Fault Injection Attacks on Secure Automotive Bootloaders

Nils Weiss <nils@dissec.to>

Enrico Pozzobon <enrico@dissec.to>

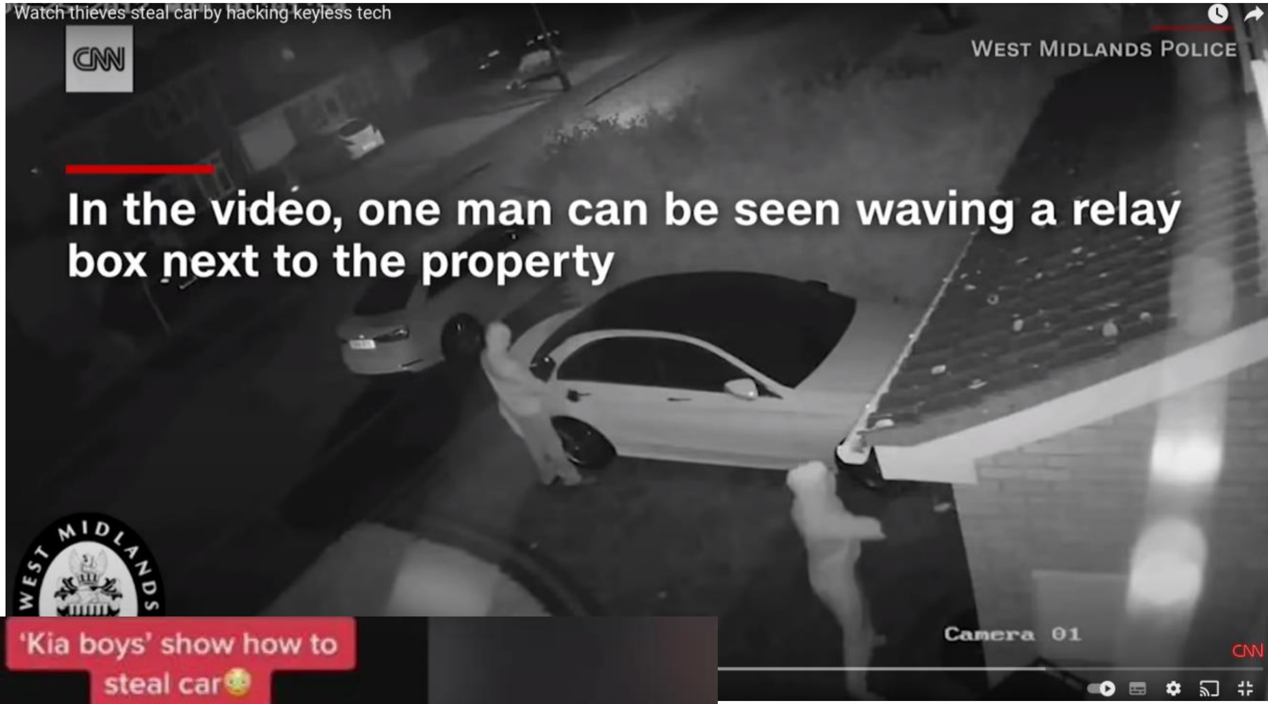
Fault Injection Attacks on Secure Automotive Bootloaders

Nils Weiss <nils@dissec.to>

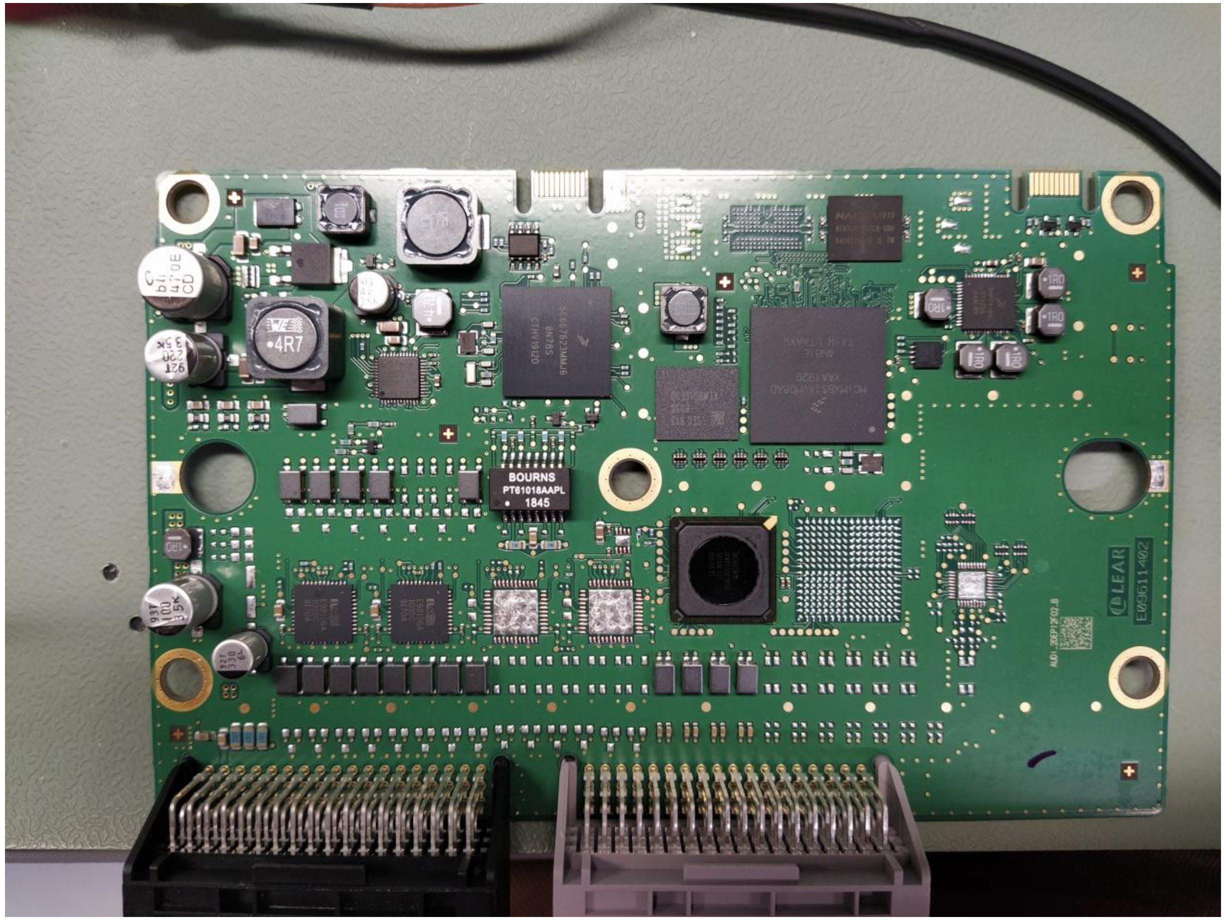
Enrico Pozzobon <enrico@dissec.to>

Threat Model for HW Attacks in Automotive

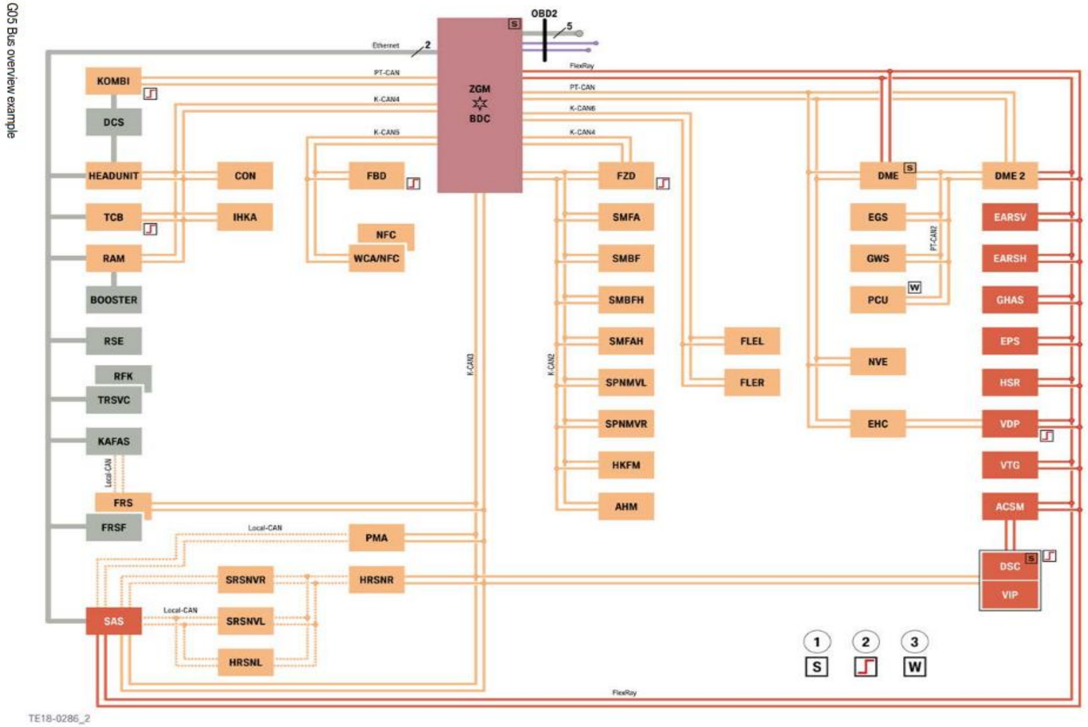
- Vehicle Theft (entire cars)
  - Break immobilizers
- Stolen ECUs aftermarket
  - Virgin ECUs
- Chip-Tuning
- Feature on Demand
- Mileage manipulation
- Ad-Blue manipulation
- E-Fuel detection



The target:



- Gateway-ECU
- Root of the Network
- Trust anchor for certain services



Safe and "Secure" microcontrollers

NXP

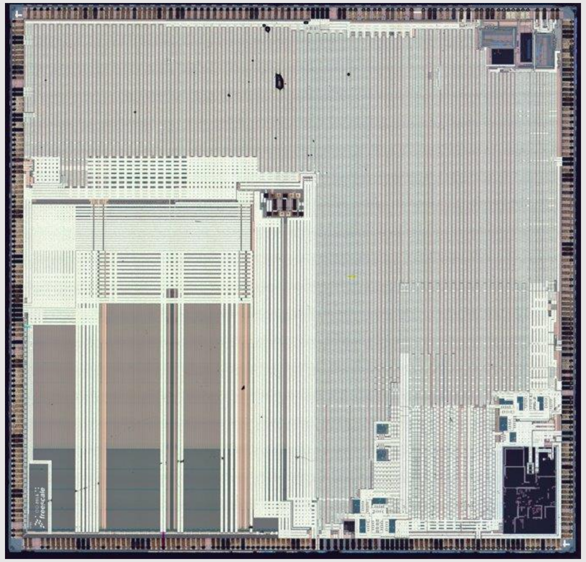
Search

Sign In / Register

LANGUAGE

Ultra-Reliable MPC574xB/C/G MCUs for Automotive and Industrial Control and Gateway

MPC574xB-C-G Receive alerts



Silicon of the MPC5748G, courtesy of Texplained

n Resources Training Support BUY/PARAMETRICS PACKAGE/QUALITY

The MPC574xB/C/G family of MCUs (eg. MPC5746C, MPC5748G) provides a highly integrated, safe and secure single-chip solution for next-generation central body control, gateway and industrial applications.

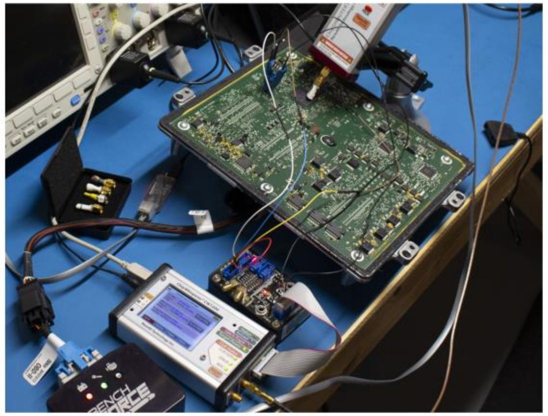
What makes an MCU "Automotive"?

- It is tolerant to a wide range of temperatures.
- It can withstand high voltage transients.
- It doesn't break easily in the presence of electromagnetic pulses.

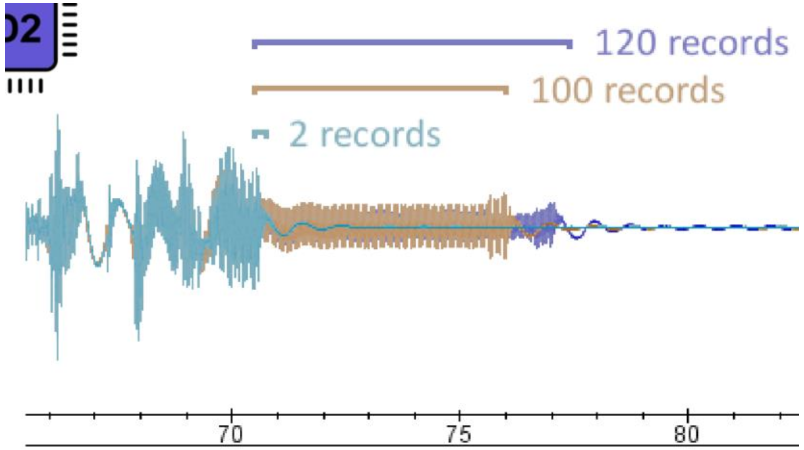
Abbildung		
Herst.-Teilenr.	ATSAME51J19A-AFT	ATSAME51J19A-AU
Herst.	Microchip Technology	Microchip Technology
Lieferant	Microchip Technology	Microchip Technology
DK-Teilenr.	ATSAME51J19A-AFTTR-ND ATSAME51J19A-AFTCT-ND ATSAME51J19A-AFTDKR-ND	ATSAME51J19A-AU-ND
Beschreibung	IC MCU 32BIT 512KB FLASH 64TQFP	IC MCU 32BIT 512KB FLASH 64TQFP
Preis	6,62000 €	5,89000 €
Lagerbestand	0	116
Mindestmenge	1	1
Serie	Automotive, AEC-Q100, SAM E51	SAM E51

Existing glitching attacks on ECUs

- Safety ≠ Security from Riscure (Attacking DCF Record Loading)
- BAM BAM by Colin o'Flynn
- Nasahl and Timmers used glitching attacks on an evaluation setup to obtain code execution on an AUTOSAR-based demonstration ECU



(c) E41 ECU "In-Situ" Target



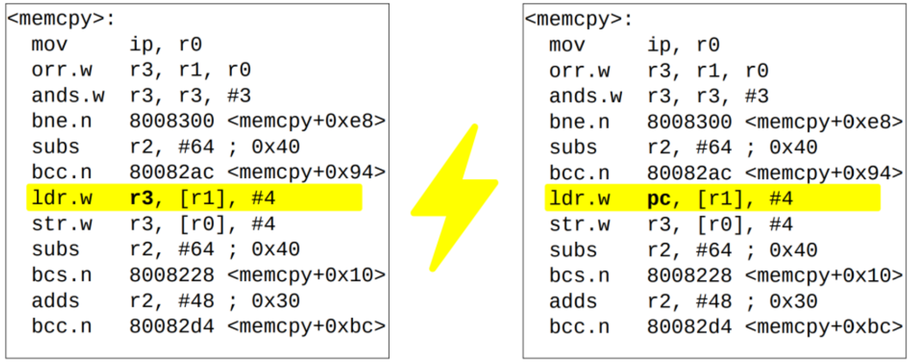
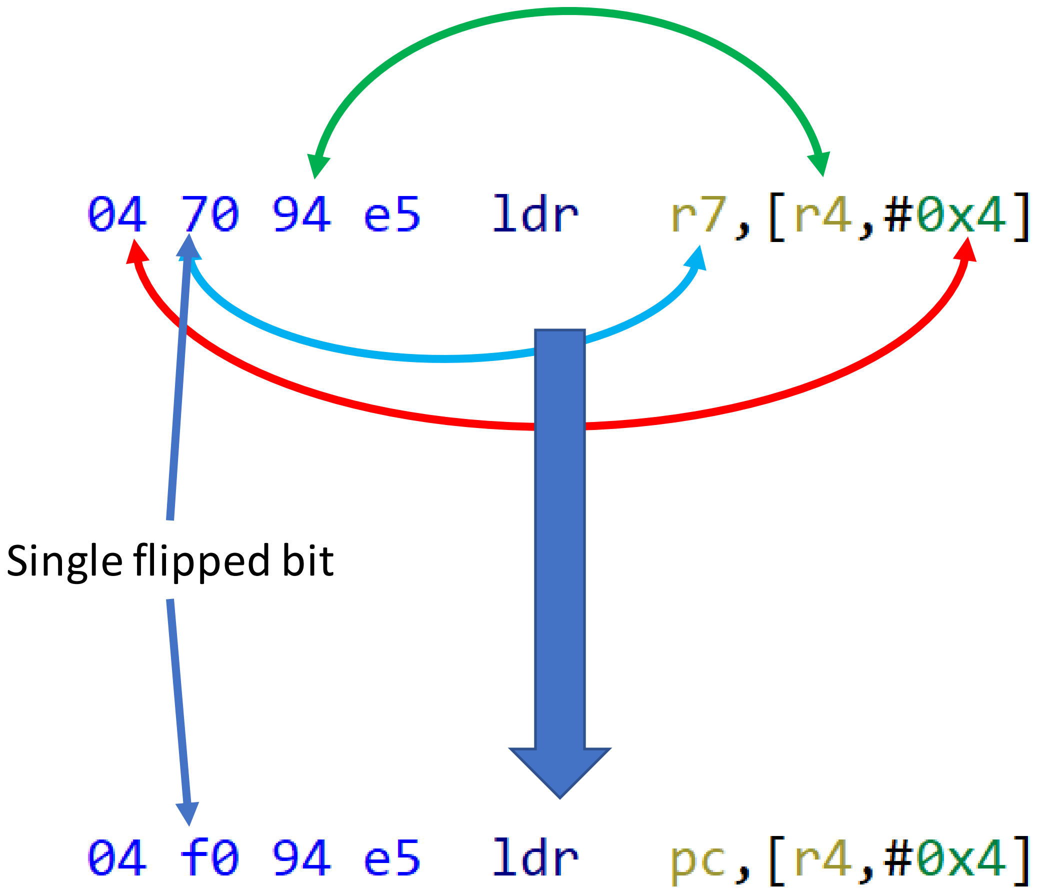


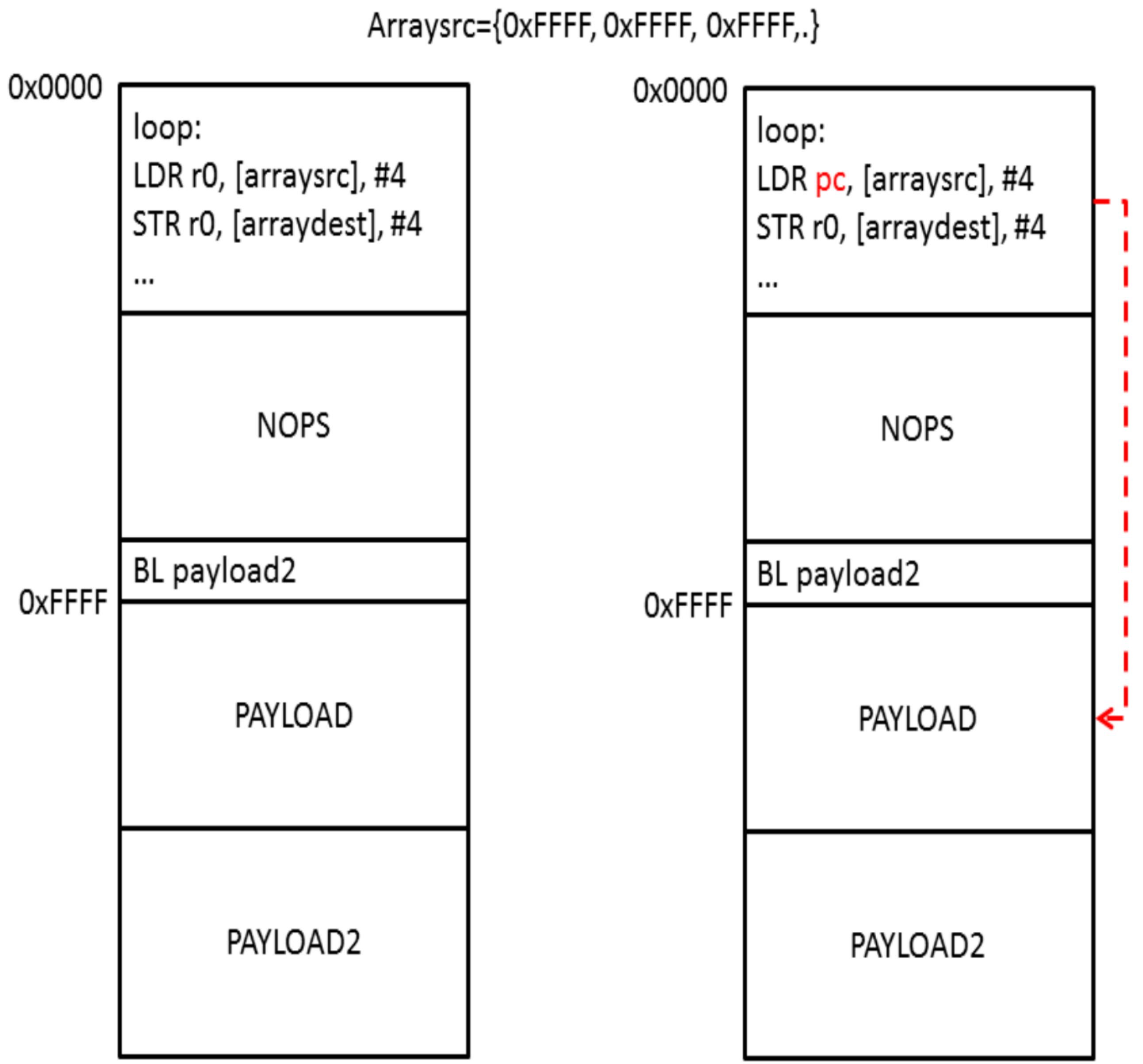
Fig. 3. Glitch injection moment during the *memcpy* function.

Controlling PC by Fault Injection on ARM



Wild Jungle Jumps

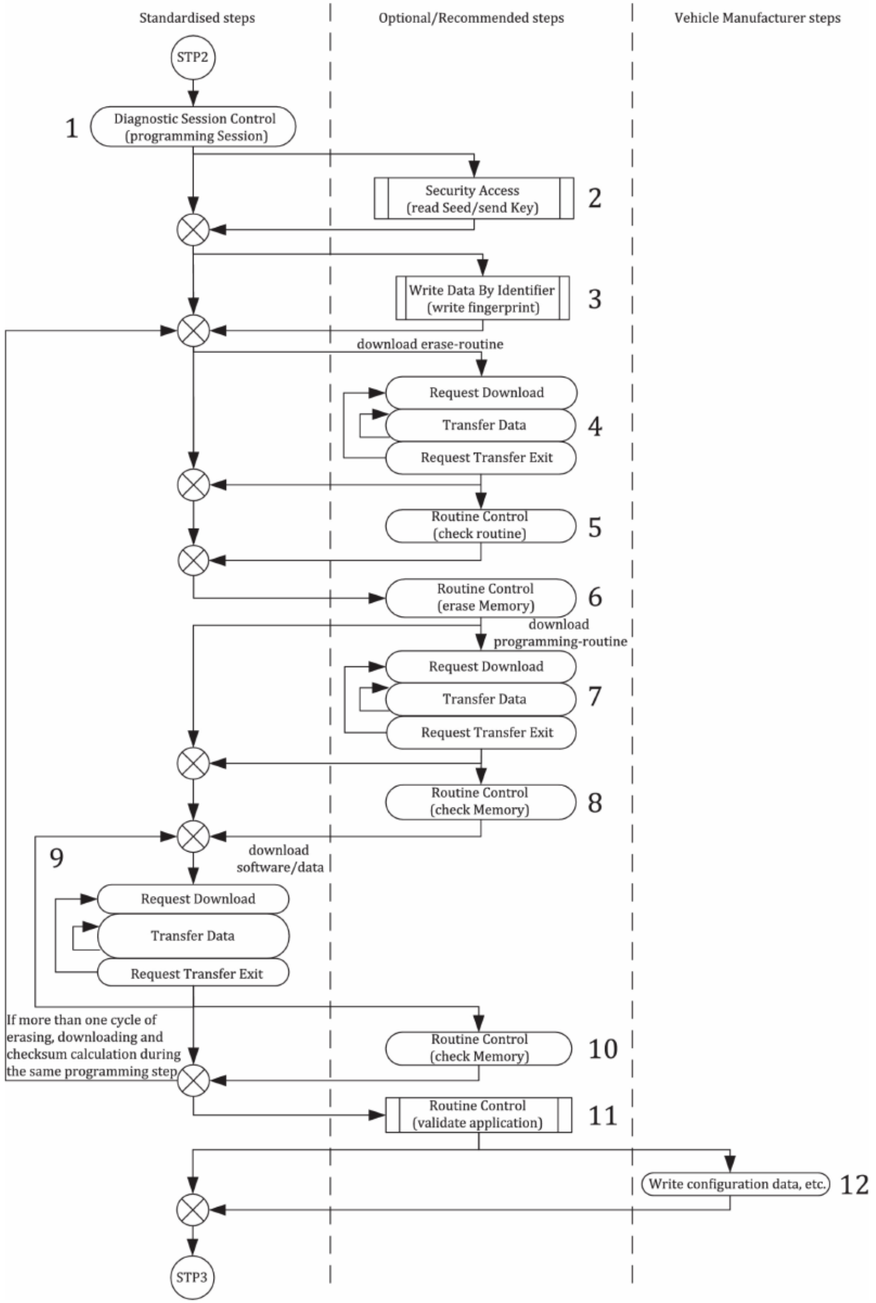
- "Until now, wild jungle jumps were only exploitable in laboratory environments and considered impossible in practice" - Spensky et al. *Glitching demystified: Analyzing control-flow-based glitching attacks and defenses*



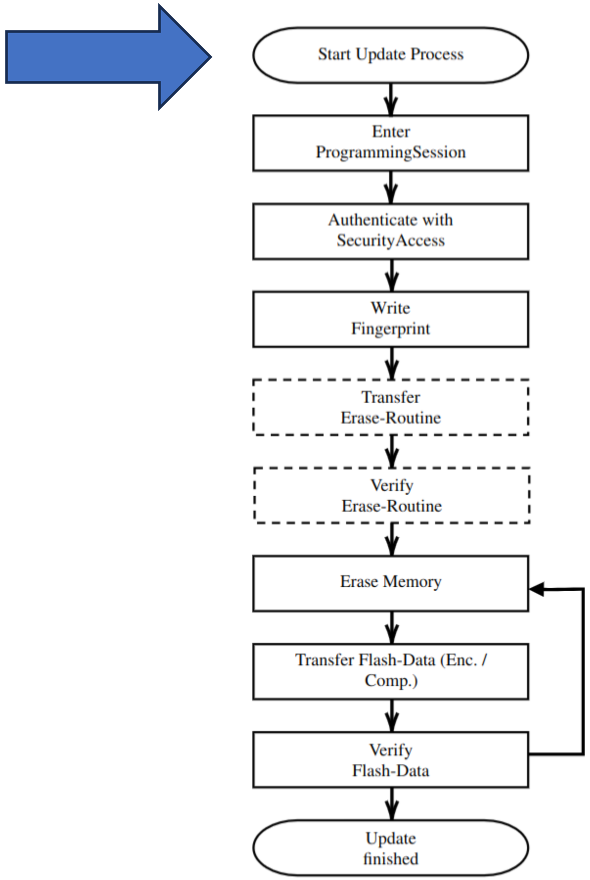
Picture by James Gratchof – Proving the wild jungle jump

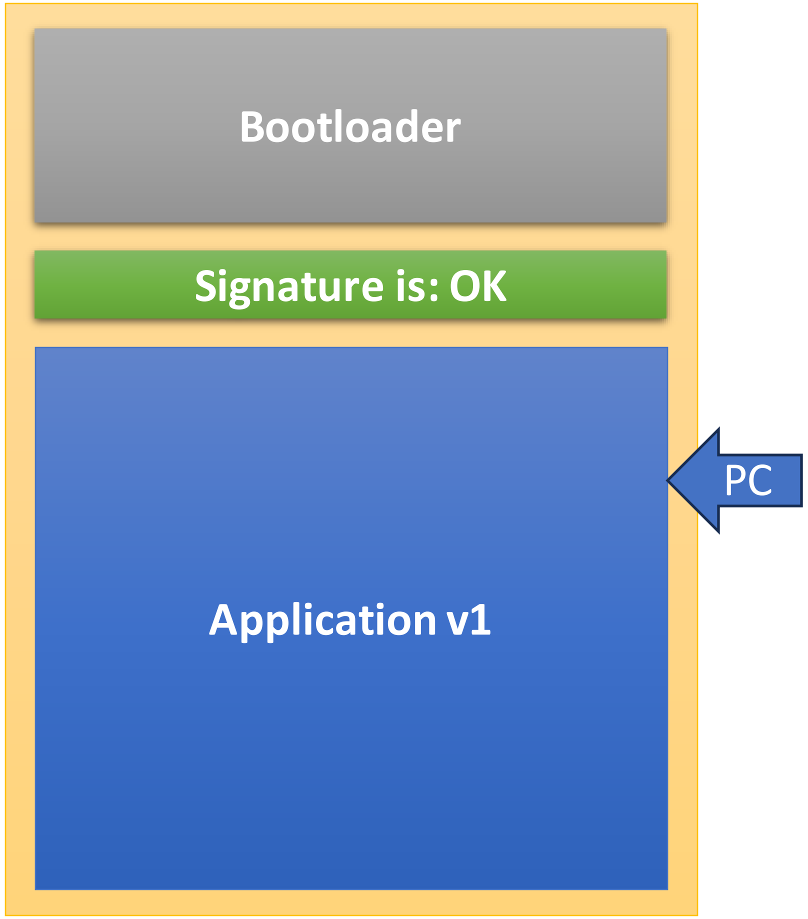
UDS / ISO 14229-1:2020

- Communication with ECUs is mostly standardized
- Modern ECUs supports UDS (Unified Diagnostic Services)
  - Configuration of ECUs
  - Reading Information and DTCs
  - Erasing / Flashing
- UDS defines Flashing-Procedure
  - Small variations for each individual OEM

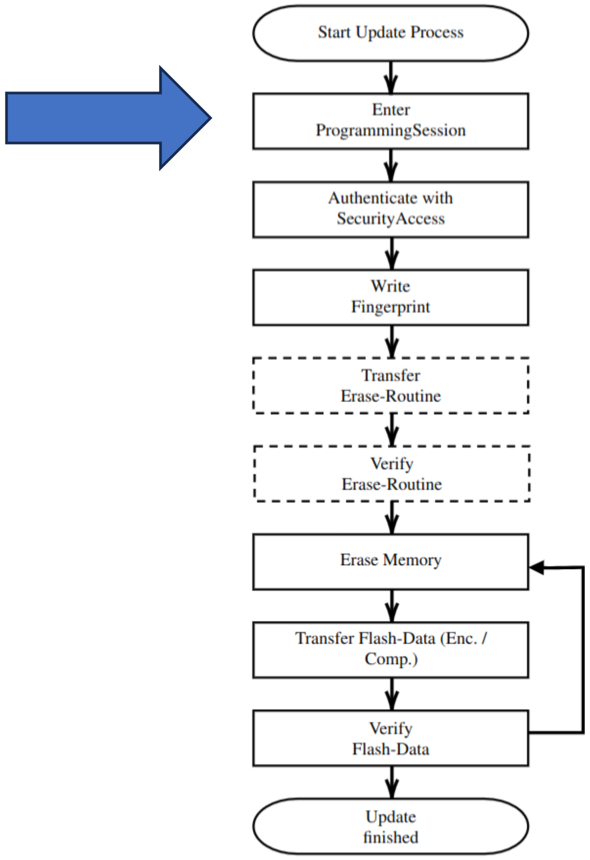


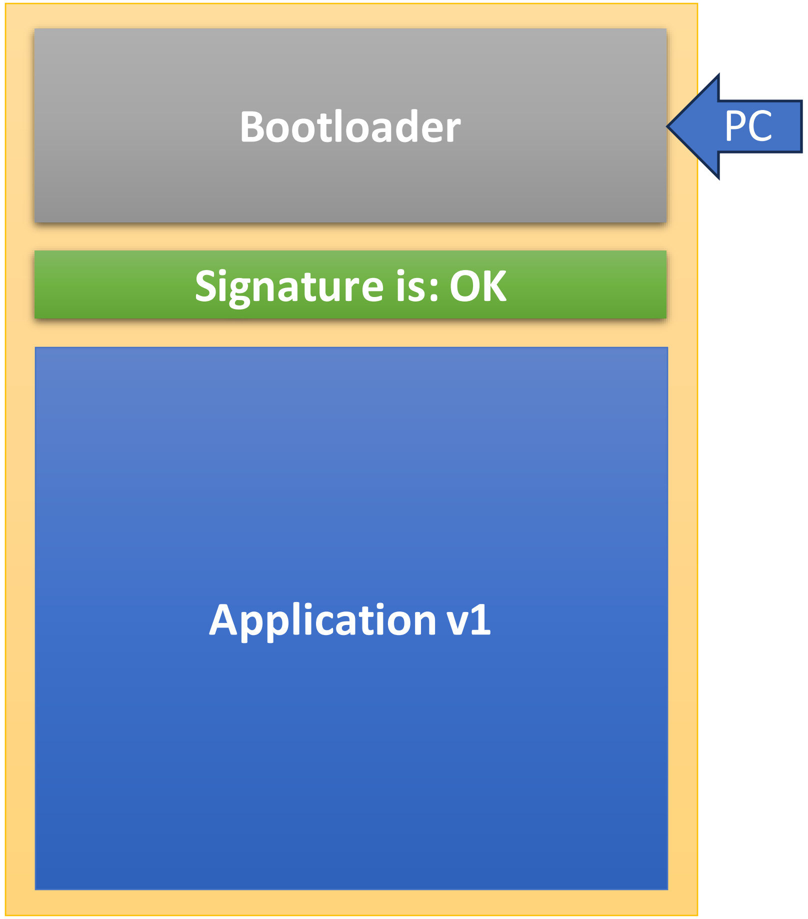
UDS Update Process



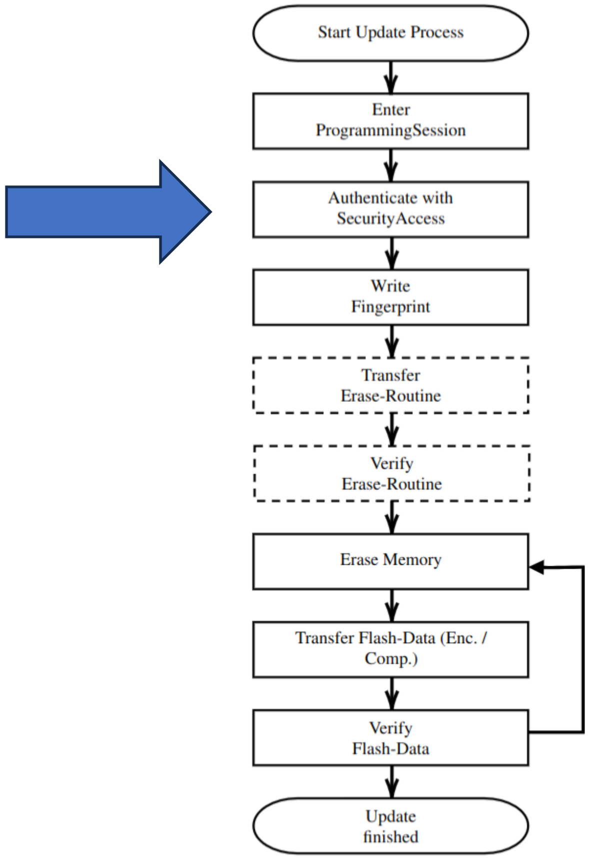


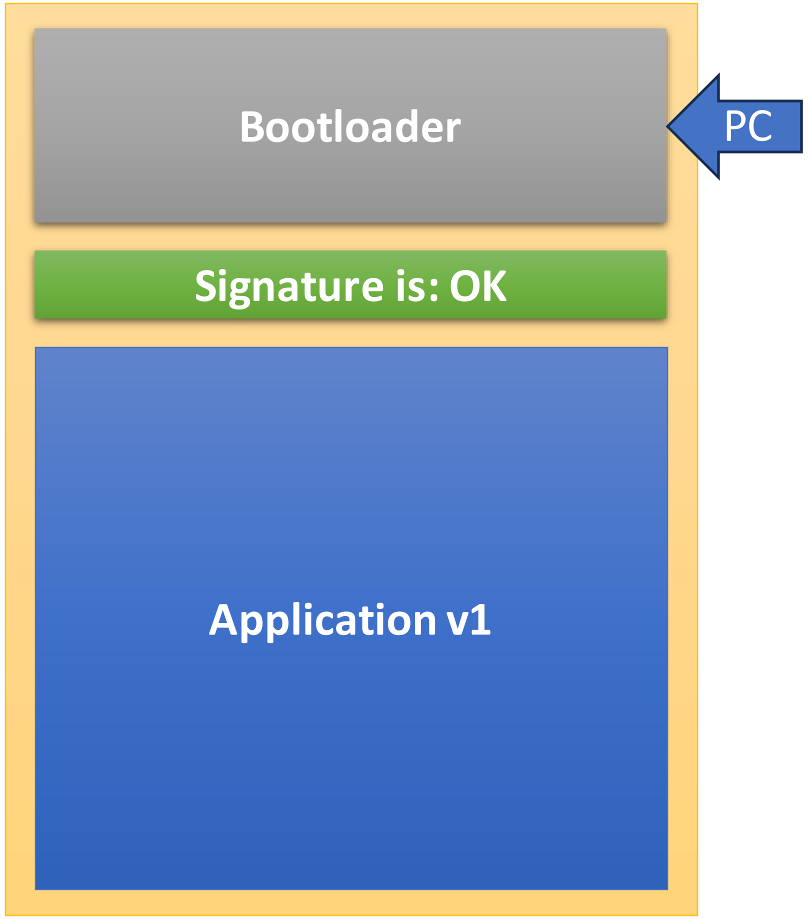
UDS Update Process



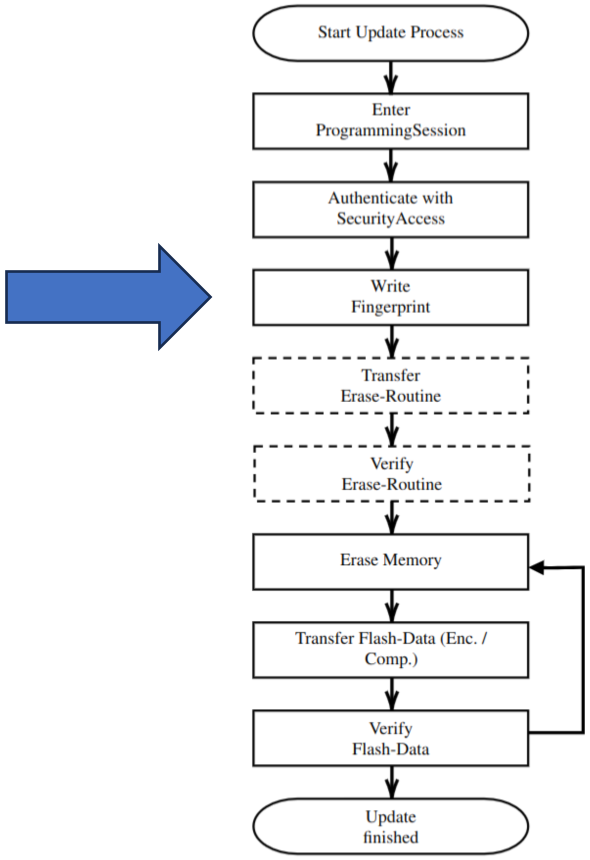


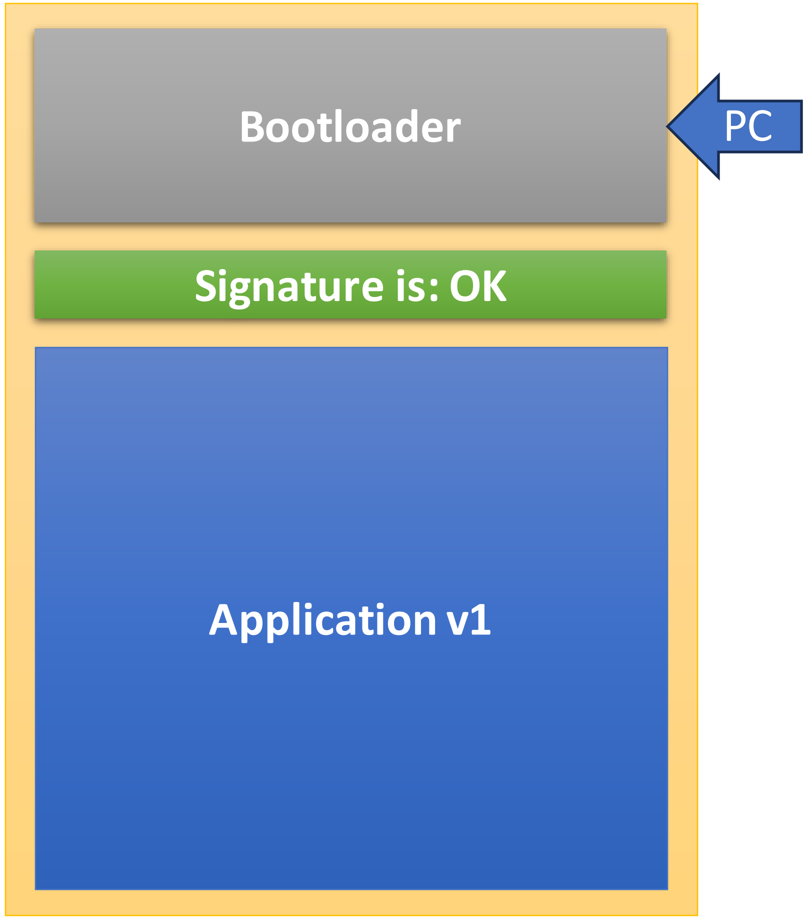
UDS Update Process



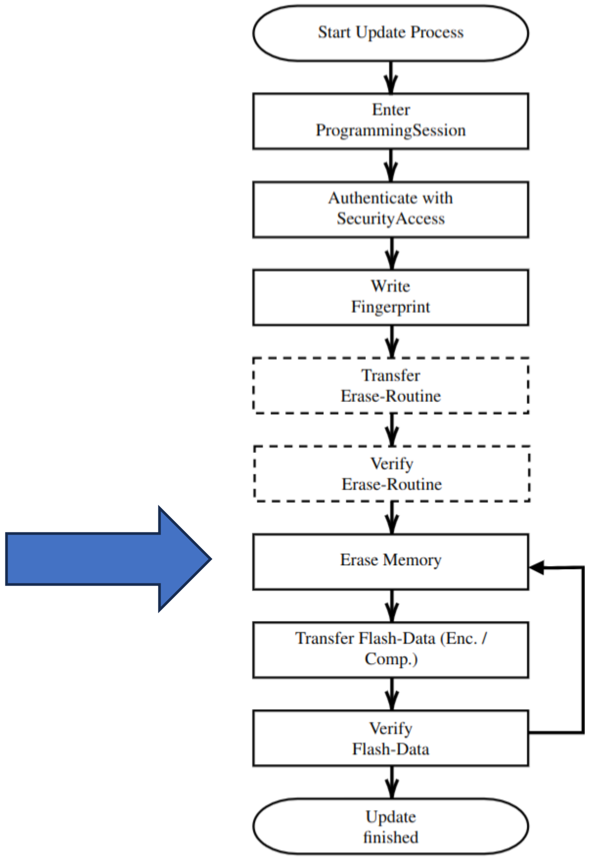


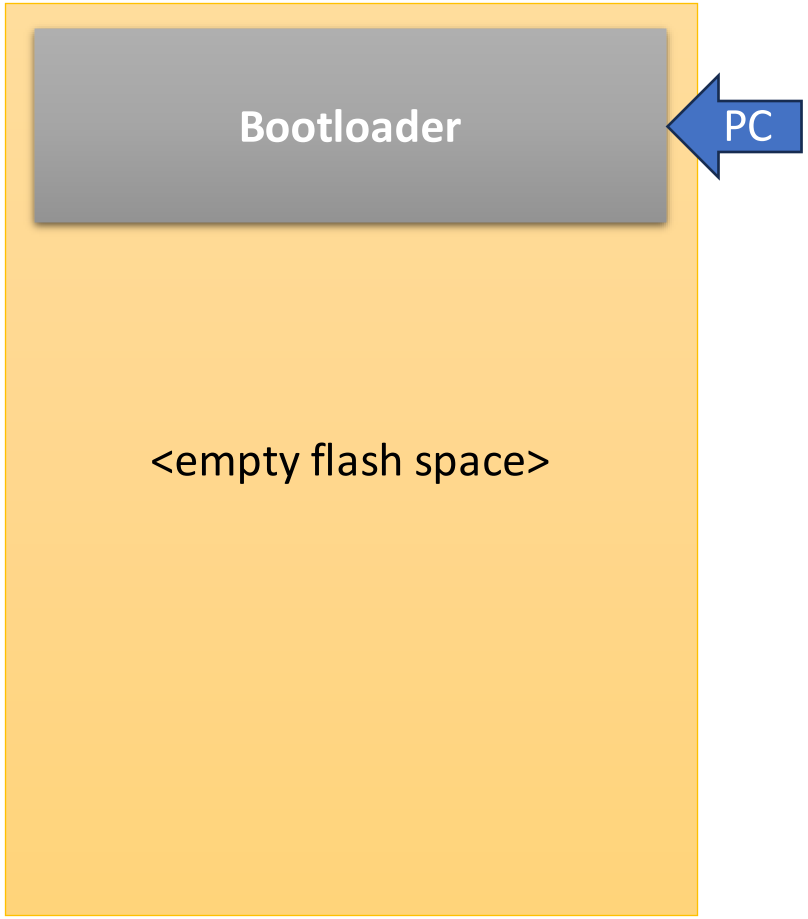
UDS Update Process



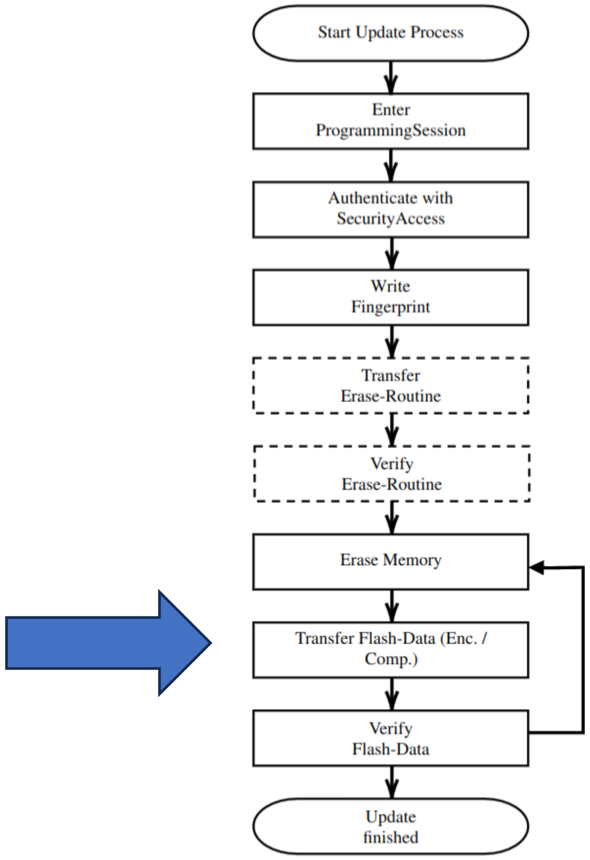


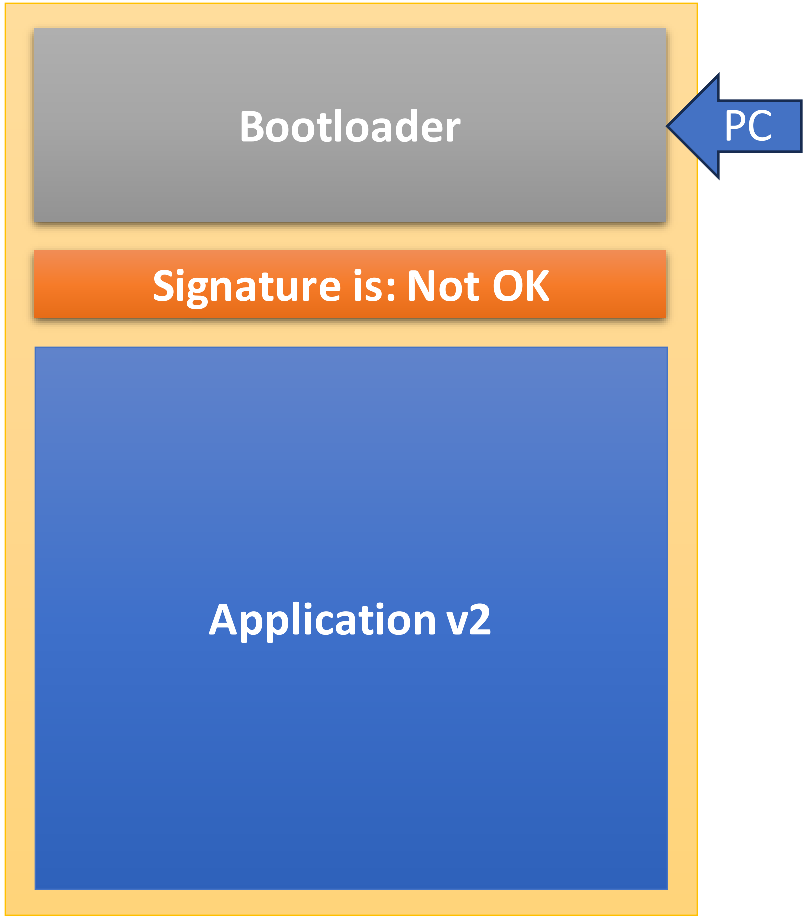
UDS Update Process



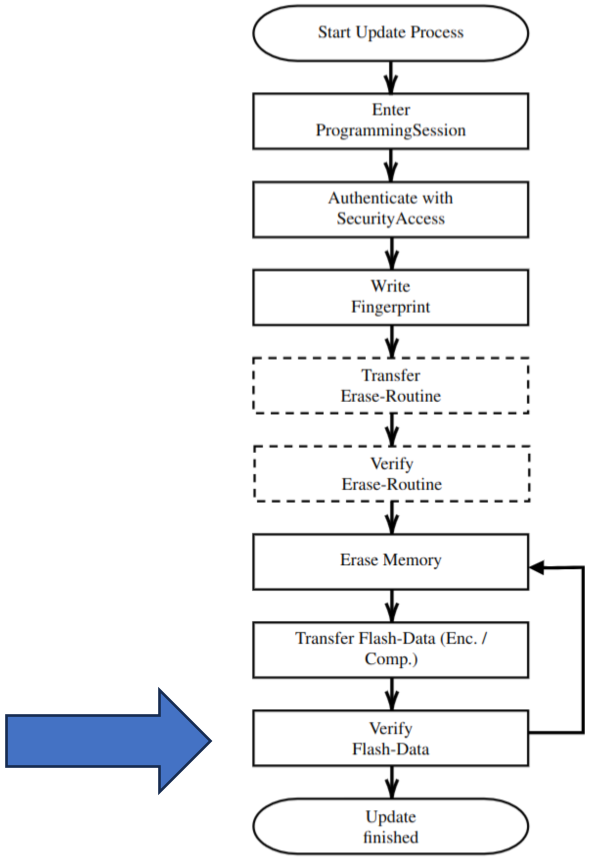


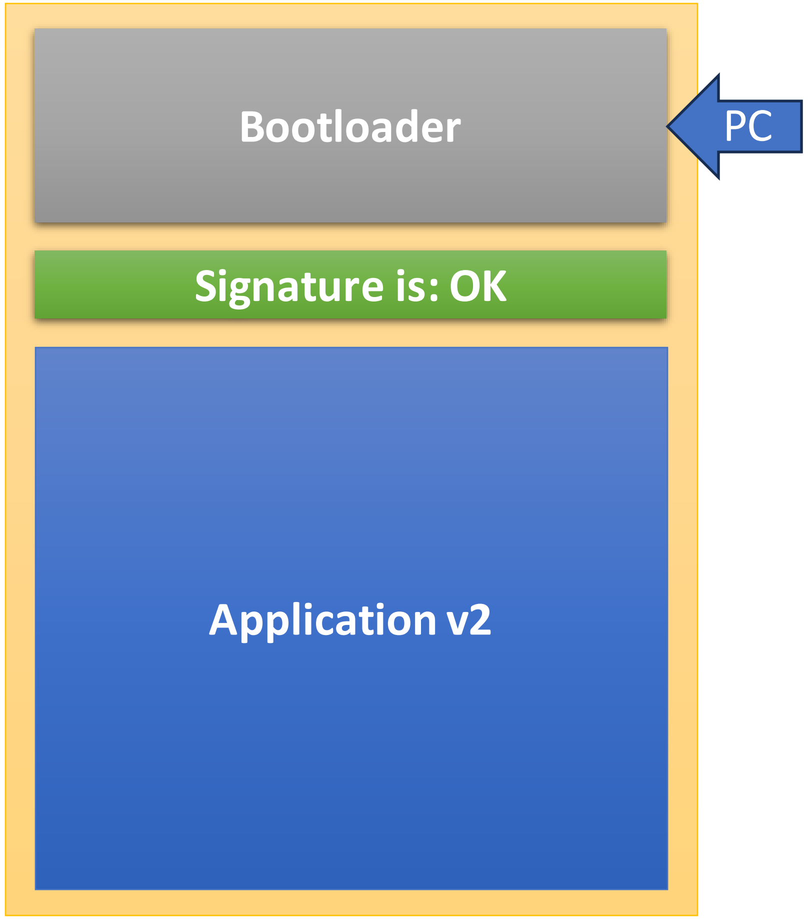
UDS Update Process



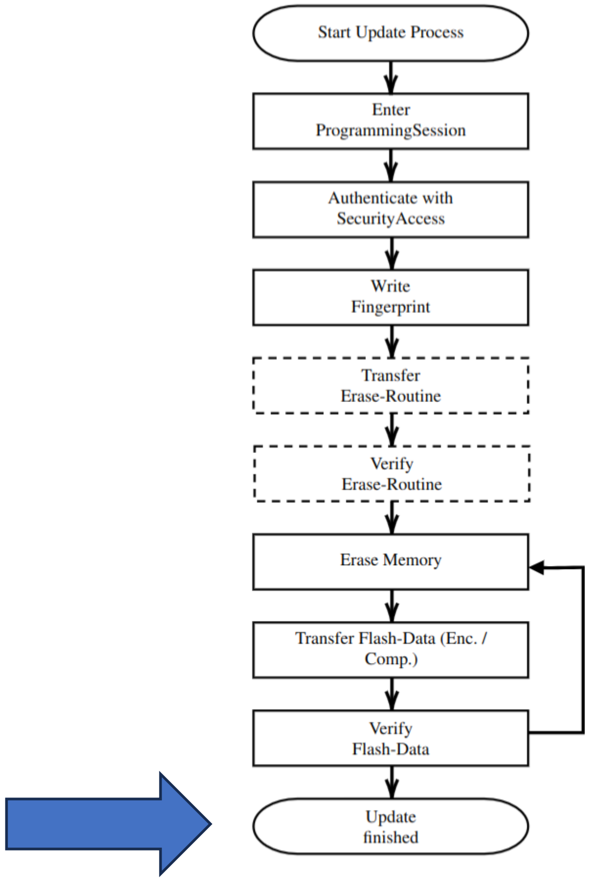


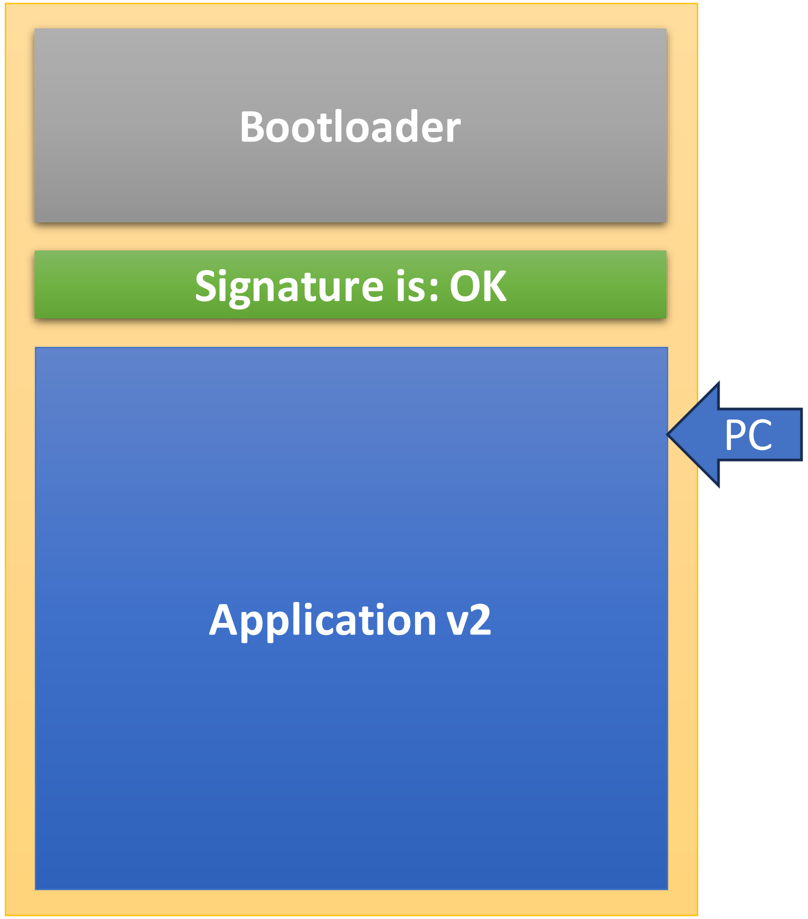
UDS Update Process



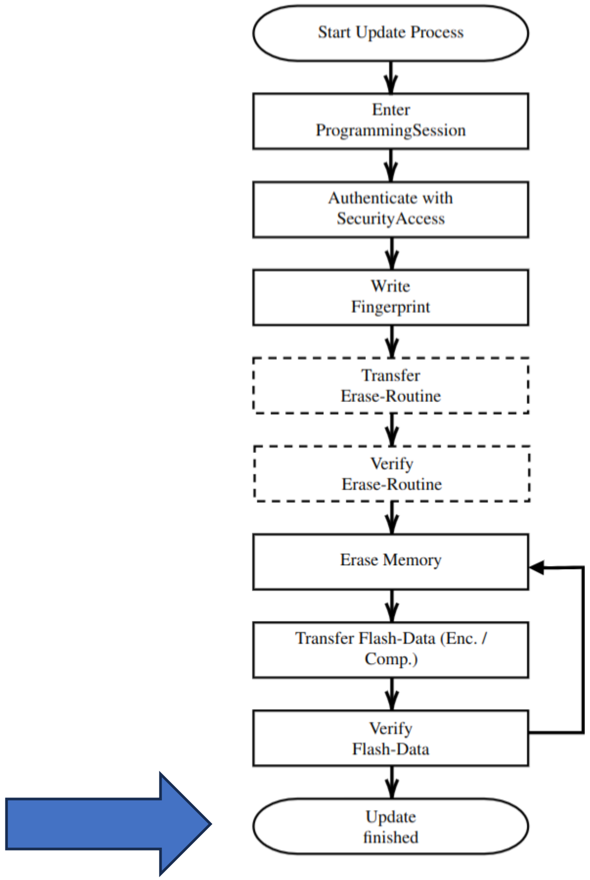


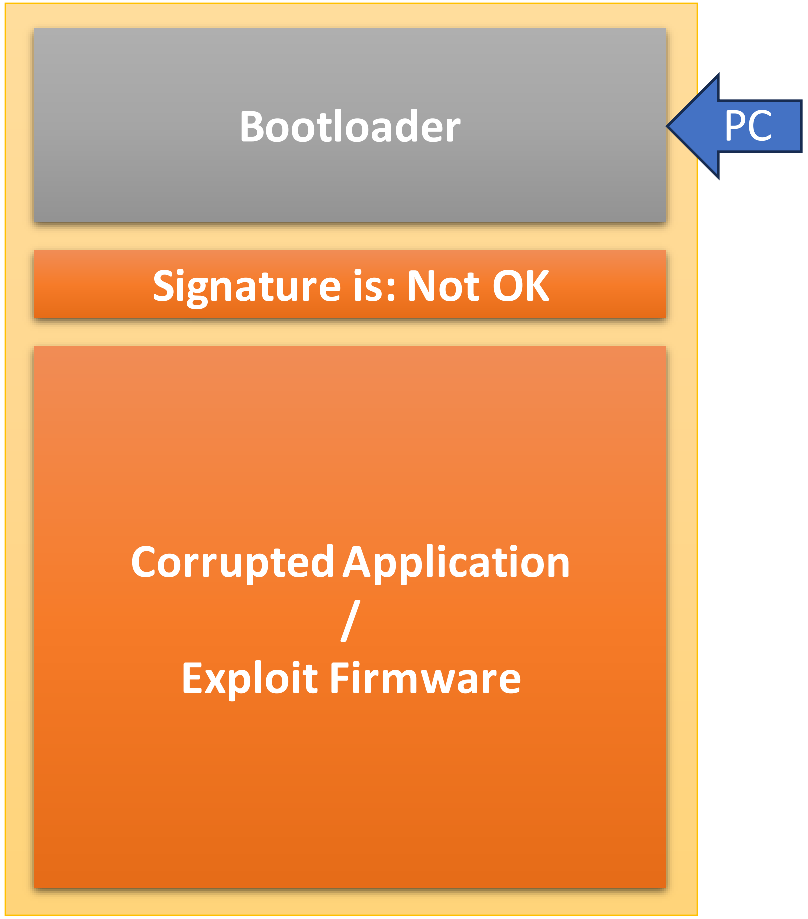
UDS Update Process



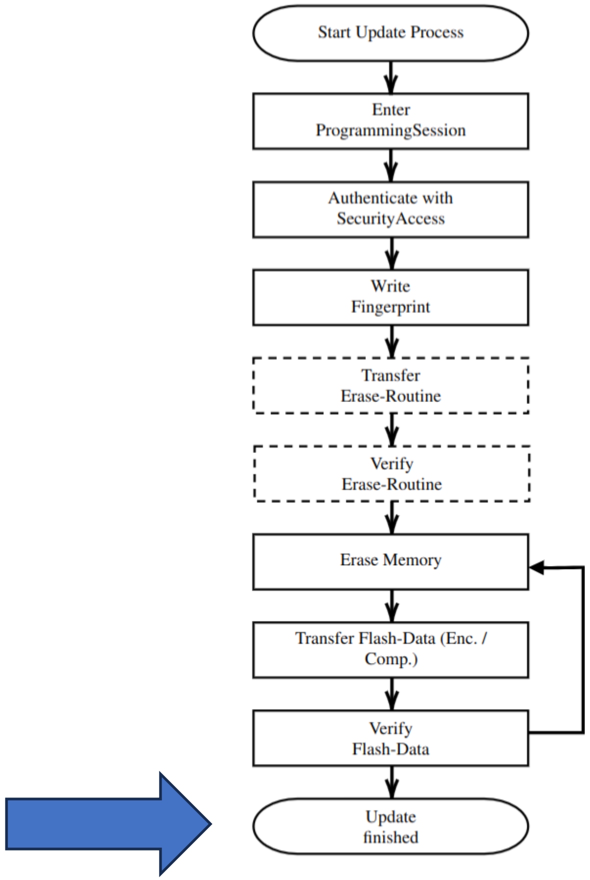


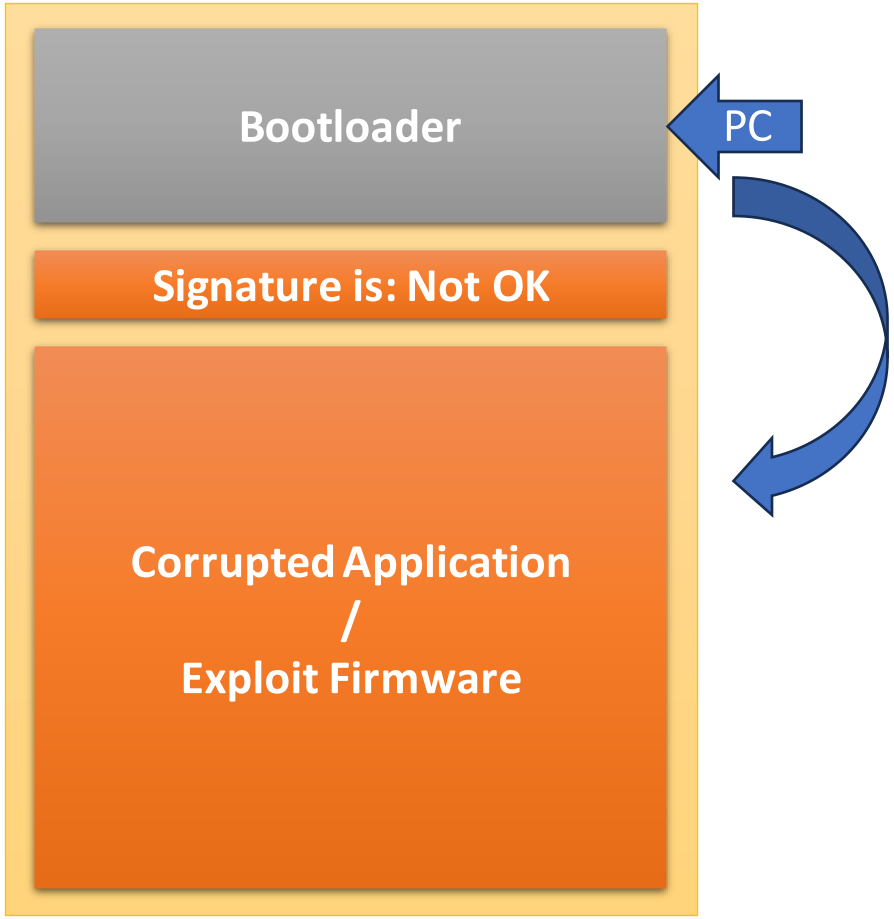
If the application signature verification fails, the bootloader will not jump to the application





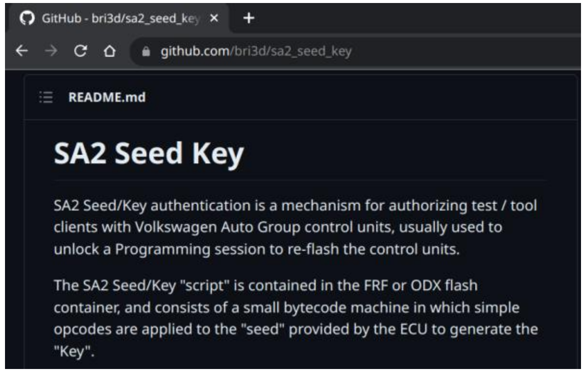
The attack: Use fault injection to jump from bootloader to unauthenticated payload

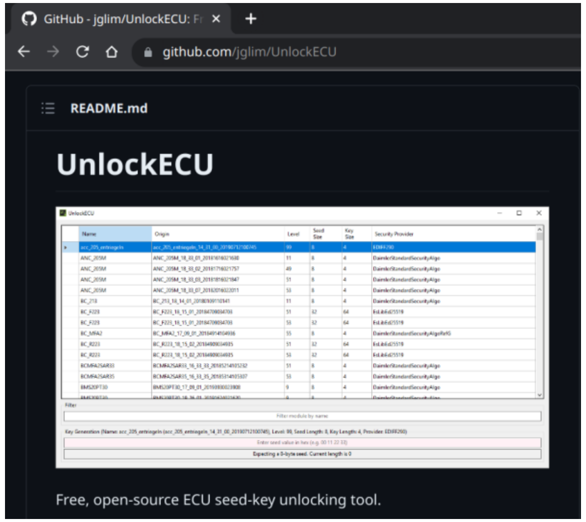




UDS Security Access

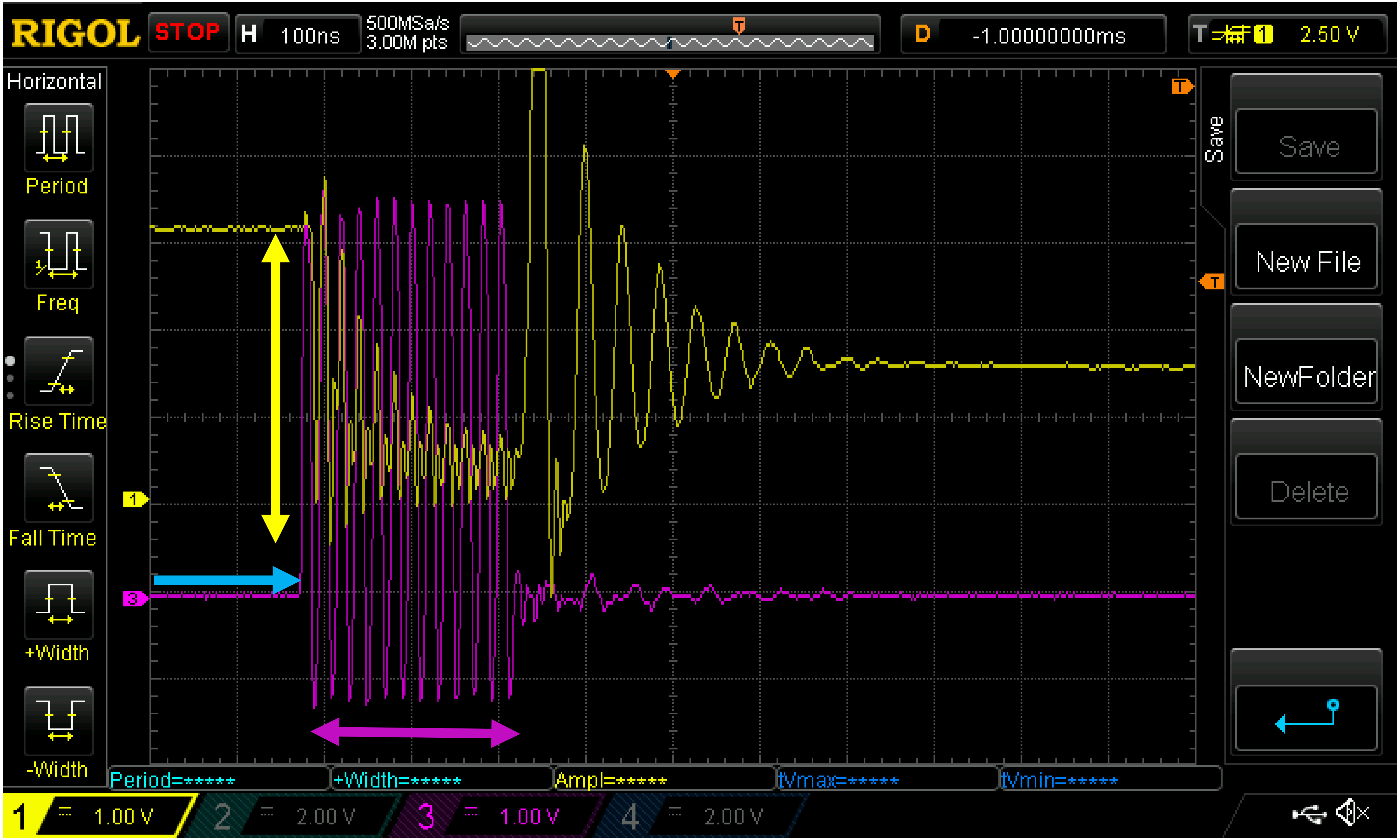
- Security Access Algorithms are available on GitHub





- OEM-Tools leak on shady internet forums

- Many Security Access implementations are leaked or broken or easy to overcome



RIGOL
STOP
H 100ns
500MSa/s
3.00M pts
D -1.00000000ms
T 1 2.50 V
Horizontal
Period
Freq
Rise Time
Fall Time
+Width
-Width
Save
Save
New File
NewFolder
Delete
Period=*****
+Width=*****
Ampl=*****
tVmax=*****
tVmin=*****
1 1.00 V
2 2.00 V
3 1.00 V
4 2.00 V

EMFI parameters (search space)

- **Injection coil:** (shape, size, number and direction of turns),
- **Position** (x, y, z) in space of the injection coil,
- **Duration** of the activation of the coil,
- **Voltage** across the coil (aka across the injector reservoir capacitor),
- **Offset** from trigger signal,
  - if the target firmware has deterministic execution time, this is equivalent to choosing which instruction to attack!
- **Memory** / state of the target
  - Depends on the messages exchanged before the fault.

- And other environmental factors that can't be accounted for on stage.

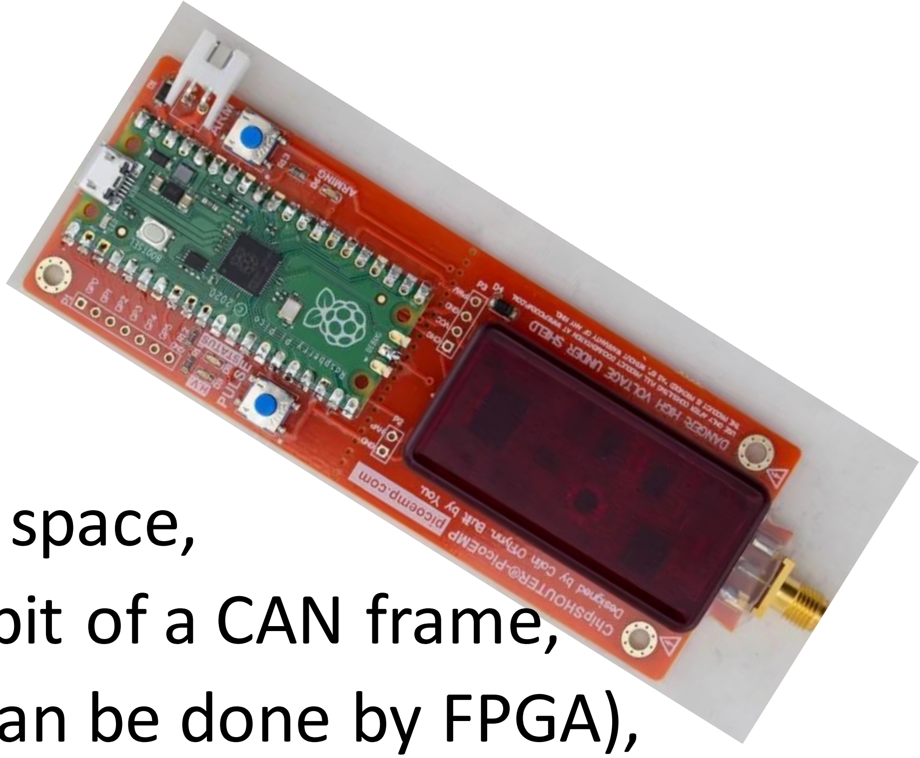
EMFI parameters (search space)

- **Injection coil:** (shape, size, number and direction of turns),
- **Position** (x, y, z) in space of the injection coil,
- **Duration** of the activation of the coil,
- **Voltage** across the coil (aka across the injector reservoir capacitor),

These don't depend on the target software, only on the hardware

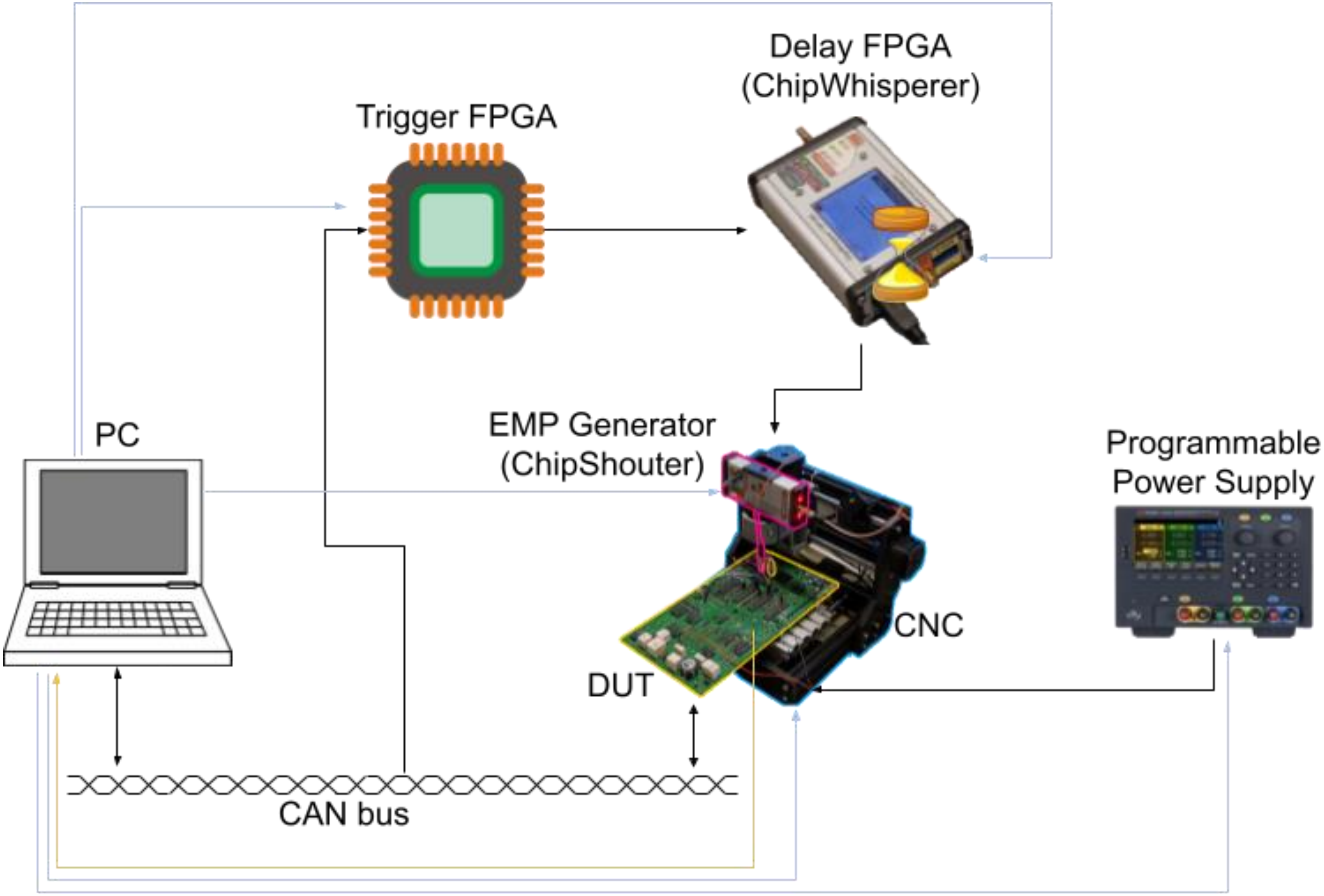
- **Offset** from trigger signal,
  - if the target firmware has deterministic execution time, this is equivalent to choosing which instruction to attack!
- **Memory** / state of the target
  - Depends on the messages exchanged before the fault.

EMFI Fault setup

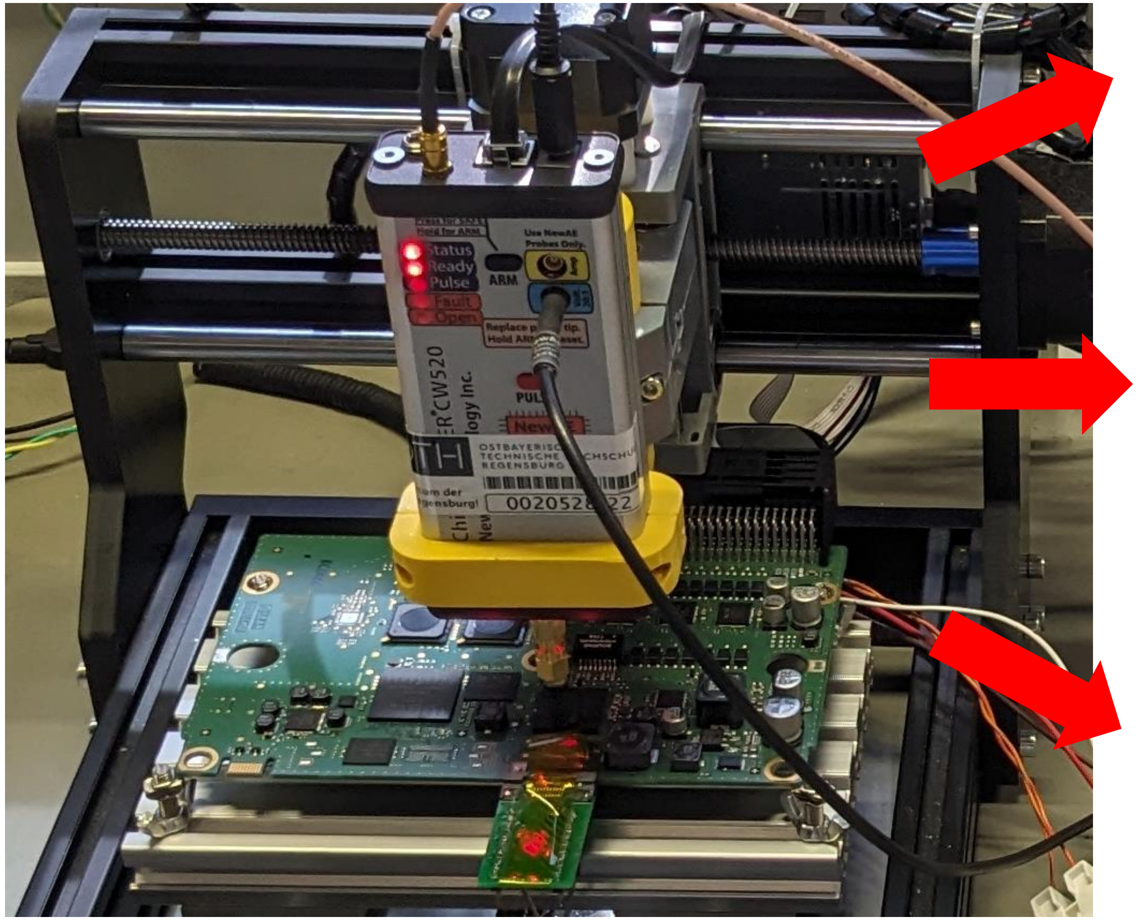


- **ChipSHOUTER**: generates the EMP,
- **CNC Mill**: Positions the injection coil in the 3D space,
- **Generic FPGA**: Precisely triggers on a specific bit of a CAN frame,
- **ChipWhisperer**: Delays the trigger (optional, can be done by FPGA),
- **Programmable supply**: to power-cycle the target when it crashes
- **CAN interface**: to transfer the exploit and bring the ECU to a specific state
- **UART interface**: to get feedback from the target

Total cost: ~5000$ (can be reduced to ~300$ by using PicoEMP)


Delay FPGA
(ChipWhisperer)
Trigger FPGA
PC
EMP Generator
(ChipShouter)
Programmable
Power Supply
CNC
DUT
CAN bus

Fault Outcomes



- Normal Response
- Corrupted CAN response
- ECU resets, no response
  - Emission of an exception Stack Dump over UART

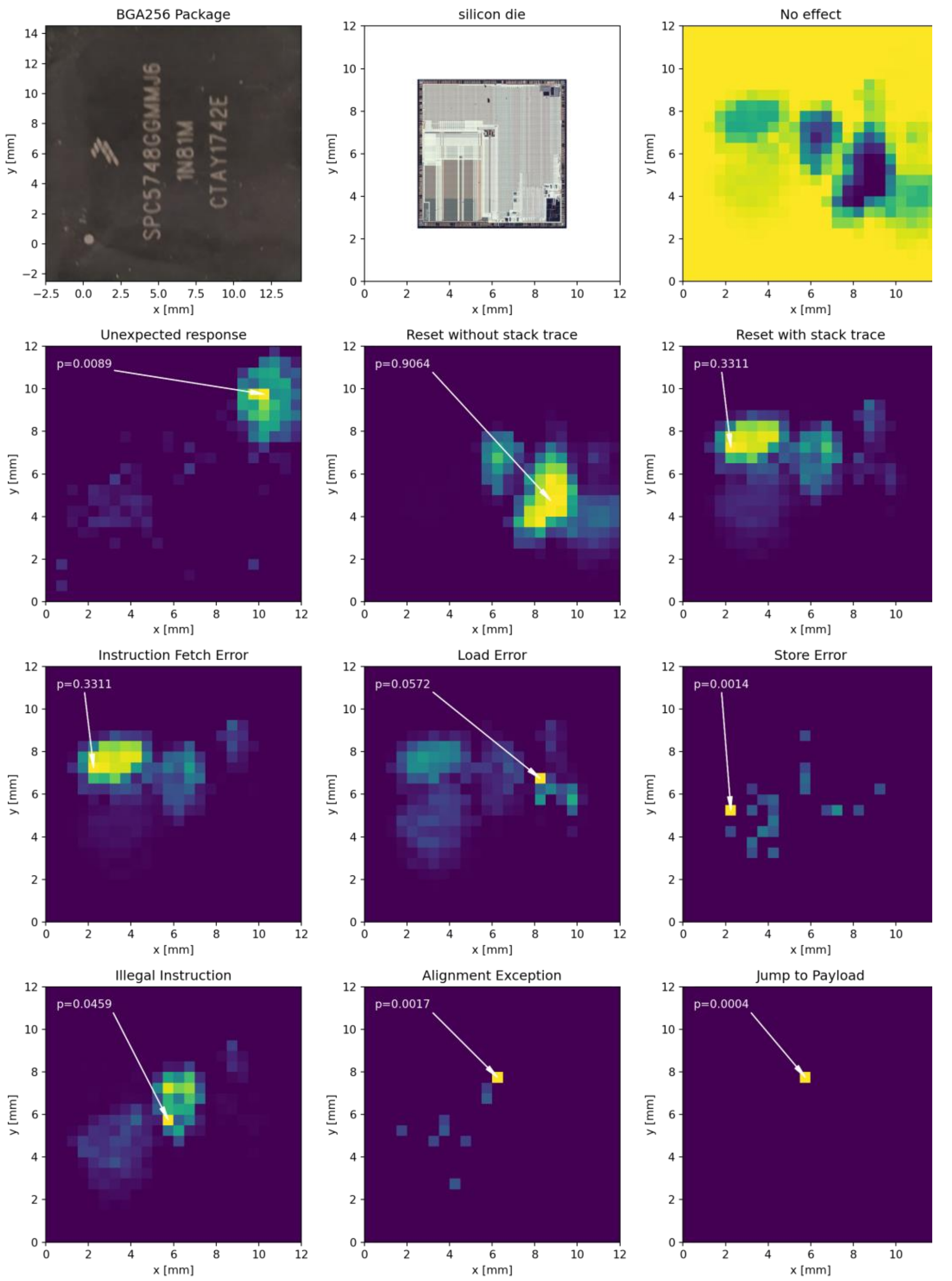
Example of a stack trace

```
Machine Check Exception
Exception number:          1
Exception address:         0105D1EE
Stack pointer:             40006F98
R0    010F2FB8  R8     400070EC  R16  00000000  R24   400070EC
R1    40006F98  R9     013996A8  R17  00000000  R25   4004FAD8
R2    013DF918  R10    00000005  R18  00000000  R26   00000002
R3    02029200  R11    FFF1E400  R19  00000000  R27   00000002
R4    0000FFF1  R12    400070DC  R20  00000000  R28   0000E400
R5    00000000  R13    4001DD90  R21  00000000  R29   0000FFF1
R6    010F3130  R14    00000000  R22  00000000  R30   40007090
R7    0000FFF1  R15    00000000  R23  00000000  R31   4003EFA8
-----------------------------------------------------------------
XER    00000000  CR   80000000  LR    010F2FB8
USPRG0 00000000  CTR  010F2EF4  IP    --------
-----------------------------------------------------------------
SPRG0 00000000  SRR0   013D1FD6  IVPR 01000100  MSR   00000200
SPRG1 400200C8  SRR1   02029200  DEAR 00000000  PVR   81530000
SPRG2 00000000  CSSR0  00000000  ESR  00000000
SPRG3 00000000  CSSR1  00000000  MCSR 00088008
MCSSR0 0105D1EE  MCAR 00000078
MCSSR1 02021200
PID0  00000000
-----------------------------------------------------------------
PIR 00000000

S T A C K T R A C E
> 0x010F2FB8
> 0x010F307A
> 0x010F1F1E
> 0x011281FC
...
```

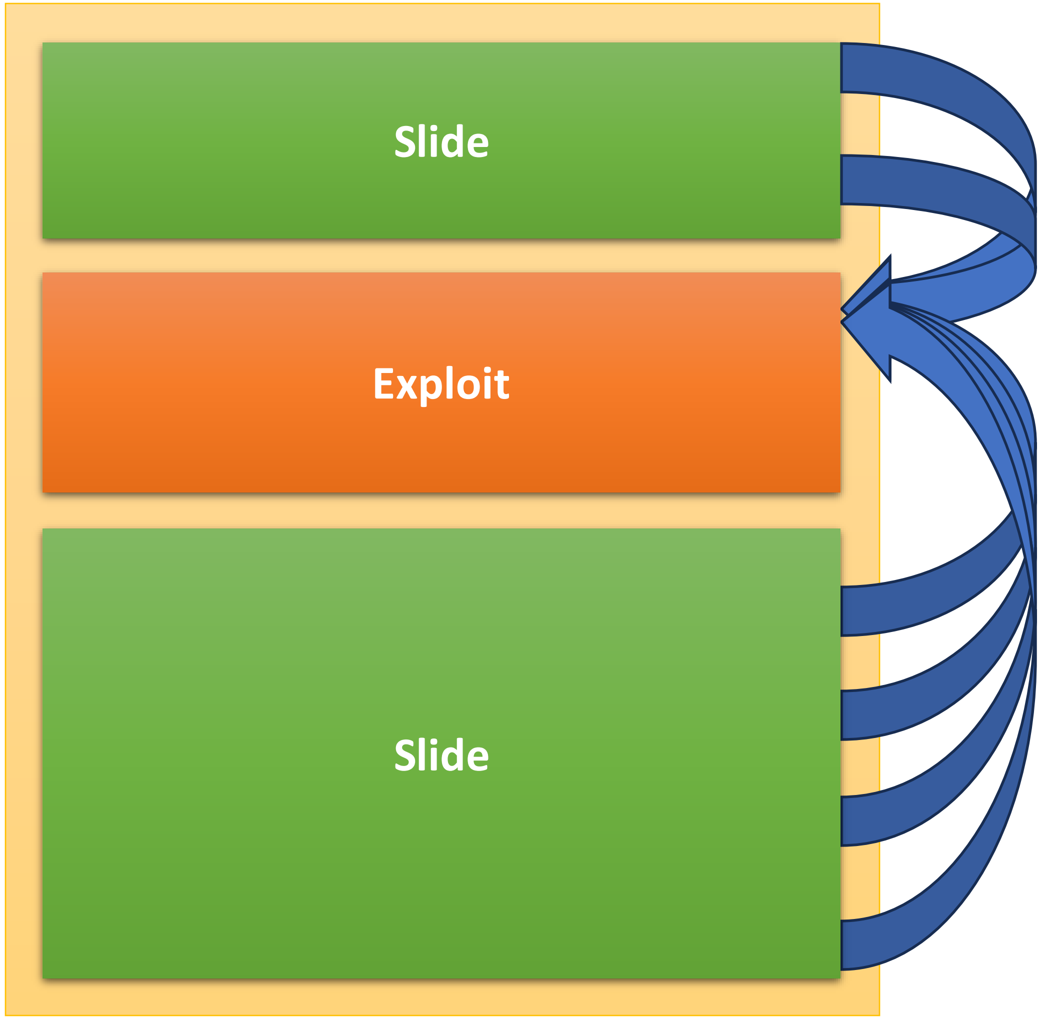
If the ECU doesn't emit stack traces, it is usually possible to source the same component or a similar one to program with a toy example firmware and find most fault parameters

Search Algorithm Optimizations

- Some parameters take longer to change (due to physical constraints)
- Some feedbacks correlate better with code execution than others
- Interrupt handlers are used as a feedback channel to rate glitches

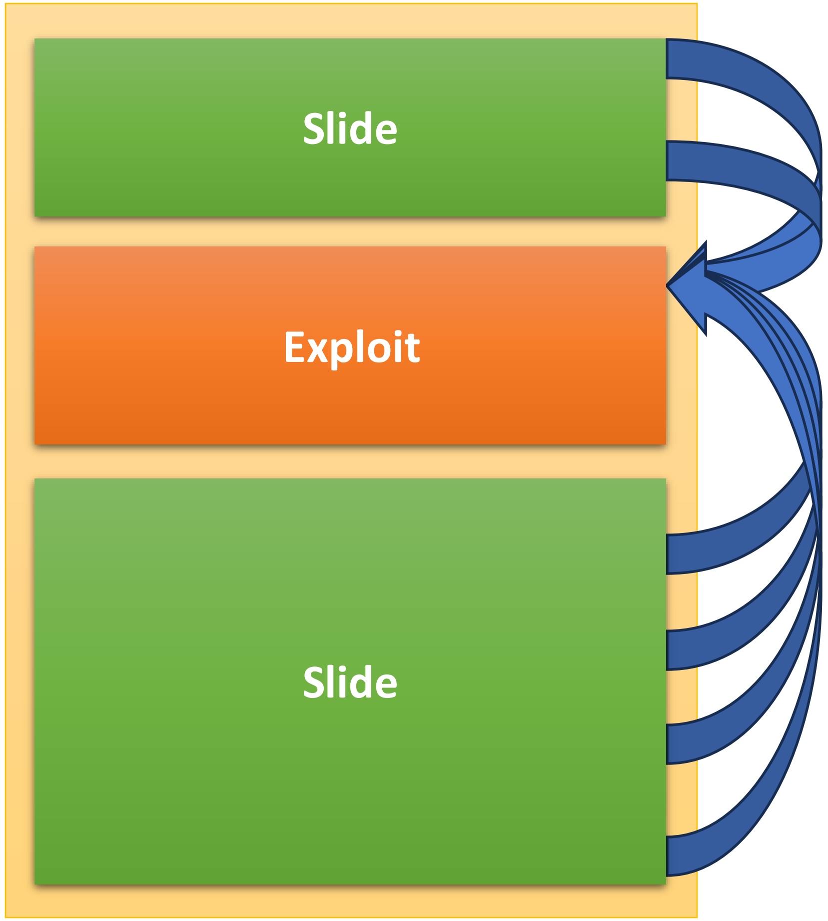


Ensure most of the unsigned firmware is composed of NOP slides / jumps to the exploit

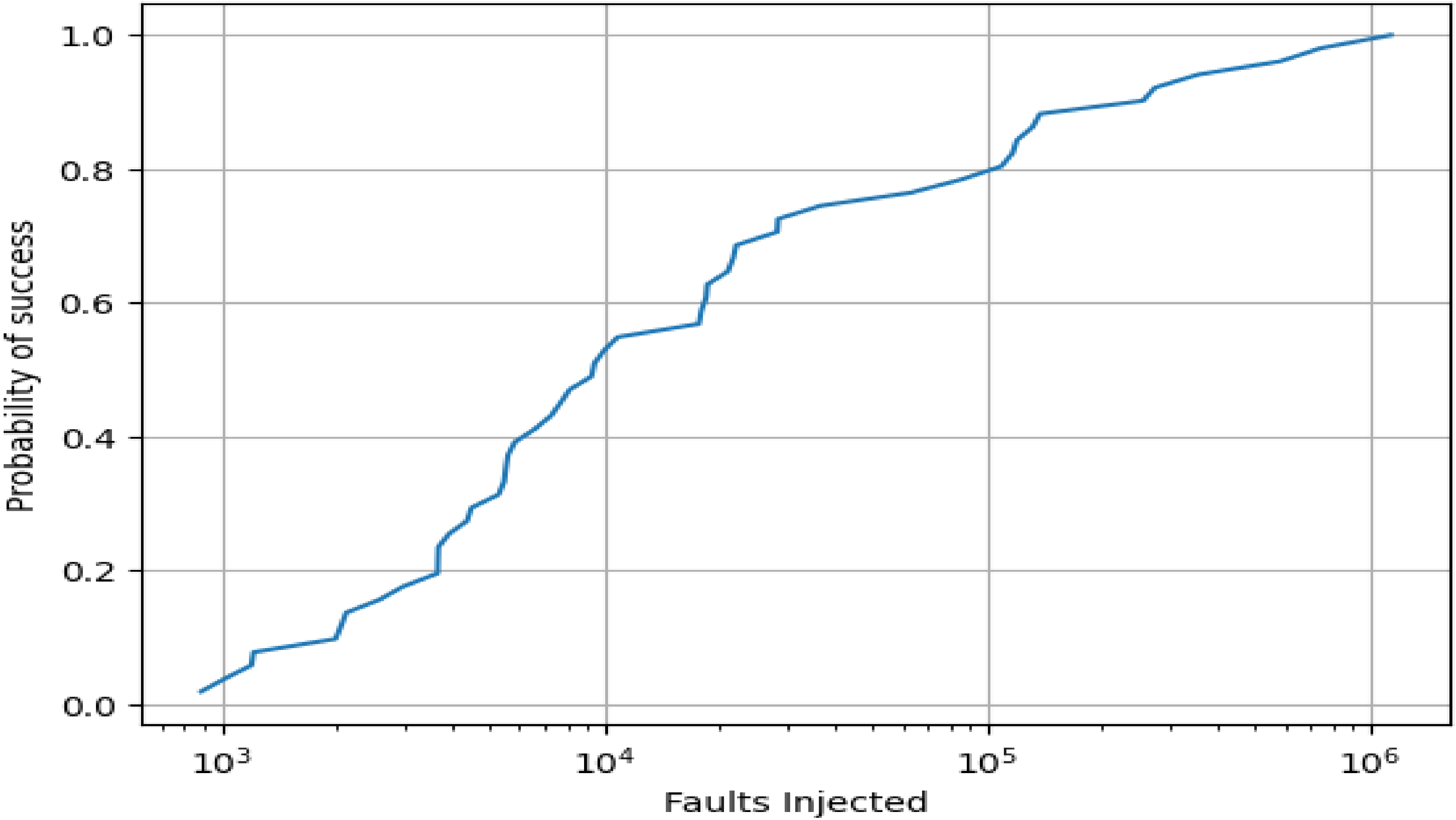


Structure of the exploit firmware

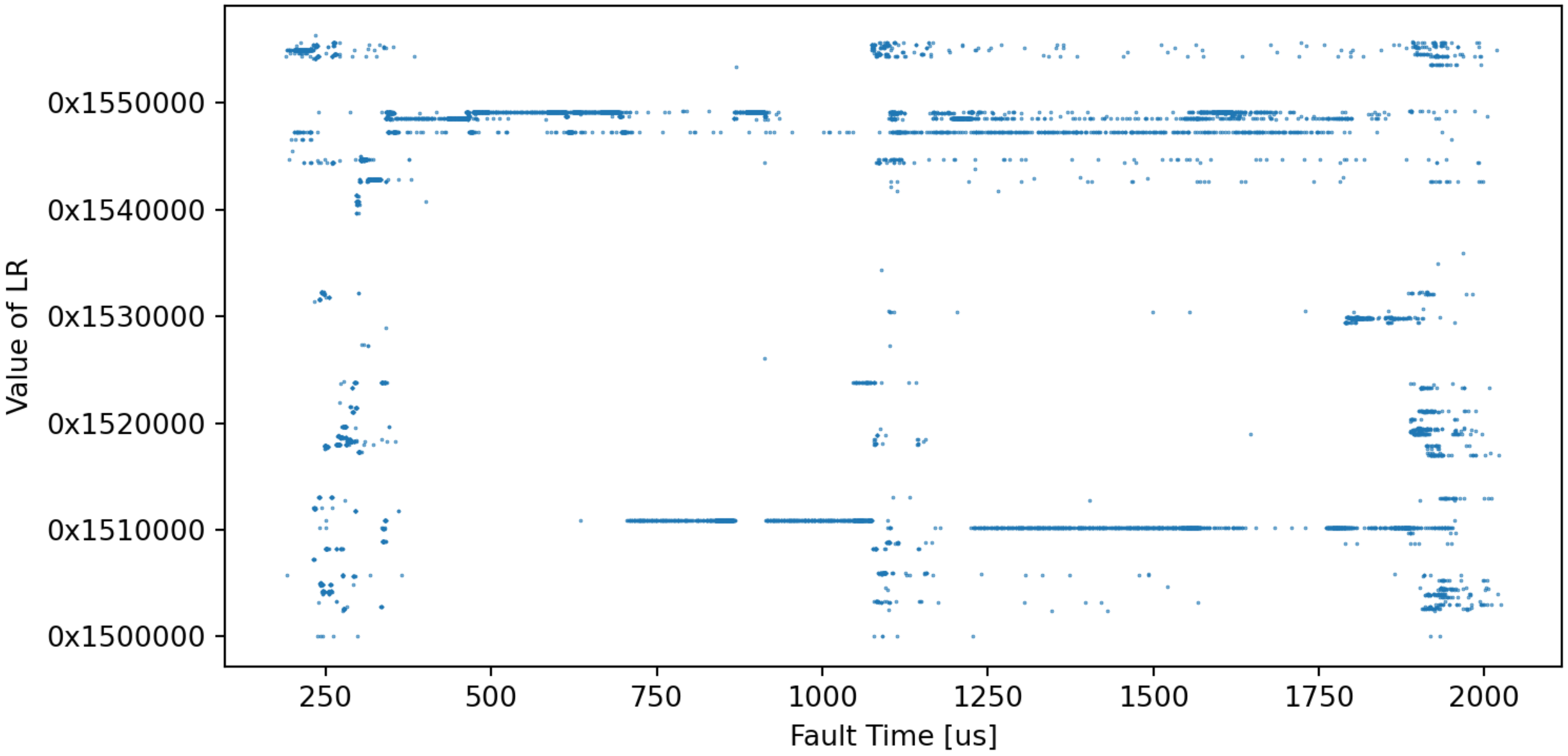
```
rept 1000
rept 113
        se_nop  // 2 bytes
endr
        e_b _start  // 4 bytes
endr
_start:
        // The actual exploit code is written here
rept 2000
rept 113
        se_nop  // 2 bytes

endr
        e_b _start  // 4 bytes
endr
```

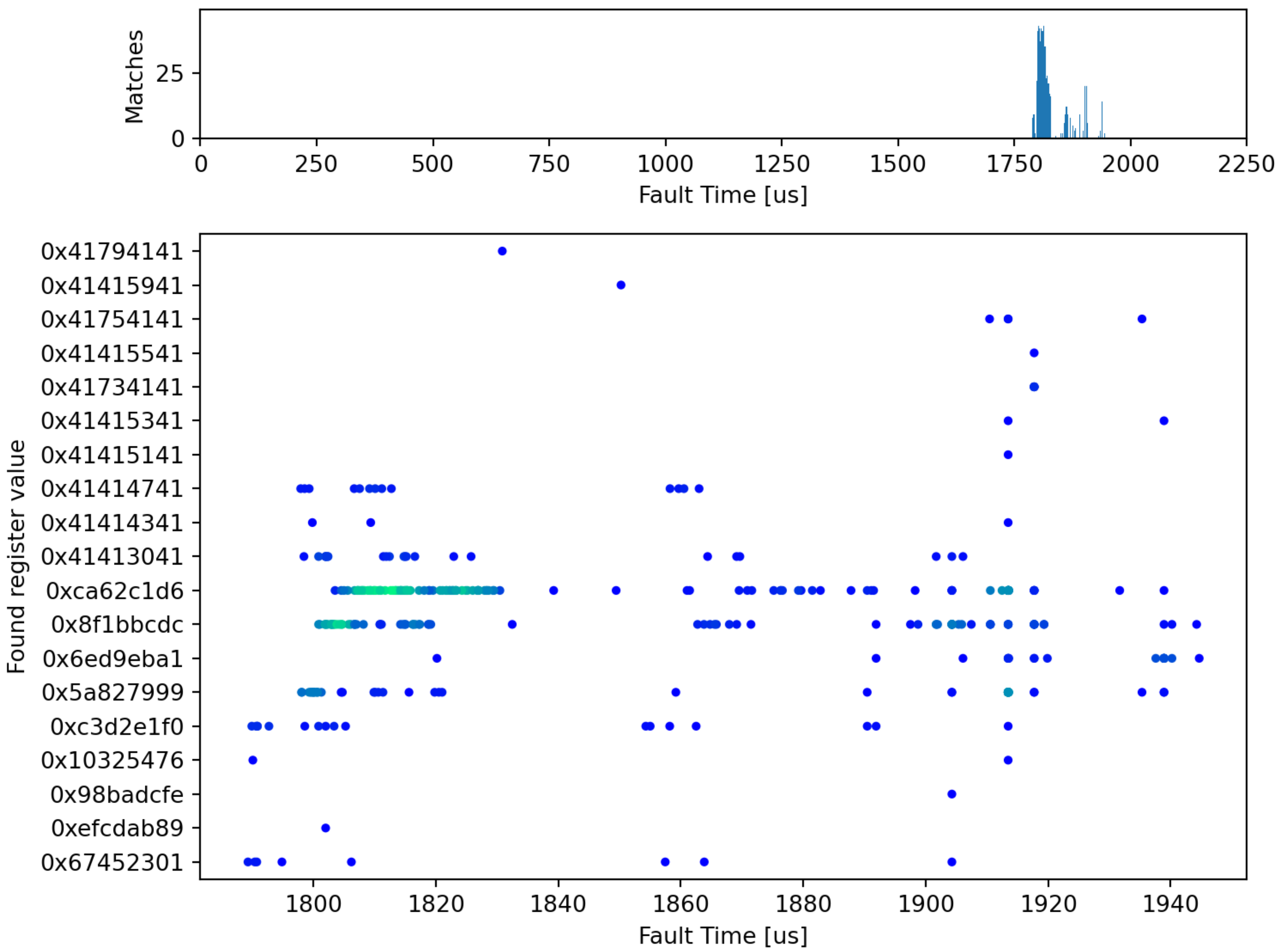


Search algorithm performance



Other interesting data leakage through faults




Matches
25
0
0
250
500
750
1000
1250
1500
1750
2000
2250
Fault Time [us]
Found register value
0x41794141
0x41415941
0x41754141
0x41415541
0x41734141
0x41415341
0x41415141
0x41414741
0x41414341
0x41413041
0xca62c1d6
0x8f1bbcdc
0x6ed9eba1
0x5a827999
0xc3d2e1f0
0x10325476
0x98badcfe
0xefcdab89
0x67452301
1800
1820
1840
1860
1880
1900
1920
1940
Fault Time [us]

Program Counter corruption on PPC

- The PPC VLE instruction set is commonly found on many ECU which are critical for security.
- In PPC, 00 is an invalid instruction, and the CPU will immediately fault if zeroes are fetched as an instruction.
- Moreover, in PPC, the program counter and linker register are special registers, so they can't be written by normal MOV or LD instructions
- Is PPC immune to fault injection attacks?

Disruption of Instruction Flow: Misaligning the Stack Pointer

Start of function

```
01109ad8 18 21 06 e0    e_stwu     r1,-0x20(r1)
01109adc 00 80          se_mflr    r0
01109ade 1b a1 09 14    e_stmw     r29,0x14(r1)
01109ae2 d9 01          se_stw     r0,0x9(r1)
```

⋮

```
01109b1c 1b a1 08 14    e_lmw      r29,0x14(r1)
01109b20 c9 01          se_lwz     r0,0x9(r1)
01109b22 00 90          se_mtlr    r0
01109b24 21 f1          se_addi    r1,0x20
01109b26 00 04          se_blr
```

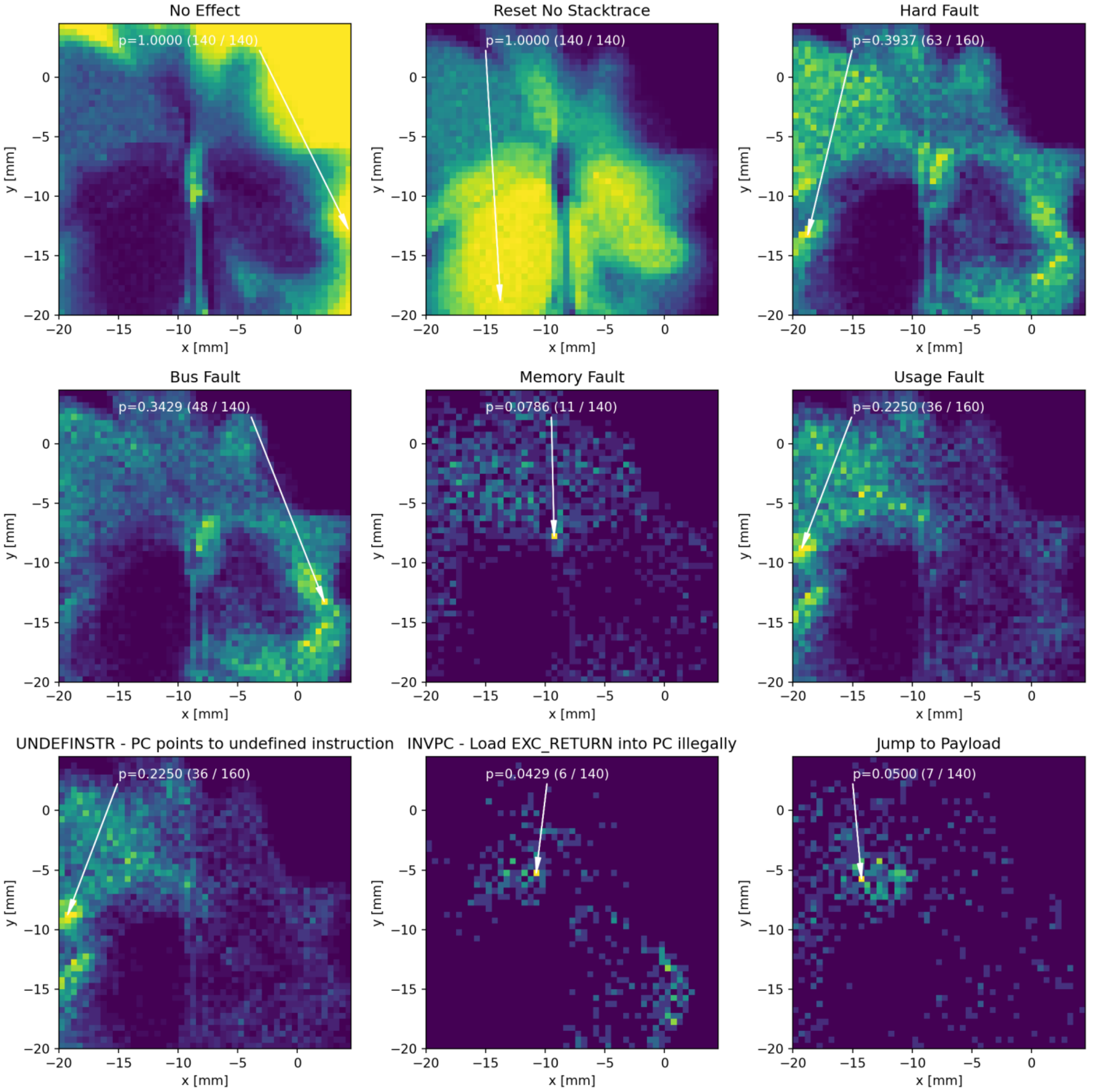
End of function

Exploitation

- Dumping memory
- JTAG
- Writing to Flash Memory
- Access to HSM API
- MPC / SPC Processors:
  - Manipulation of DCF Records (Chip Configuration)

How about ARM?

- Next generation of ECU Processors will be ARM
- Way-higher likelihood of PC corruption



Mitigation

- Use Memory Protection Units (MPU) => W^X
- Disable execution early in boot process to minimize attack surface
- Running the Bootloader in HSM might help:
  - Execution from functionally separated section of flash memory
  - Documentation for HSMs is kept secret, making exploit development harder
- ISO14229 (UDS) Software Update Process needs to be revised

Discussion

- UDS Protocol is broken in respect to fault injection
- We have Encrypted firmwares, that make the attack difficult
- It's a design flaw of UDS, adapatable to a wide range of ECUs
- No reverse engineering is required
- (Maybe HW-Reversing)
- Algorithms can be trained on EvalBoards and adapted to ECUs
- Attack can be automated
- PC corruption attack on PPC
- Attack was demonstrated on three different Gateway ECUs
- Different Processors, different OEMs, different Firmware, different Bootloader

Summary

- Efficient fault injection attacks demonstrated for code execution on real-world targets.
- EFISSA enables feasible black-box attacks
- Code injection into victim device's flash allows unauthorized execution via EMFI
- Higher success probability with larger programmable flash.
- Attack successful within minutes without knowledge of target software.
- Reproducible on multiple ECUs with minimal code changes.
- Cheap, available equipment; easy automation.
- Algorithm (EFISSA) reduces fault finding time from weeks to <1 hour.

Disclosure

- April 2022 major German OEMs were informed

Thank you for your patience!

dissecto GmbH
Franz-Mayer-Str. 1
93053 Regensburg

+ 49 941 4629 7370

contact-us@dissec.to