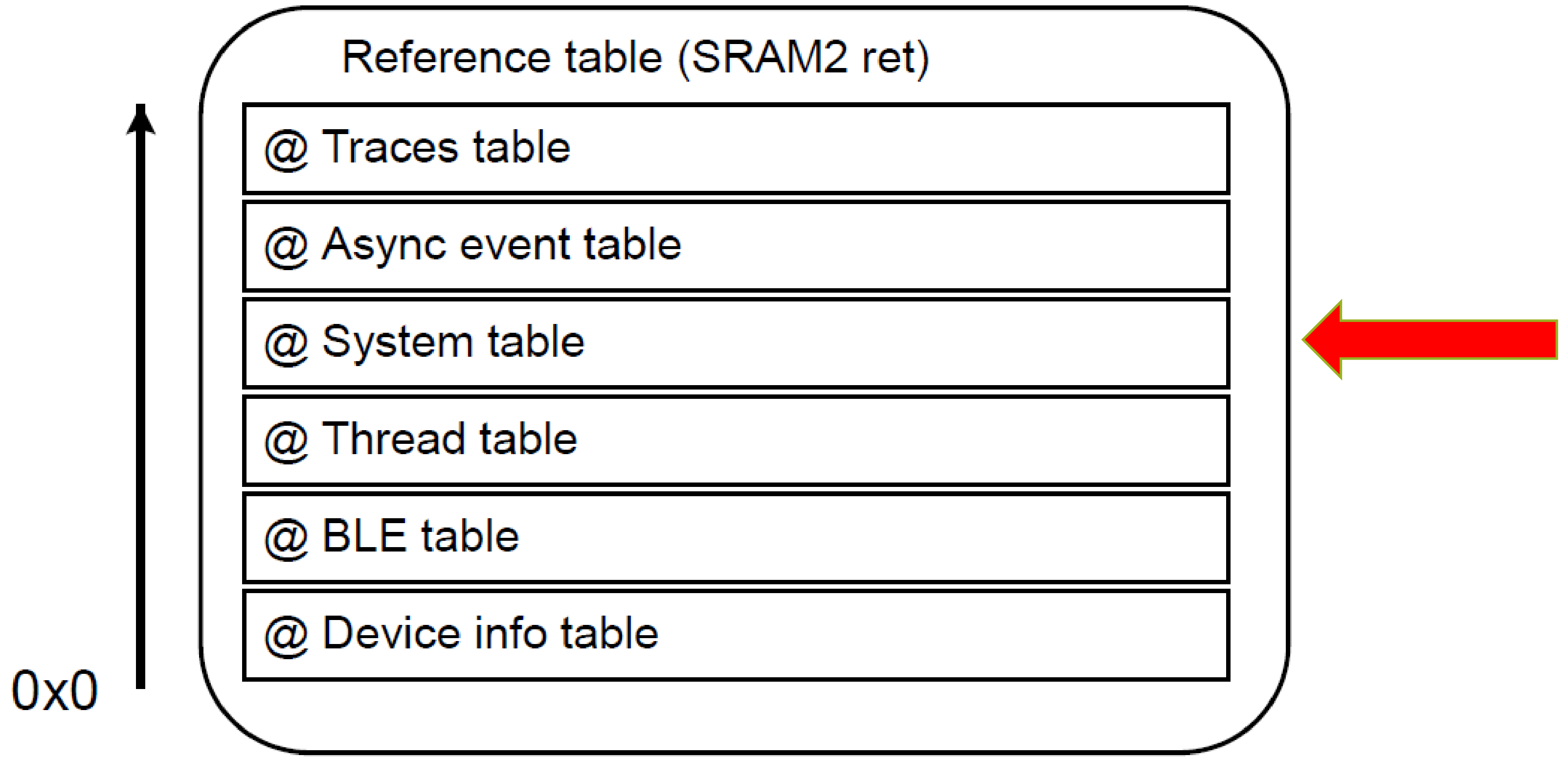
FI JTAG unlock: common MCU problem for chip vendors. However…

- RDP can then be set to lvl 0 – non-secure flash will be erased
- The secure core domain is still intact, but we can now run arbitrary code from non-secure FLASH because chip is unlocked → time for our SW exploit

Step 2- Abusing secure domain interfaces

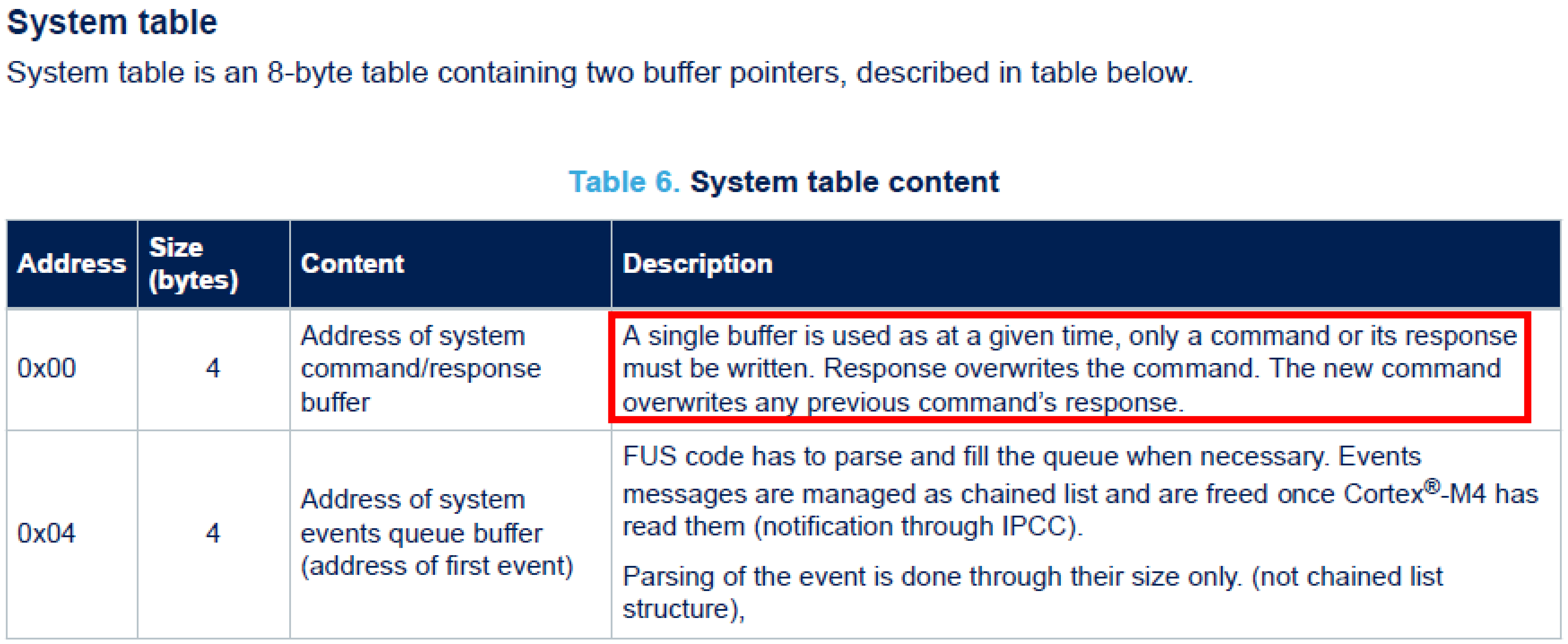
- Message passing between secure and non-secure domains is handled through a mailbox system (IPCC)
- Shared buffer in SRAM, offset in option bytes
- Messages contain a structure with pointers to certain fields
  - Secure core will parse input messages and process them (wireless commands, flash update, …)
  - Secure core will write the answer to memory pointed by those pointers
  - Flash update firmware always present → focus on it

Secure core – interface details



img from
STM32WB55
documentation

Secure core – interface details



System table

System table is an 8-byte table containing two buffer pointers, described in table below.

Table 6. **System table content**

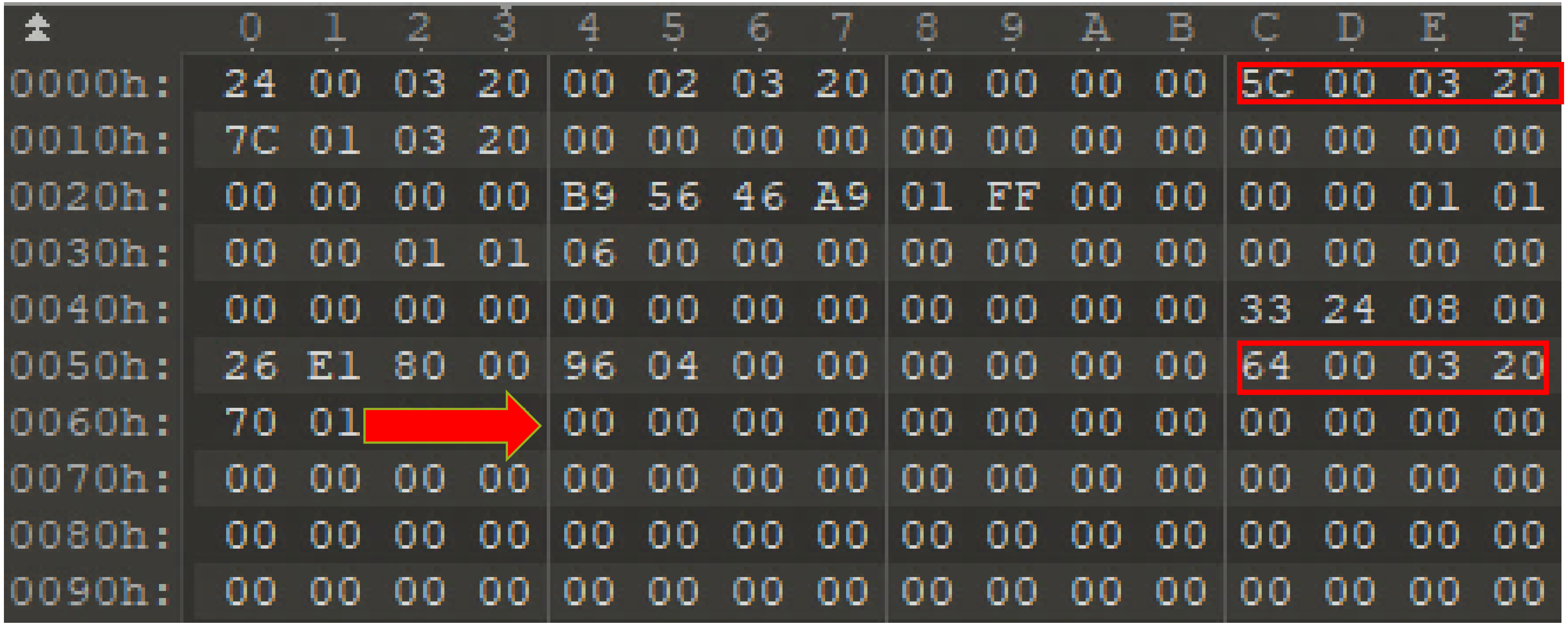
Address	Size (bytes)	Content	Description
0x00	4	Address of system command/response buffer	A single buffer is used as at a given time, only a command or its response must be written. Response overwrites the command. The new command overwrites any previous command's response.
0x04	4	Address of system events queue buffer (address of first event)	FUS code has to parse and fill the queue when necessary. Events messages are managed as chained list and are freed once Cortex®-M4 has read them (notification through IPCC). Parsing of the event is done through their size only. (not chained list structure),

img from STM32WB55 documentation

Secure core – interface details

- ROM code interfaces with the secure core
  - Used for secure core FW upgrade
- Tables must have been already set up…
- Just read them out

Secure core – interface details



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000h: 24 00 03 20 00 02 03 20 00 00 00 00 5C 00 03 20
0010h: 7C 01 03 20 00 00 00 00 00 00 00 00 00 00 00 00
0020h: 00 00 00 00 B9 56 46 A9 01 FF 00 00 00 00 01 01
0030h: 00 00 01 01 06 00 00 00 00 00 00 00 00 00 00 00
0040h: 00 00 00 00 00 00 00 00 00 00 00 00 33 24 08 00
0050h: 26 E1 80 00 96 04 00 00 00 00 00 00 64 00 03 20
0060h: 70 01       00 00 00 00 00 00 00 00 00 00 00 00
0070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0080h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0090h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Secure core – possible vulnerability

- Two pointers to be dereferenced
- Is it done right?
    - Both pointers should point to shared SRAM
    - The check should be performed both when reading and writing

Secure core – interface attack plan

- Try a (sort of) TOCTOU attack
- Set up a correct table by looking at what the BootROM does
- Send a command
- Wait some time, change the table to make it point to other memory locations
- Have the secure core write to its internal memory
- ???
- Profit

Secure core – testing attack hypothesis

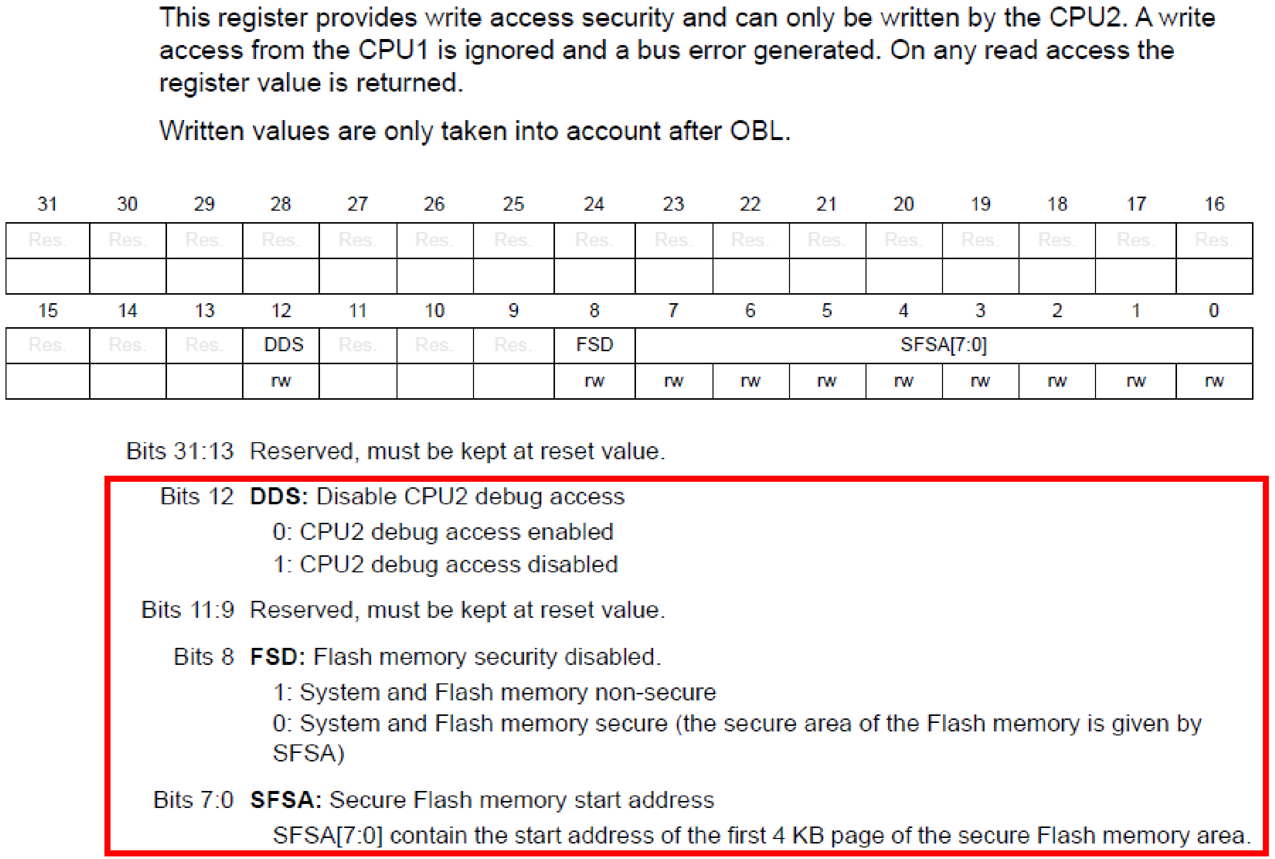
- Tried the approach: it seems to work!
- What do we want to overwrite?
  - And with what?

- Possible sets of answers seems very limited:
  - Fixed header + ack of command ID + 1 status byte (always 5 bytes in total)
  - We still don’t have access to the M0 FW
  - Only flash is executable
  - Very difficult primitive to use blindly

Secure core – attack revisited

- Couldn’t come up with anything smart for actual attack
- Let’s try to be simple: just point the response buffer onto the secure option bytes
    - Secure option bytes are memory mapped
    - Hopefully they will be overwritten with whatever the M0 writes
    - Response copied in the buffer byte wise
    - Odd behavior when writing to registers
    - Maybe we are lucky…

SECURE OPTION BYTES



This register provides write access security and can only be written by the CPU2. A write access from the CPU1 is ignored and a bus error generated. On any read access the register value is returned.

Written values are only taken into account after OBL.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	DDS	Res.	Res.	Res.	FSD	SFSA[7:0]							
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:13 Reserved, must be kept at reset value.

Bits 12 **DDS:** Disable CPU2 debug access
- 0: CPU2 debug access enabled
- 1: CPU2 debug access disabled

Bits 11:9 Reserved, must be kept at reset value.

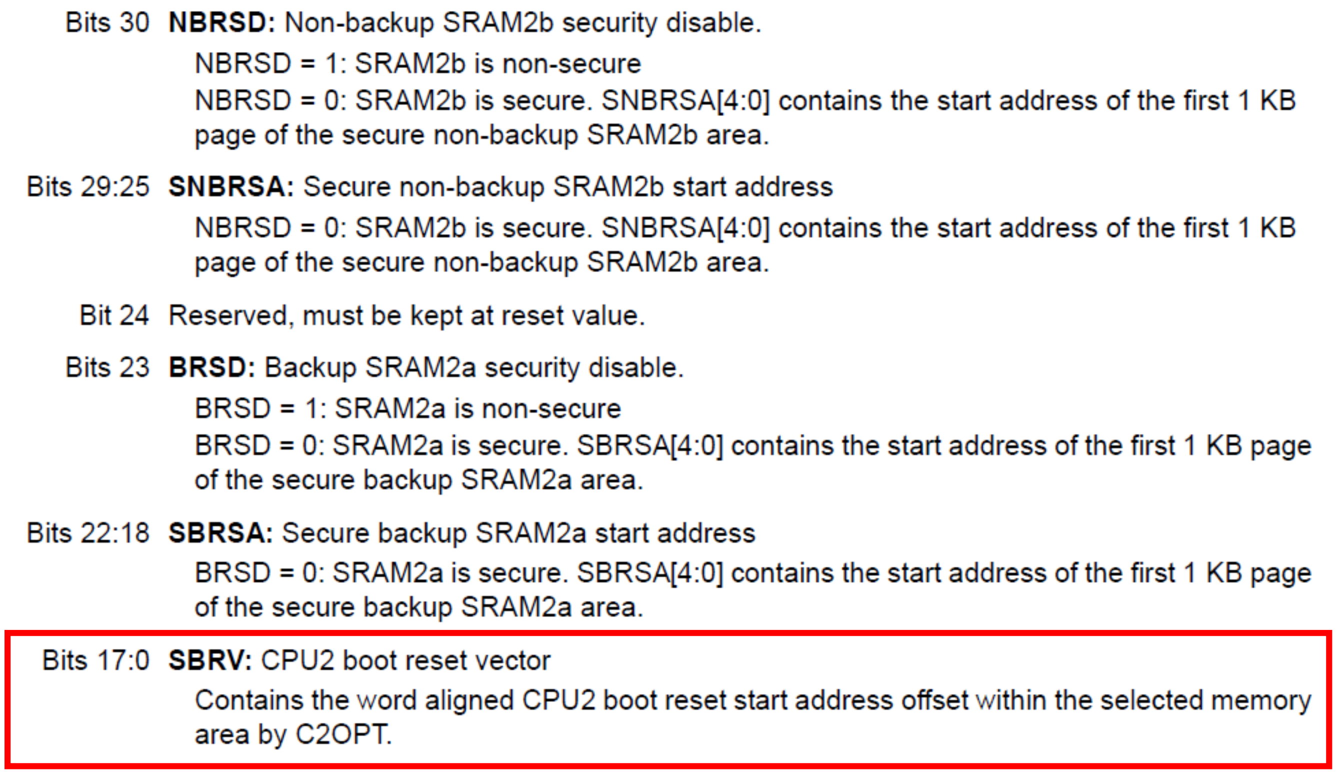
Bits 8 **FSD:** Flash memory security disabled.
- 1: System and Flash memory non-secure
- 0: System and Flash memory secure (the secure area of the Flash memory is given by SFSA)

Bits 7:0 **SFSA:** Secure Flash memory start address

SFSA[7:0] contain the start address of the first 4 KB page of the secure Flash memory area.

img from STM32WB55 documentation

SECURE OPTION BYTES



Bits 30 **NBRSD:** Non-backup SRAM2b security disable.

NBRSD = 1: SRAM2b is non-secure
NBRSD = 0: SRAM2b is secure. SNBRSA[4:0] contains the start address of the first 1 KB page of the secure non-backup SRAM2b area.

Bits 29:25 **SNBRSA:** Secure non-backup SRAM2b start address

NBRSD = 0: SRAM2b is secure. SNBRSA[4:0] contains the start address of the first 1 KB page of the secure non-backup SRAM2b area.

Bit 24 Reserved, must be kept at reset value.

Bits 23 **BRSD:** Backup SRAM2a security disable.

BRSD = 1: SRAM2a is non-secure
BRSD = 0: SRAM2a is secure. SBRSA[4:0] contains the start address of the first 1 KB page of the secure backup SRAM2a area.

Bits 22:18 **SBRSA:** Secure backup SRAM2a start address

BRSD = 0: SRAM2a is secure. SBRSA[4:0] contains the start address of the first 1 KB page of the secure backup SRAM2a area.

Bits 17:0 **SBRV:** CPU2 boot reset vector

Contains the word aligned CPU2 boot reset start address offset within the selected memory area by C2OPT.

img from STM32WB55 documentation

Running the attack

Secure core – full attack

- We can make the secure core overwrite its configuration bytes
  - Namely, its start address
  - After running the exploit, the start address points to non-secure flash
- Reprogram the start address with a small shellcode that disables all security
- We can read secure flash!
  - Both from non-secure core and with JTAG

Secure core – attack results

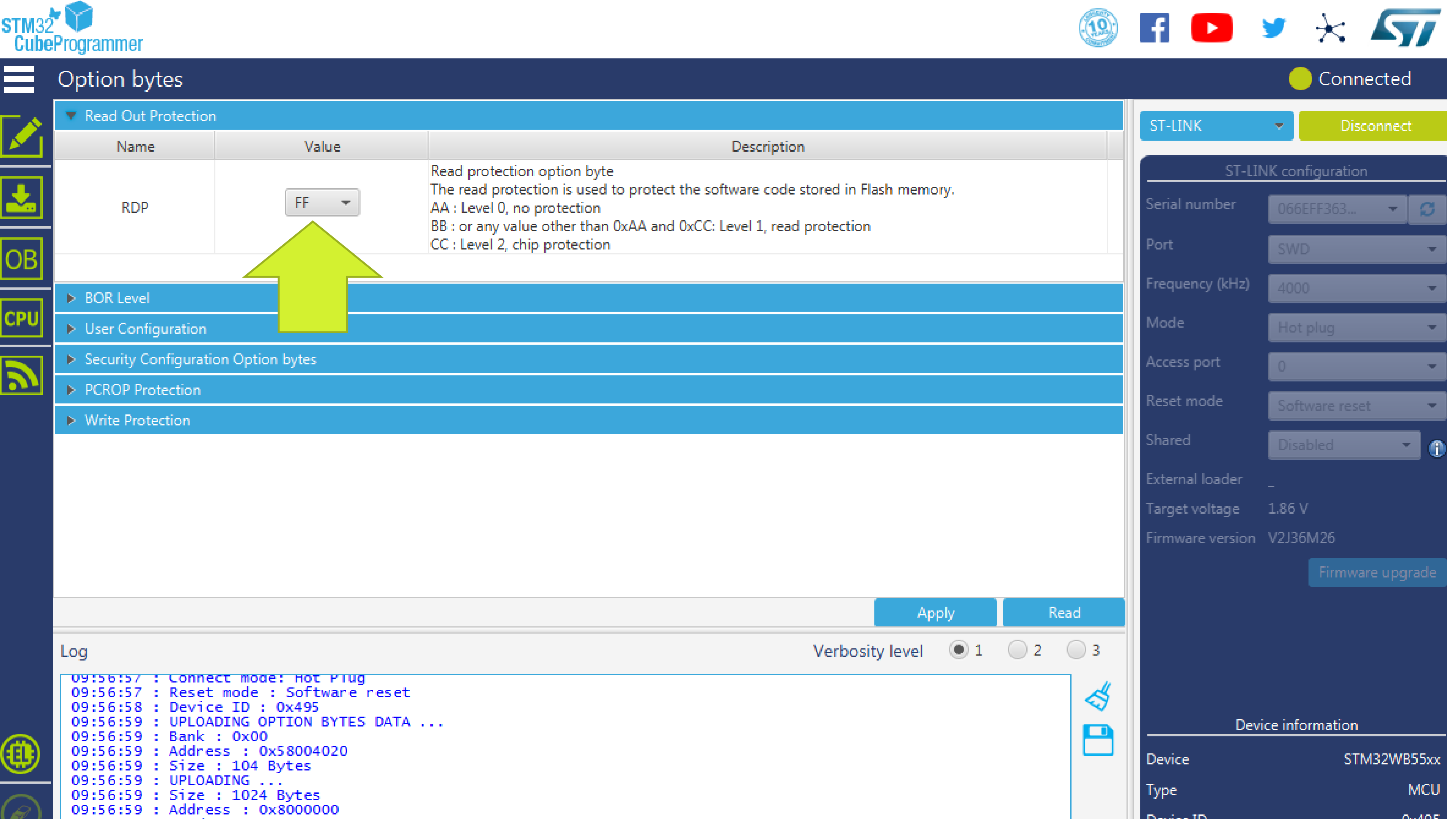
- Read out (everything)
- We have JTAG access to the Secure Core
    - M0 core has access to everything in the system

Divide and conquer: full FI + SW exp attack

- Get a device fully locked
- Use FI to downgrade security to ‘partial lockdown’
    - Optional: read out SRAM contents at runtime via JTAG in ‘partial lockdown’
- Erase the flash & go to unlocked mode & load a SW exploit in non-secure memory
- Run a SW exploit from non-secure domain in order to unprotect secure core domain
- Read out (everything)

DEMO TIME

Demo of the attack



Name	Value	Description
RDP	FF	Read protection option byte The read protection is used to protect the software code stored in Flash memory. AA : Level 0, no protection BB : or any value other than 0xAA and 0xCC: Level 1, read protection CC : Level 2, chip protection

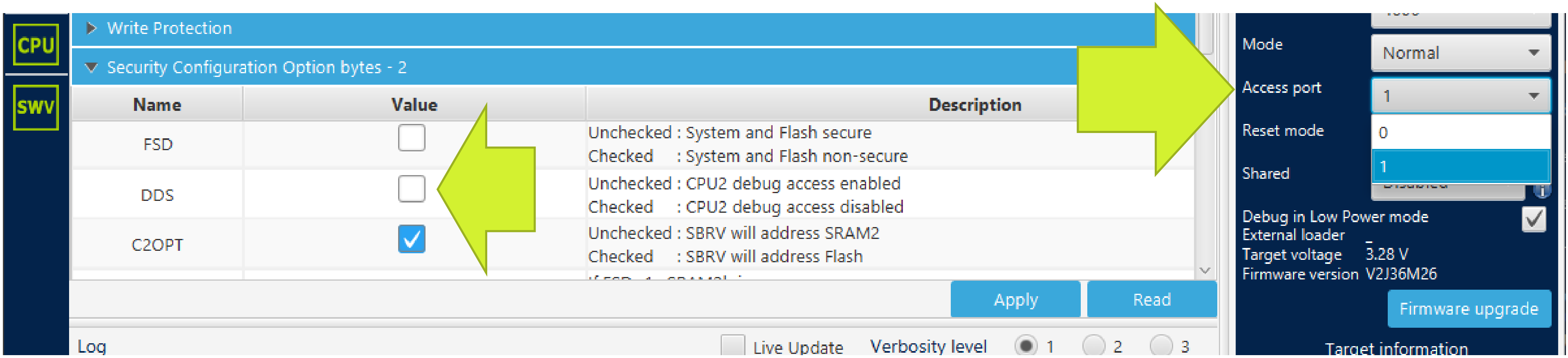
Serial number	066EFF363...
Port	SWD
Frequency (kHz)	4000
Mode	Hot plug
Access port	0
Reset mode	Software reset
Shared	Disabled
External loader	_
Target voltage	1.86 V
Firmware version	V2J36M26

Device	STM32WB55xx
Type	MCU

Log — Verbosity level [✓ 1] 2 3

```
09:56:57 : Connect mode: Hot Plug
09:56:57 : Reset mode : Software reset
09:56:58 : Device ID : 0x495
09:56:59 : UPLOADING OPTION BYTES DATA ...
09:56:59 : Bank : 0x00
09:56:59 : Address : 0x58004020
09:56:59 : Size : 104 Bytes
09:56:59 : UPLOADING ...
09:56:59 : Size : 1024 Bytes
09:56:59 : Address : 0x8000000
```

DEMO SW EXPLOIT: SECURE CORE DEBUG ENABLED



Name	Value	Description
FSD	☐	Unchecked : System and Flash secure Checked : System and Flash non-secure
DDS	☐	Unchecked : CPU2 debug access enabled Checked : CPU2 debug access disabled
C2OPT	✓	Unchecked : SBRV will address SRAM2 Checked : SBRV will address Flash

Mode	Normal
Access port	1
Reset mode	0 / 1
Shared	
Debug in Low Power mode	✓
External loader	_
Target voltage	3.28 V
Firmware version	V2J36M26

Wrap-up, mitigations and conclusions

Wrap-up

With the divide & conquer approach, we managed to:

- Glitch the non-secure domain readout protection mechanism
- Find & exploit a software vulnerability in the interface
- Achieve a full compromise of the security domain

There is a thing we didn't get due to our 'simple' approach:

- Glitching & downgrading RDP level triggers a non-secure FLASH erase
    - In other words: our 'simple' attack approach deletes the user area
    - Not so relevant for our purpose, but may delete interesting assets
    - It may be possible to avoid the downgrade step, but we didn't explore it

Mitigations

How to mitigate combined attacks: put obstacles everywhere!

Specifically in the presented case study:

- as a developer: include FI attackers in your threat model
- use the STM32WB55 SMPS whenever possible – makes VFI harder
- add secrets not only in the secure area, but also in the user area
  - this would make the non-secure FLASH erasure a problem for attackers
- **update the firmware for patching software vulnerabilities**
  - The IPCC mailbox vuln has been patched in STM32CubeWB MCU FW ≥v1.10.1
  - Link to latest version: https://github.com/STMicroelectronics/STM32CubeWB

Coordinated disclosure

We followed a Coordinated Disclosure procedure with ST PSIRT

- Sept 20, 2020: Vulnerability report + PoC exploit shared with ST PSIRT
- Oct 1, 2020: ST PSIRT reports they are investigating the issue
- Oct 15, 2020: ST PSIRT confirms the vulnerability and informs Riscure that it will be patched in the next FW release and gives an estimated timeline
- Nov 10, 2020: ST PSIRT updates Riscure on the patch release date (W04 '21)
- Feb 5, 2021: Riscure checks with ST PSIRT if released FW update contains patch
- Feb 8, 2021: ST PSIRT confirms that FW version 1.10.1 contains the patch

Kudos to ST PSIRT for addressing the issue in a professional manner

Conclusions (attacker view)

Dividing a complex FI or SW exploitation scenario into a combined FI+SW attack can be a viable attack path

A simple FI attack can change the SW exploit scenario

- FI resilience is hard
- Check if MCU threat model considers FI attacks
- If no FI protection: divide & conquer!

Combined FI/SW attacks can be simple yet very powerful

- More and more often found in infosec news

Conclusions (developer view)

FI is still getting more and more popular

Add obstacles for attackers in every step:

- Include FI attackers in your threat model
- Patch firmwares: this mitigates the impact of FI vulnerabilities
- Use all security properties/features of your platform
- Distribute secrets/dependencies so that an attacker only succeeds with a full system compromise

Riscure B.V.
Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
inforequest@riscure.com

Riscure North America
550 Kearny St., Suite 330
San Francisco, CA 94108 USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure China
Room 2030-31, No. 989, Changle Road, Shanghai 200031
China
Phone: +86 21 5117 5435
inforcn@riscure.com

Further questions/information:

Federico Menarini
Principal Security Analyst (@ffmenarini, federico@riscure.com)
Rafael Boix Carpi
Principal Security Specialist (@rafabxc , rafael@riscure.com)

www.riscure.com

riscure

driving your security forward