The Forgotten Threat of Voltage Glitching: A Case Study on Nvidia Tegra X2 SoCs

Otto Bittner[*§], Thilo Krachenfels[*§], Andreas Galauner[†], Jean-Pierre Seifert[*‡]

* Technische Universität Berlin, Chair of Security in Telecommunications
† Independent Researcher
‡ Fraunhofer SIT

§ These authors contributed equally to this work

FDTC 2021

SECT

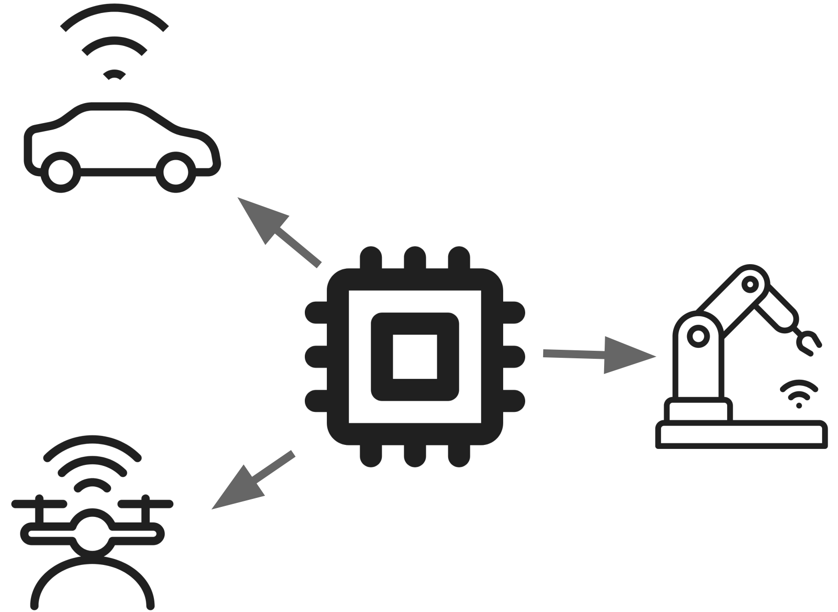
Technische Universität Berlin

Outline

1. Motivation & Threat model
2. Voltage glitching Background
3. Attack recipe
4. Attack steps and results
5. Conclusion

Motivation

- Complex SoCs used in safety-critical applications
- Physical access by an attacker often possible
- Vendors must consider device tampering

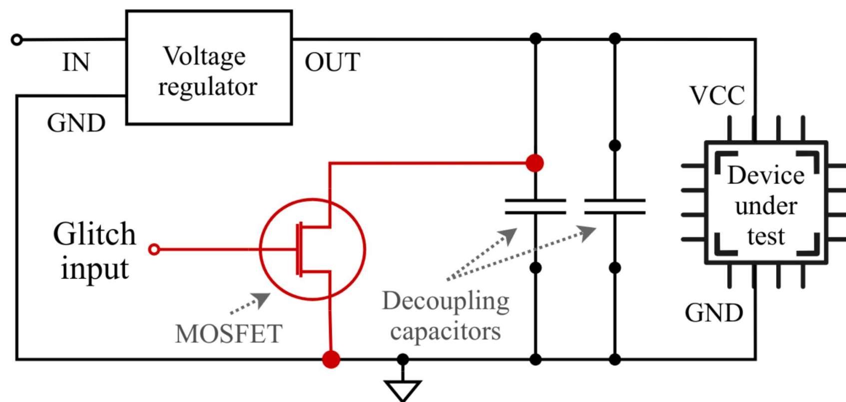


Threat Model

- Voltage fault injection (FI) around for more than 20 years
- Can be used to influence the device
  - Corrupt of data values
  - Skip security checks
  - Enter protected code paths
- Commercial SoCs are often not protected against such attacks (e.g., Nintendo Switch hack)

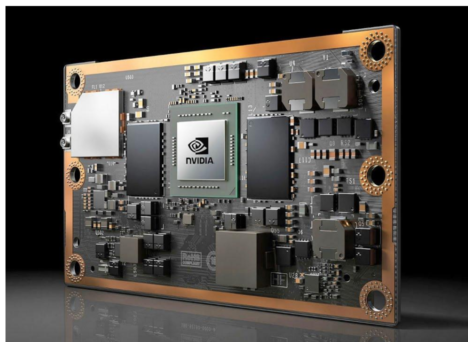
Background: Voltage Glitching

- Idea: Over- or undervolting supply voltage
- Here: Undervolting using a Crowbar circuit
  - Short circuit of voltage rail and ground
- DUT operated out of the rated supply voltage levels
- Errors in the computation occur



Attack Overview

Device Under Test



[1]

X2 SoC

Goal

Leak Read-Protected BootROM

Outcome

?

Voltage FI Recipe

Recipe for FI identified in this work:

Step 1	Determining the feasibility of FI
Step 2	Identifying the FI target and a success indicator
Step 3	Finding a trigger signal
Step 4	Finding glitch parameters
Step 5	Generating target payload

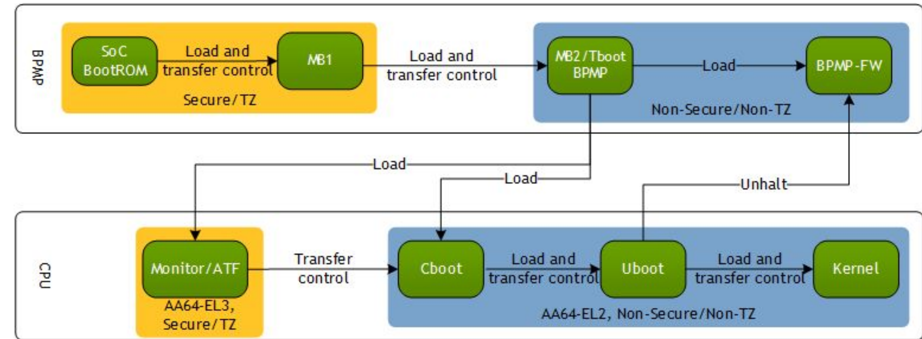
Voltage FI Recipe

Recipe for FI identified in this work:

Step 1	**Determining the feasibility of FI**
Step 2	Identifying the FI target and a success indicator
Step 3	Finding a trigger signal
Step 4	Finding glitch parameters
Step 5	Generating target payload

Step 1: FI Feasibility - Code Execution
X2 Boot Flow

- Separate boot processor: BPMP
- Two privilege levels
- BootROM → Root of trust
- MB1 → Update mechanism
- MB2 → Device initialization



[2]

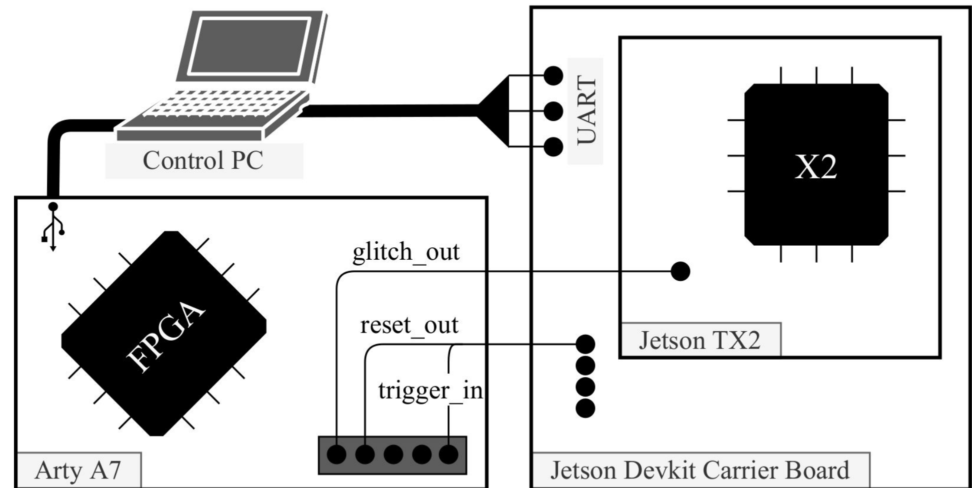
Step 1: FI Feasibility - Code Execution
Build Custom MB2

- Setup Cross Compilation Toolchain
- Find correct base address - rbasefind
- Study TRM → UART & GPIO output

Step 1: FI Feasibility - Hardware Preparation
Basic Fault Injection Setup

- Control PC → Parse X2 traffic
- FPGA → Control glitch, reset X2
- X2 → DUT

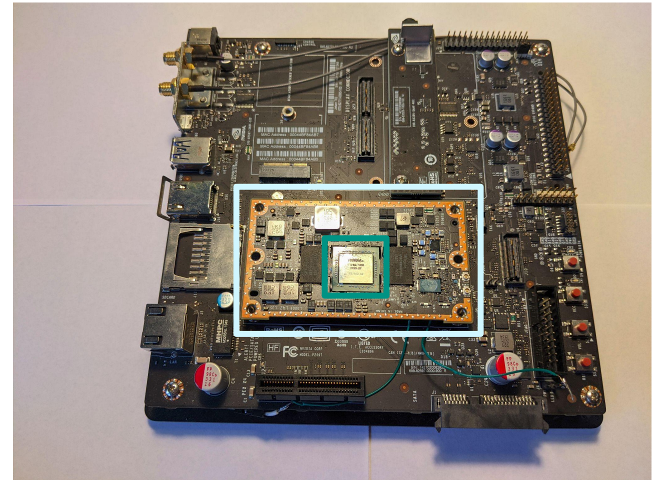
- Uses Thomas Roth's chipfail-glitcher[1]



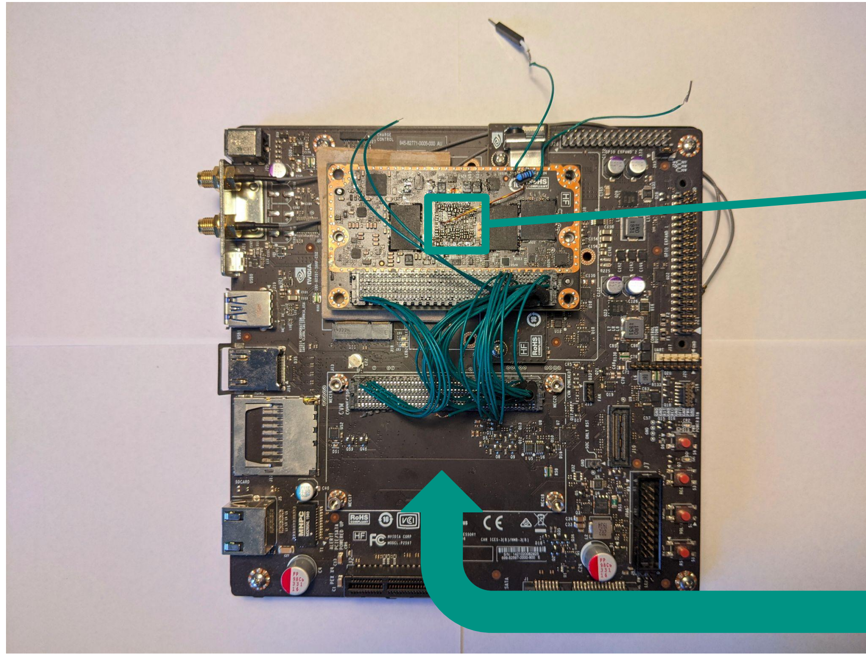
1: https://github.com/chipfail/chipfail-glitcher

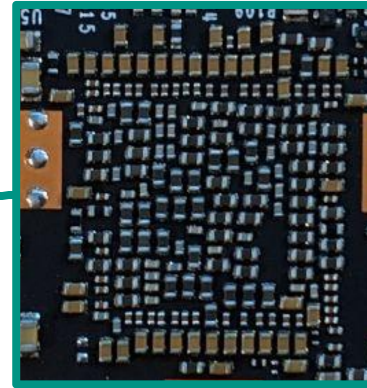
Step 1: FI Feasibility – Hardware Preparation
Crowbar Circuit

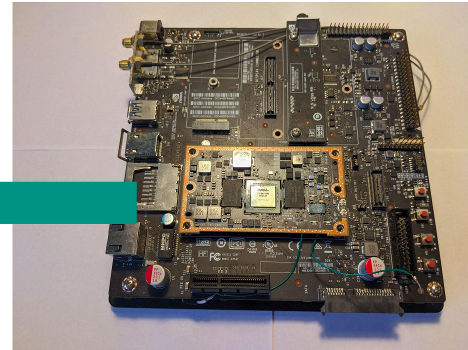
VDD
20V
onboard
voltage regulators
0.5V–5V
glitch_5v
10k
1%
4
8
1
IRF8736PbF
1
2
1
2
DUT
GND
Jetson TX Module



Step 1: FI Feasibility - Hardware Preparation
Measurement Setup







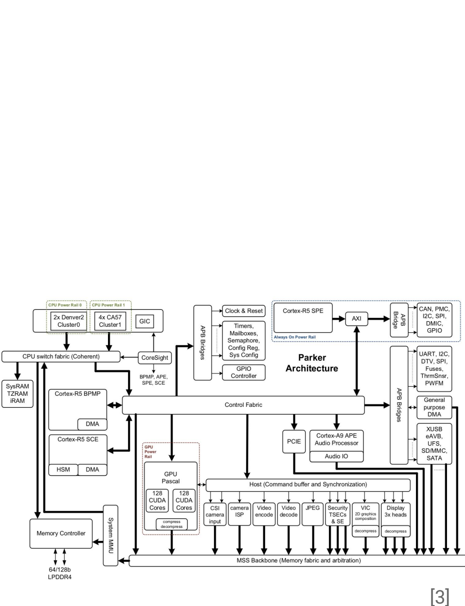
Probe these?

Step 1: FI Feasibility - Hardware Preparation
Power Domains

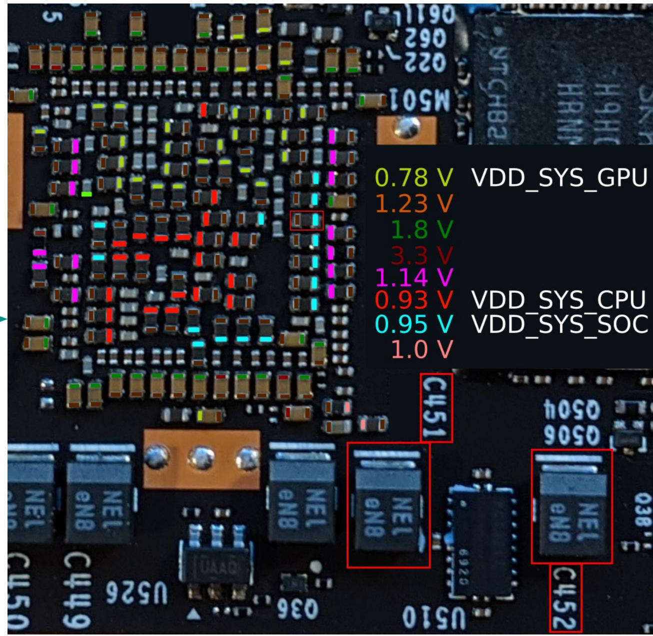
- Multiple power domains
- MB1 receives config: BCT
- BCT can change PMIC configs
- Lots of comments!

```
1   # CFG Version 1.0
2   # This is the empty CFG files for PMIC rail configuration
3   # This contains the PMIC commands in MB1.
4   #define TEGRA18x_MB1_POWER_RAIL_GENERIC 1
5   #define TEGRA18x_MB1_POWER_RAIL_CPU     2
6   #define TEGRA18x_MB1_POWER_RAIL_CORE    3
7   #define TEGRA18x_MB1_POWER_RAIL_SRAM    4
8   #define TEGRA18x_MB1_POWER_RAIL_GPU     5
9   #define TEGRA18x_MB1_POWER_RAIL_MEMIO   6
10  #define TEGRA18x_MB1_POWER_RAIL_THERMAL_CONFIG 7
11  #define TEGRA18x_MB1_POWER_RAIL_SHUTDOWN_CONFIG 8
12  #define TEGRA18x_MB1_POWER_RAIL_MAX 9
13
14  # ...
15
16  ###################### #CORE RAIL  (ID = 3) DATA ##############
17  pmic.core.3.block-count = 3;
18
19  # 1. Set 950mV voltage.
20  pmic.core.3.block[0].type = 2; # PWM Type
21  pmic.core.3.block[0].controller-id = 5; #GP_PWM6
22  pmic.core.3.block[0].source-frq-hz = 102000000; #102MHz
23  pmic.core.3.block[0].period-ns = 2600; # 384K is period.
24  pmic.core.3.block[0].min-microvolts = 710000;
25  pmic.core.3.block[0].max-microvolts = 1150000;
26  pmic.core.3.block[0].init-microvolts = 950000;
27  pmic.core.3.block[0].enable = 1;
```

Step 1: FI Feasibility – Hardware Preparation
Power Domains – Identified Voltage Rails



[3]



```
1  # CFG Version 1.0
2  # This is the empty CFG files for PMIC rail configuration
3  # This contains the PMIC commands in MB1.
4  #define TEGRA18x_MB1_POWER_RAIL_GENERIC 1
5  #define TEGRA18x_MB1_POWER_RAIL_CPU     2
6  #define TEGRA18x_MB1_POWER_RAIL_CORE    3
7  #define TEGRA18x_MB1_POWER_RAIL_SRAM    4
8  #define TEGRA18x_MB1_POWER_RAIL_GPU     5
9  #define TEGRA18x_MB1_POWER_RAIL_MEMIO   6
10 #define TEGRA18x_MB1_POWER_RAIL_THERMAL_CONFIG 7
11 #define TEGRA18x_MB1_POWER_RAIL_SHUTDOWN_CONFIG 8
12 #define TEGRA18x_MB1_POWER_RAIL_MAX 9
13
14 # ...
15
16 ###################### #CORE RAIL  (ID = 3) DATA ##############
17 pmic.core.3.block-count = 3;
18
19 # 1. Set 950mV voltage.
20 pmic.core.3.block[0].type = 2; # PWM Type
21 pmic.core.3.block[0].controller-id = 5; #GP_PWM6
22 pmic.core.3.block[0].source-frq-hz = 102000000; #102MHz
23 pmic.core.3.block[0].period-ns = 2600; # 384K is period.
24 pmic.core.3.block[0].min-microvolts = 710000;
25 pmic.core.3.block[0].max-microvolts = 1150000;
26 pmic.core.3.block[0].init-microvolts = 950000;
27 pmic.core.3.block[0].enable = 1;
```

Step 1: FI Feasibility - Proof of Concept

- Run fault-sensitive code in MB2
- Three independent counters
- Expected Result: `!100 - 100 - 10000\n`

```
1  void tightly_coupled_loops(){
2      volatile int ctr = 0;
3      for(int i = 0; i < 100; i++){
4          for(int j = 0; j < 100; j++){
5              ctr++;
6          }
7      }
8      iprintf("%i - %i - %i\n", i, j, ctr);
9  }
```

Step 1: FI Feasibility - Proof of Concept
Success!

- Code is glitchable
- Other examples in paper
  (jump out of loop, change branch dir.)

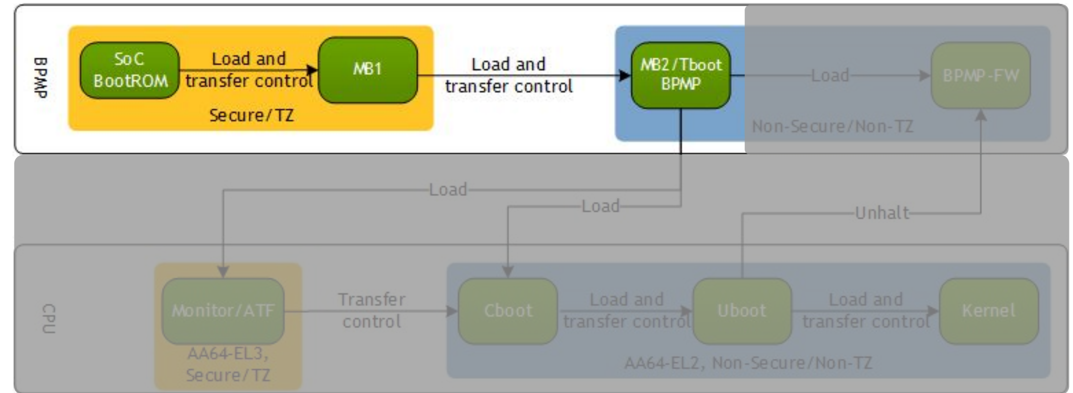
Offset (us)	Pulse Length (us)	Response
0	11.05	'!100 - 100 - 12375\n'
0	11.06	'!100 - 100 - 10134\n'
0	11.07	'!100 - 100 - 10158\n'
0	11.08	'!100 - 100 - 10153\n'
0	11.09	'!100 - 100 - 10251\n'
0	11.10	'!100 - 100 - 10142\n'

Voltage FI Recipe

Step 1	Determining the feasibility of FI ✔
Step 2	**Identifying the FI target and a success indicator**
Step 3	Finding a trigger signal
Step 4	Finding glitch parameters
Step 5	Generating target payload

Step 2: FI Target and Success Indicator
X2 Boot Flow

- MB2: Unprivileged
- MB1: Encrypted & Signed
- BootROM: Read-protected?



[2]

Step 2: FI Target and Success Indicator
BootROM Readability

- Most of BootROM is non-readable
- including: `NVBOOT_IROM_SECRET_STORAGE`
- 1 kilobyte of data is readable

Step 2: FI Target and Success Indicator
BootROM Reversing

Leaked Source Code on GitHub → X1 (T210) and X1+ (T214)

backup-organization/mariko-bootrom

nvboot/core/startup/boot0c.c

```
11   //#include "arbpmp_atcmcfg.h"
12   #include "arsecure_boot.h"
13   #include "project.h"
14   #include "nvboot_hardware_access_int.h"
```

C   Showing the top match   Last indexed on Mar 27

zerospace-nx/switch-bootroms

bootroms/mariko-t214-bootrom/nvboot/core/bpmp/nvboot_bpmp.c

```
41   //#include "arscratch.h"
42   #include "nvrm_drf.h"
43   //#include "arbpmp_atcmcfg.h"
44   //#include "armiscreg.h"
45   //#include "artsa.h"
```

C   Showing the top match   Last indexed on Apr 23

Step 2: FI Target and Success Indicator
BootROM Reversing

- Hidden UART Bootloader
- Deactivated using fuses
- No security checks
- Highest privilege

Step 2: FI Target and Success Indicator

Software Target

`is_fam:` returns 0

`is_ppm:` returns 0

`success:` desired branch

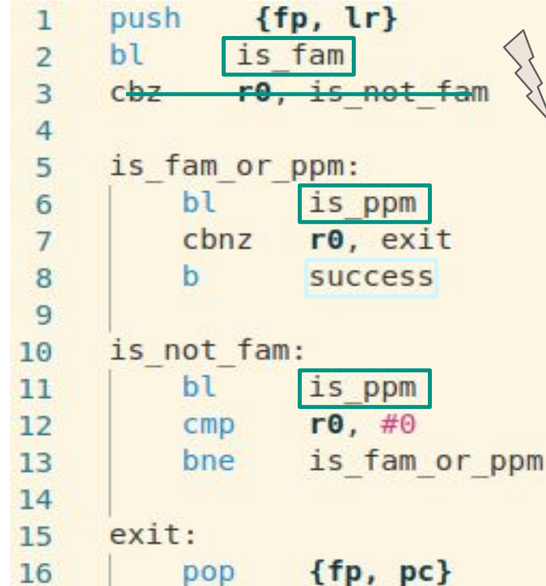
```
1    push    {fp, lr}
2    bl      is_fam
3    cbz     r0, is_not_fam
4
5    is_fam_or_ppm:
6        bl      is_ppm
7        cbnz    r0, exit
8        b       success
9
10   is_not_fam:
11       bl      is_ppm
12       cmp     r0, #0
13       bne     is_fam_or_ppm
14
15   exit:
16       pop     {fp, pc}
```

Step 2: FI Target and Success Indicator
Software Target

`is_fam:` returns 0

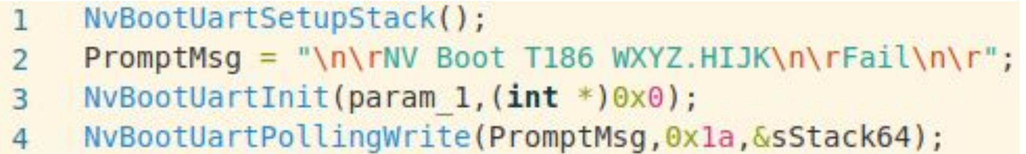
`is_ppm:` returns 0

`success:` desired branch



Step 2: FI Target and Success Indicator
Success Indicator

After entering desired branch:



Voltage FI Recipe

Step 1	Determining the feasibility of FI	✓
Step 2	Identifying the FI target and a success indicator	✓
Step 3	**Finding a trigger signal**	
Step 4	**Finding glitch parameters**	
Step 5	Generating target payload	

Steps 3 and 4: Glitching the target

Success #2!

- Entered UART Bootloader with:

  Offset: 2.6338 ms

  Pulse: 11.32 µs

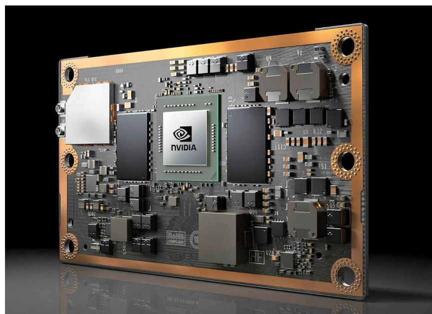
- Repeatable within < 10 seconds

Voltage FI Recipe

Step 1 Determining the feasibility of FI ✓
Step 2 Identifying the FI target and a success indicator ✓
Step 3 Finding a trigger signal ✓
Step 4 Finding glitch parameters ✓
Step 5 Generating target payload

Attack Overview – Revisited

Device Under Test



[1]

X2 SoC

Goal

Leak Read-Protected BootROM

Outcome

Privileged Code Exec.

Leaked Decryption Keys

Leaked Read-Protected BootROM

Conclusion

- Manufacturers should not forget attacks that are around for more than 20 years
- Do not ship debug bootloaders unprotected against reactivation via fault injection
- Implement glitching detection/countermeasures

Preprint:
https://arxiv.org/abs/2108.06131

References

[1] https://i.ytimg.com/vi/1tWqlM8uULc/maxresdefault.jpg (15.08.21)

[2] https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/Tegra%20Linux%20Driver%20Package%20Development%20Guide/images/image2_6.png (15.08.21)

[3] Nvidia Corp., Technical Reference Manual Nvidia Parker Series SoC, v1.0p (Jun. 2017)

Thank you for your attention!