Decoding EM-FI Attacks:

Lessons Learned from Glitching the GigaDevice GD32F407

Jonathan Andersson & Thanos Kaliyanakis

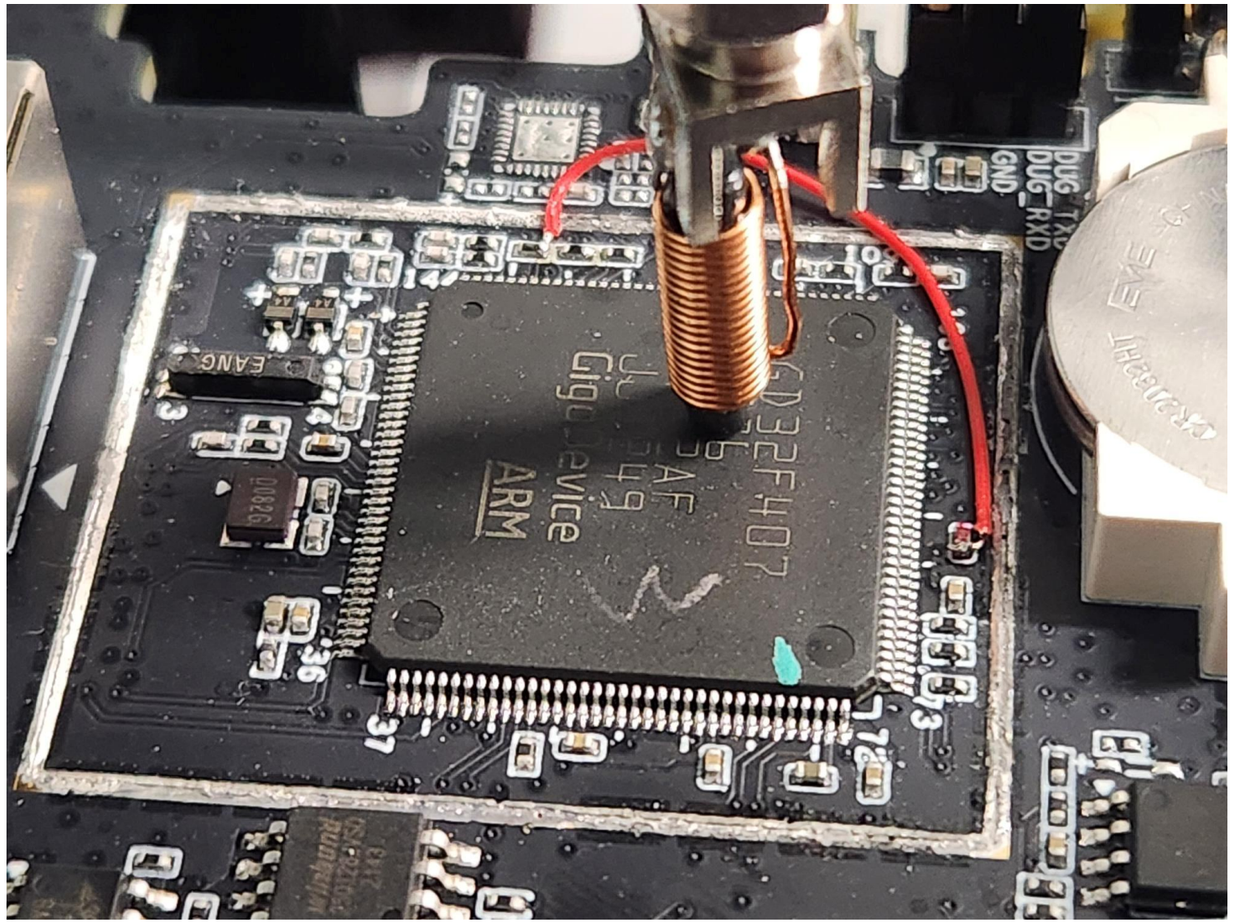
black hat EUROPE 2024

Jonathan Andersson
Sr. Manager
Advanced Security Research Group
Trend Micro ZDI

Thanos Kaliyanakis
Vulnerability Researcher
Advanced Security Research Group
Trend Micro ZDI

Agenda

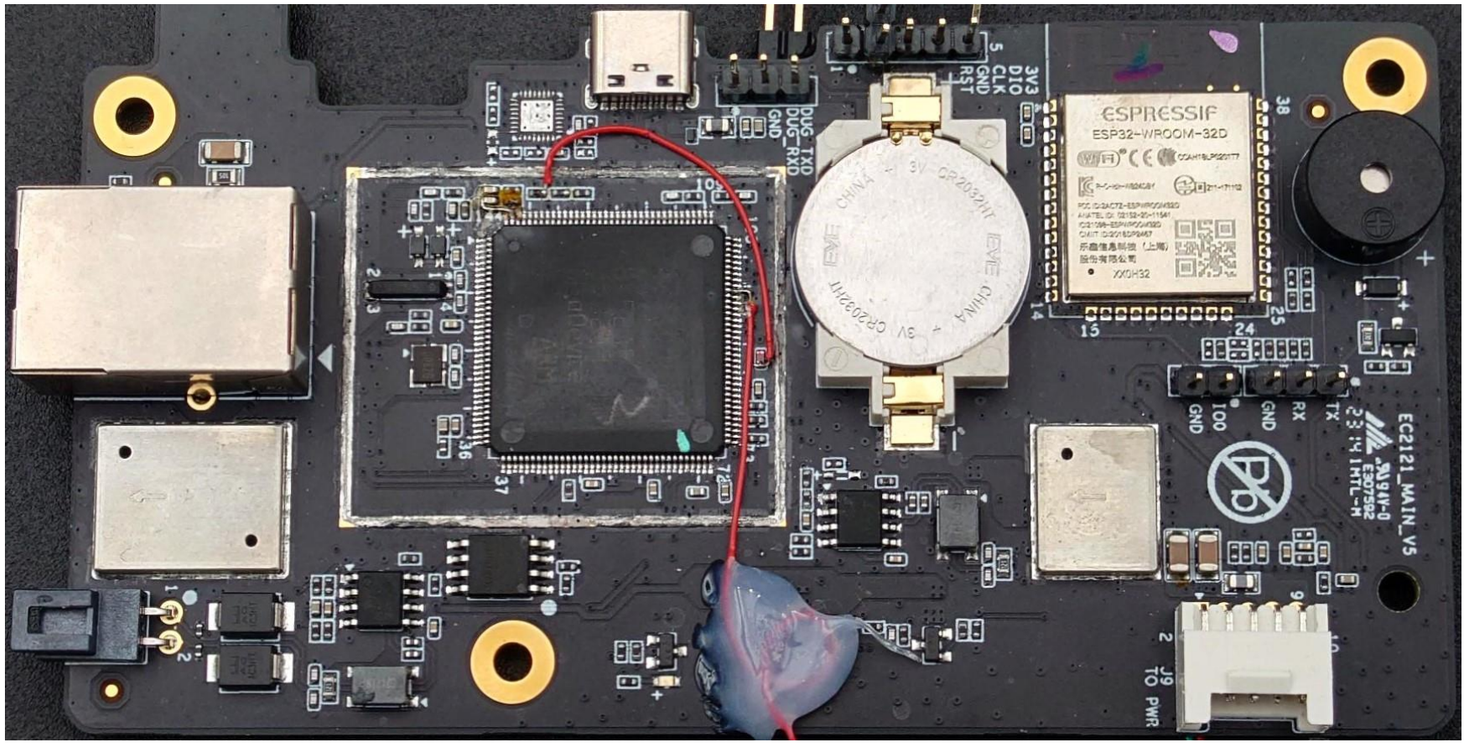
- Introduction & Background
- The Rig
- Getting Started
- The Attack
- Calibration
- Perfecting the Glitch
- Attack Results
- Mitigations
- Conclusions



Introduction & Background

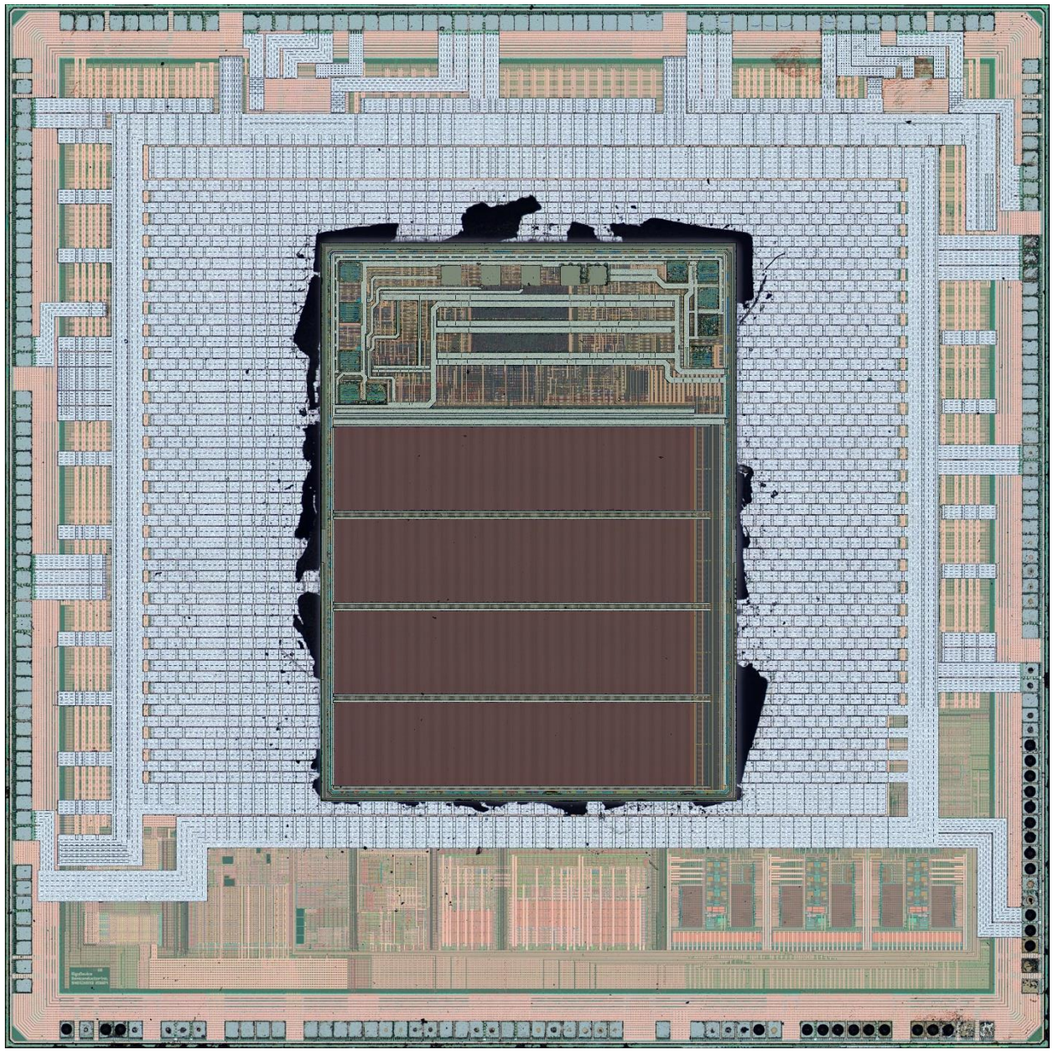
Introduction

- Why the GD32F407?
- Why fault injection?
- Why EM-FI?

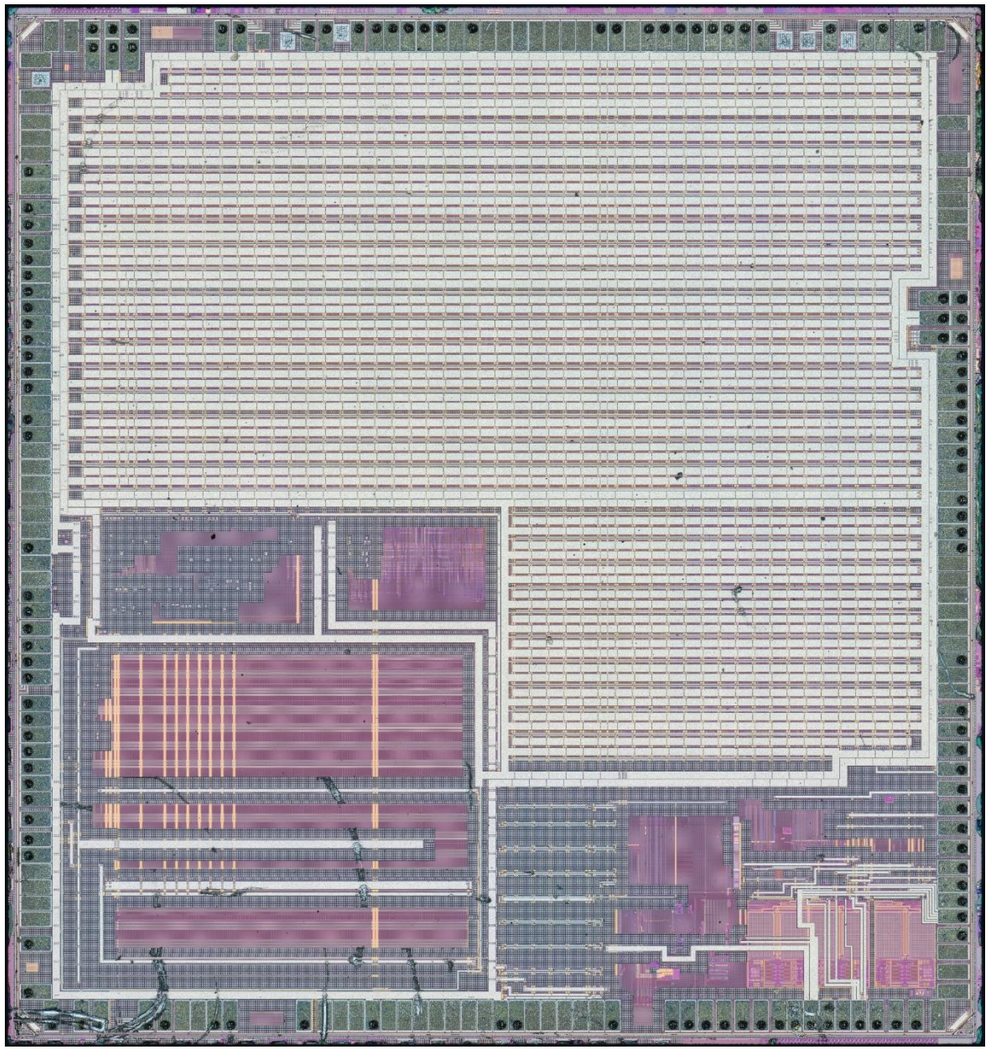


Autel MaxiCharger

GigaDevice vs STMicro - 32F407



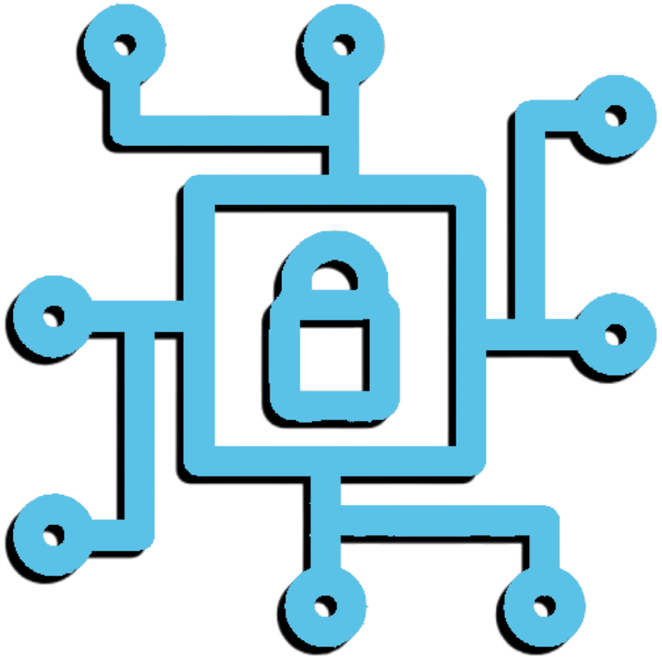
GD32F407



ST32F407

Read Out Protection Levels

- None / RDP 0
  - No restrictions
- Low / RDP 1
  - Flash accessible only in flash boot mode
  - Flash disabled when SWD attached
  - Can be reverted to None / RDP 0 but flash gets erased
- High / RDP 2
  - SWD cannot attach
  - Only flash mode boot allowed
  - No reversion back to lower security levels possible

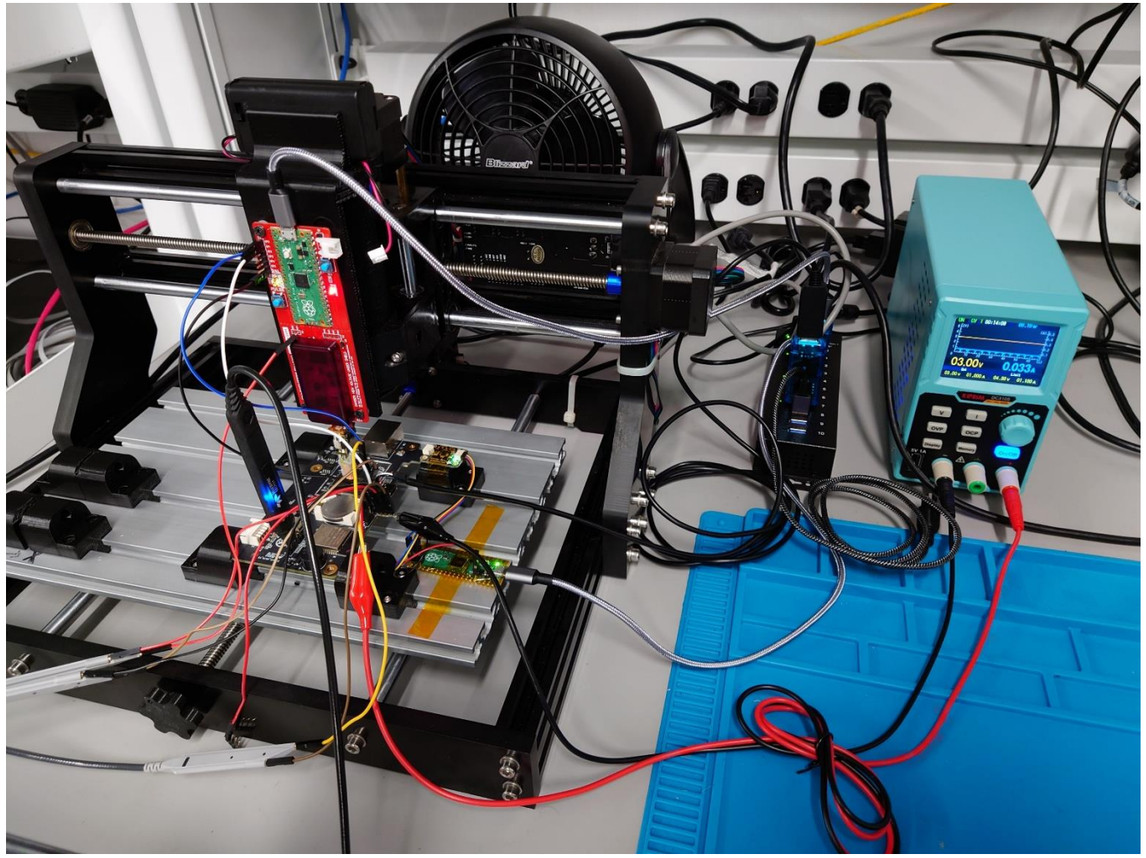


Lock by Puspa Kusuma CC BY 3.0 / blue with shadow

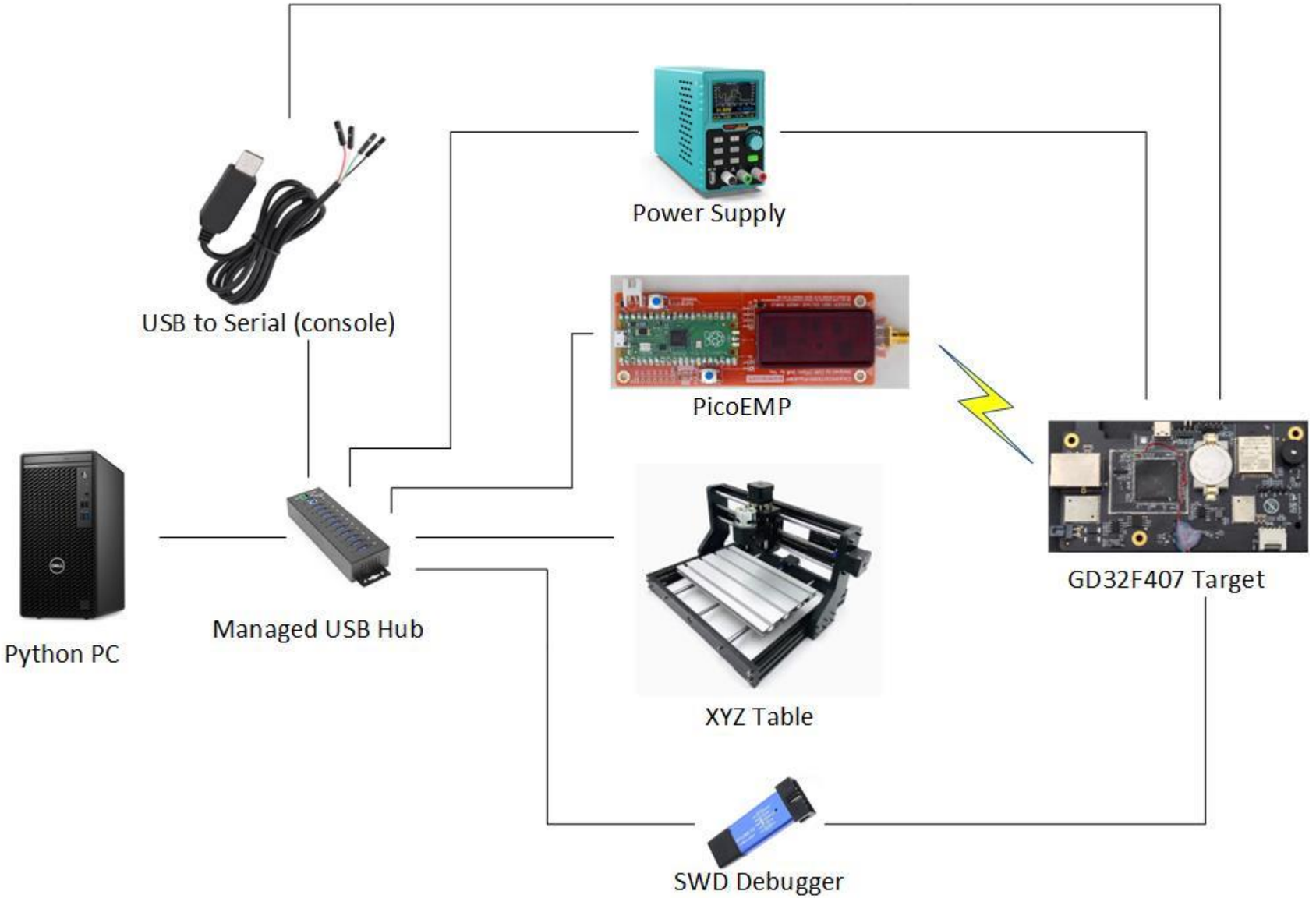
The Rig

The Rig (~$600 USD)

- ChipSHOUTER PicoEMP with firmware additions
- Managed USB hub with individual power control
- XYZ table (G-Code)
- Programmable power supply
- SWD debugger (OpenOCD / GDB)
- USB to serial (console)
- 3D printed parts (PicoEMP & PCB mounts)
- Custom python code driving the rig



The Rig



The Rig – Purchase Links

PicoEMP $100

XYZ Table $137

Power Supply $140

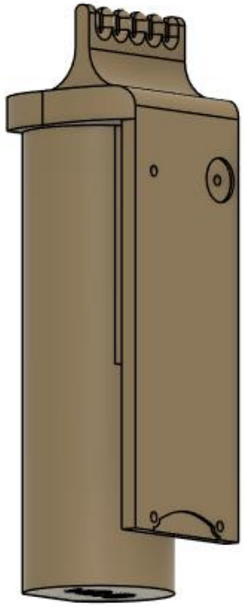
Managed USB Hub $176

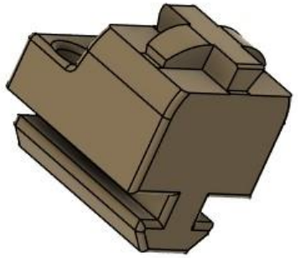
USB to serial $14

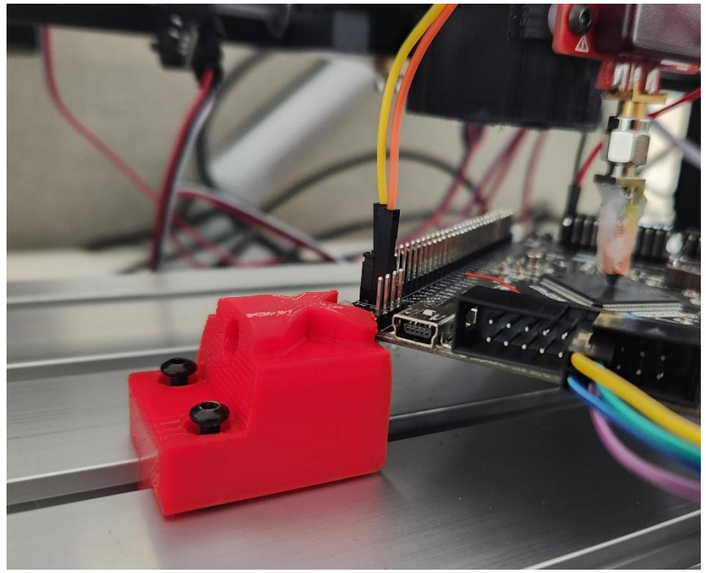
SWD Debugger $10

Custom 3D Printed Parts

- PicoEMP & PCB mount 3D models on www.zerodayinitiative.com/blog soon…







- Limit switch mounts (Thingiverse)

ChipSHOUTER PicoEMP – DIY

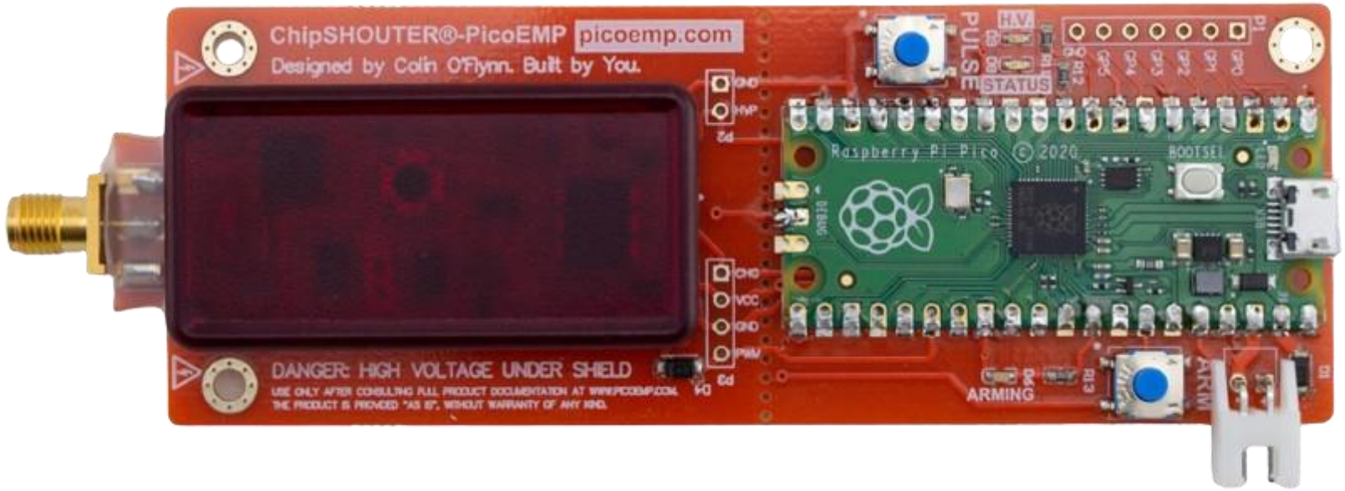
- https://github.com/newaetech/chipshouter-picoemp
- PCB + ~$90 in parts
- Required DigiKey and Mouser BOMs:

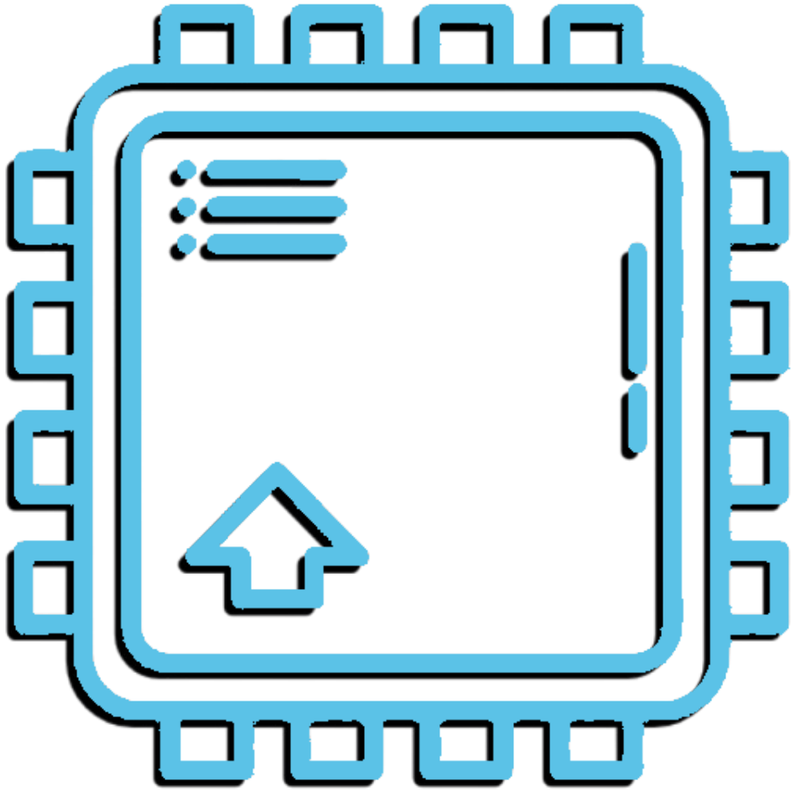
DigiKey Cart

Mouser Cart

PicoEMP Firmware Additions

- Added an all-in-one (4 into 1) command to maximize glitch rate
- Added deterministic pulse counting trigger in PIO
  - Used for basic serial triggering functionality
- Posted to original git soon…





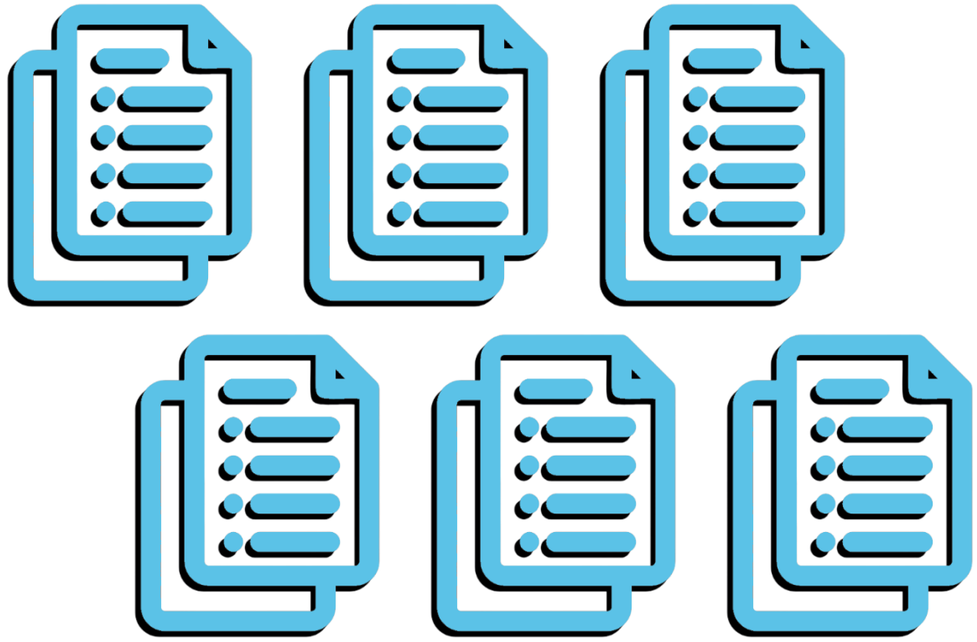
chipset by pictranoosa CC BY 3.0 / blue with shadow

Getting Started

First, Educate Yourself

Target CPU & supporting hardware

- Datasheets
- Programming resource guides
- Security application notes
- Prior research



Document by Dicky Prayudawanto CC BY 3.0 / multiple blue with shadow

- Understand how HW security mitigations are supposed to work in detail…

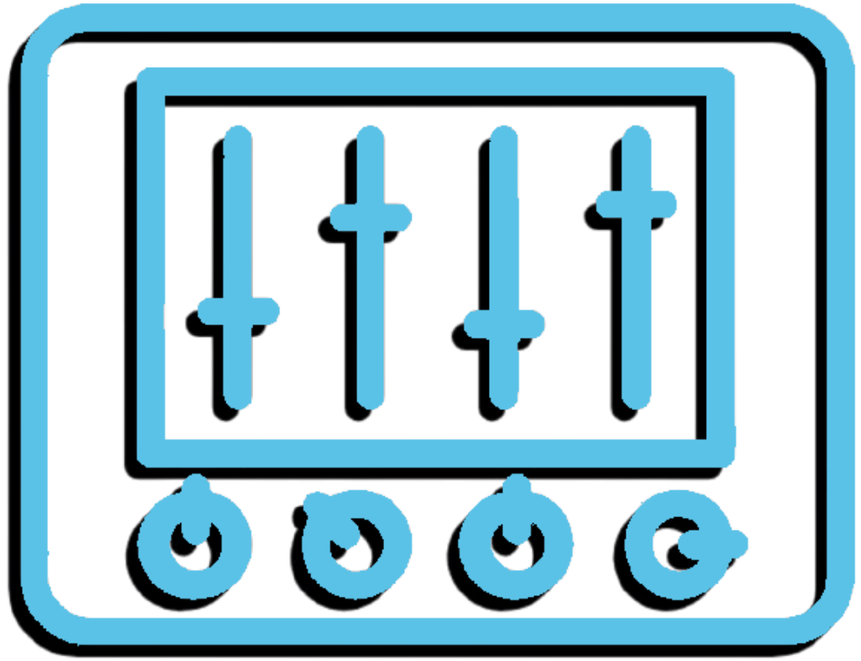
Enumerate Vendor Mitigations & Security Measures

- Don’t believe the datasheet, verify for yourself!
- Confirm the device’s behavior in various scenarios…
  - What can and can’t be done with JTAG/SWD in various modes?
  - What memories can be accessed and how?
  - Are register changes allowed?
  - Is stepping allowed?
- Examine the edge cases…
  - Reset / boot sequencing / debug attachment
  - Memory persistence?
  - Halt/run states

Detective by Visual Glow CC BY 3.0 / blue with shadow

What Dimensions to Control & Automate?

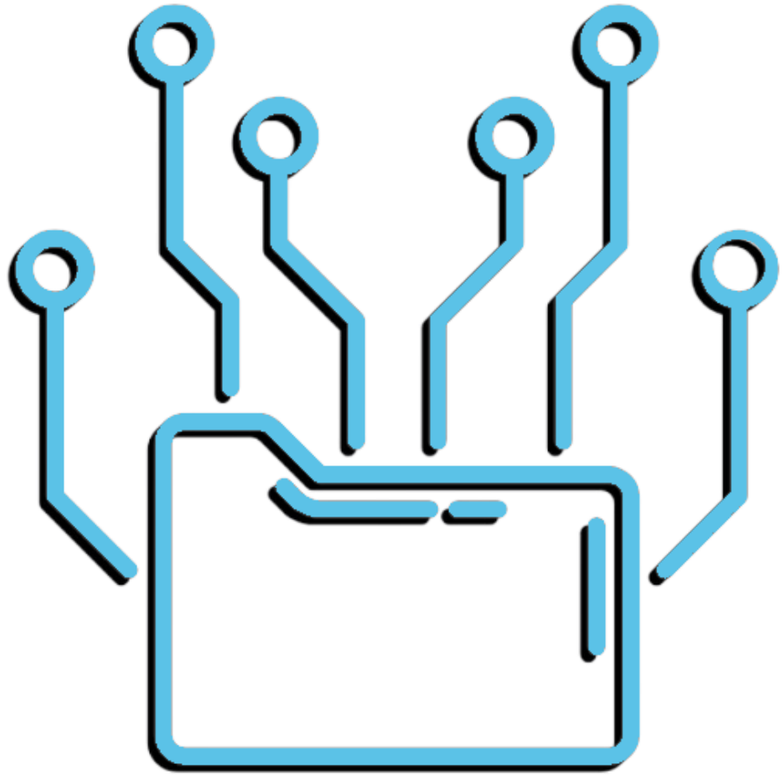
- Trigger
- Controlled parameters :
  - Target (V)oltage (VCC)
  - (D)elay
  - Spatial dimensions (X, Y, Z)
  - (P)ulse power
  - System state / device mode



control by SHAHAREA CC BY 3.0 / blue with shadow

What Data to Collect?

- Primary
  - Controlled dimensions
  - Target's current draw
  - Clock out / memory signals / other I/O
- Secondary
  - Program counter / CPU fault status
  - Console output
- Glitch loop / code timing!
- Use a scope to validate everything from the beginning
- Monitor ongoing operations with the scope!



data collection by Fahri CC BY 3.0 / blue with shadow

Coding & Ops

- Code
  - Isolate config parameters into a single file
  - Code for resumability
  - Use exception handling
- Log everything!
  - Keep a textual diary
  - Log what you did and plan to do
  - Document and analyze observations every run...
  - Document data thoroughly – today’s trash is tomorrow’s treasure!

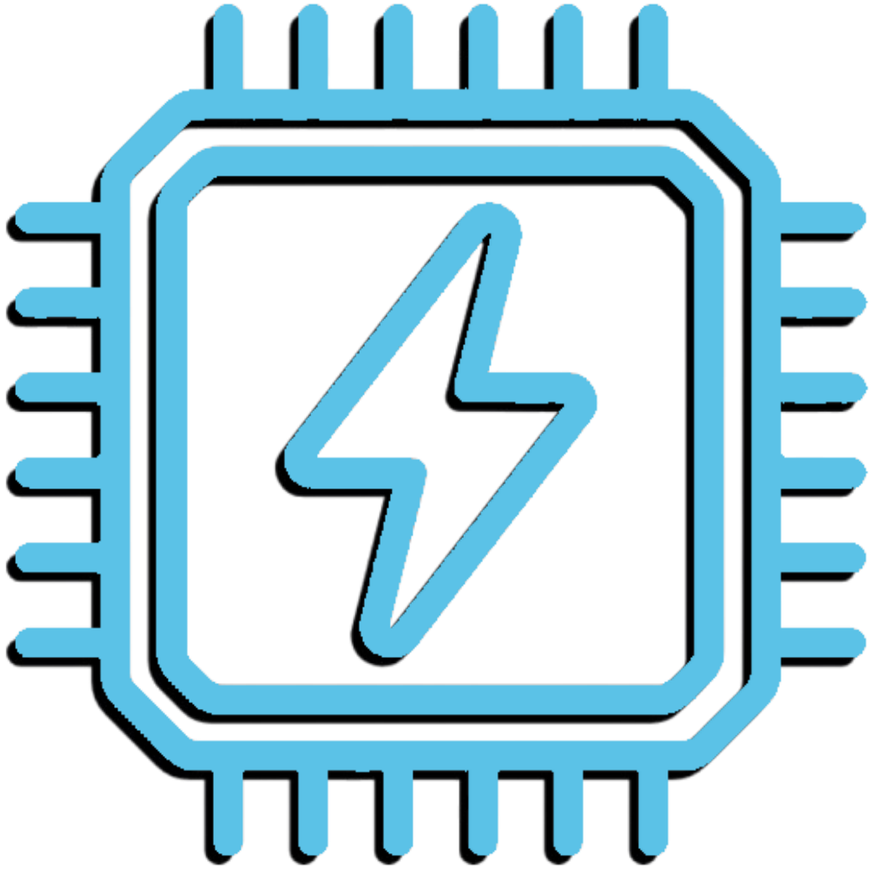
Recycle by Vicons Design CC BY 3.0 / blue with shadow

The Attack

The Attack – Bootloader Mode

- Extracted bootloader from unlocked GD32F407 via SWD
- Reversed in Ghidra
- Noted bootloader compatible with ST
- Located same Read Memory command

- The attack
  - Boot into bootloader
  - Issue Read Memory command (0x11)
  - Glitch
  - Wait for NACK (0x7F) or ACK (0x79)



processing power by Juicy Fish CC BY 3.0 / blue with shadow

The Attack – Read Memory Command

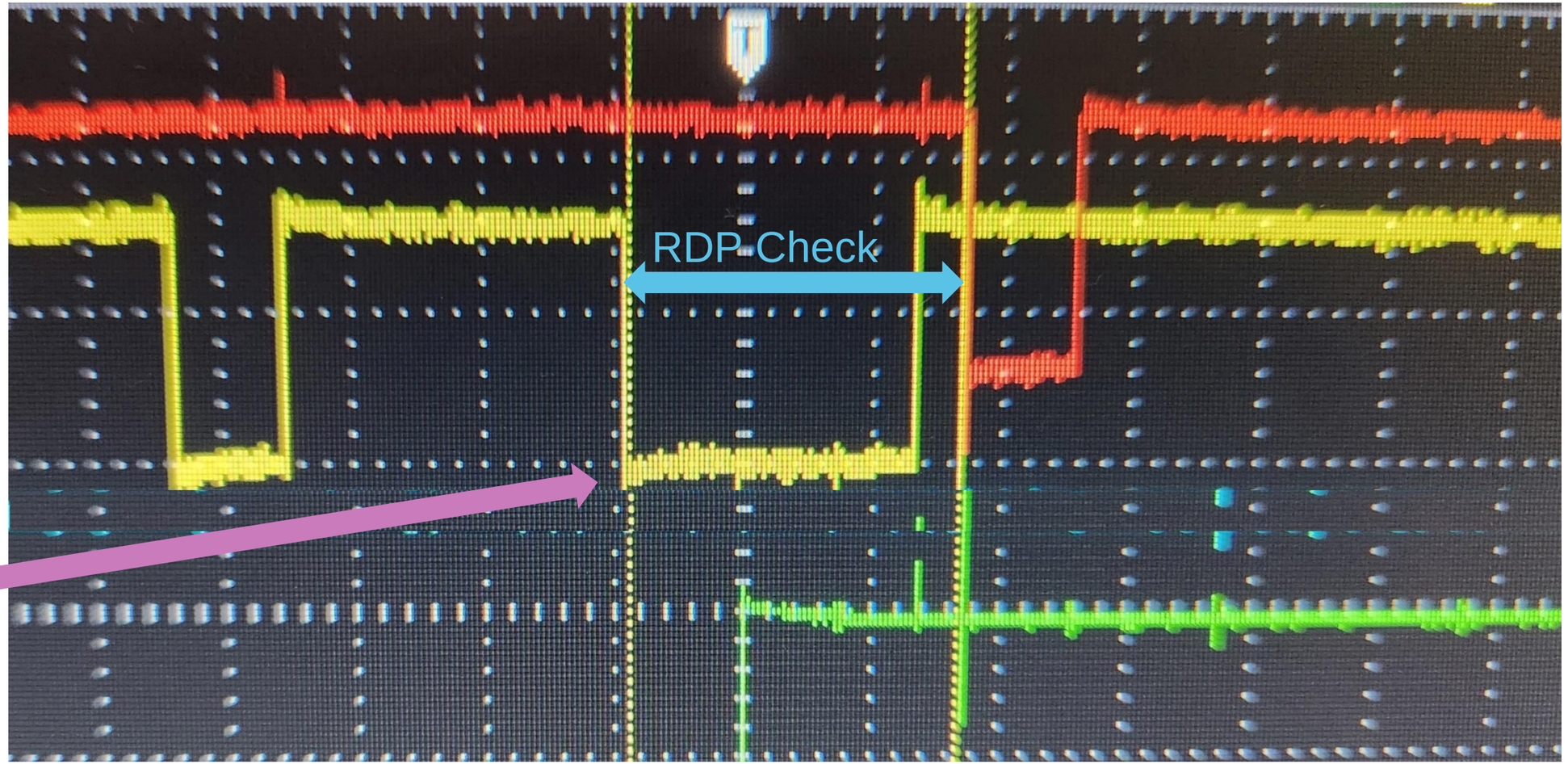
```
void Read_Memory_cmd_FUN_1fff0774(void)

{
  int p_address;
  int address_chk;
  uint opt_and_length;
  uint length_comp;

  opt_and_length = Peripherals::FLASH.OPTCR;
  if ((opt_and_length & 0xff00) == 0xaa00) {
                    /* Command ACK (1/4) */
    Serial_TX_byte_FUN_1fff3c04(0x79);
    p_address = get_address_FUN_1fff3af0();
```

```
                    /* Read Memory */
if (command == 0x11) {
  Read_Memory_cmd_FUN_1fff0774();
  goto Bootloader_Main_Loop_LAB_1fff46fa;
}
```

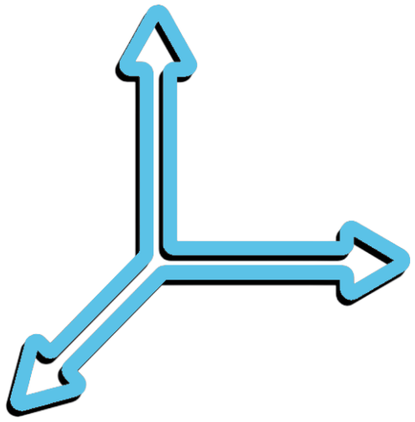
The Attack – An Example Glitch



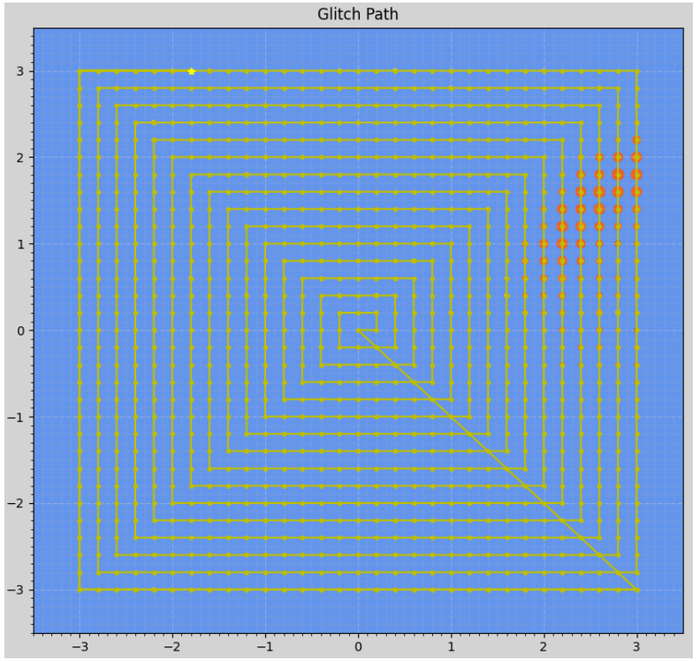

Calibration:
Establishing Glitch Influence

Scan For a Sensitive Spot…

- First, verify glitch production with a scope!
- Use *strong glitches*: Z = min & P = max & Target VCC = min (lift brownout protection pin)
  - Z is correlated to (P)ulse power and VCC
- Step size: our probe tip resolution is ~1-2 mm^2, use ~0.2mm steps
- Post-glitch look at current / brownouts / resets / PC / CPU faults / CLK OUT / memory bus
- Spiral scan…

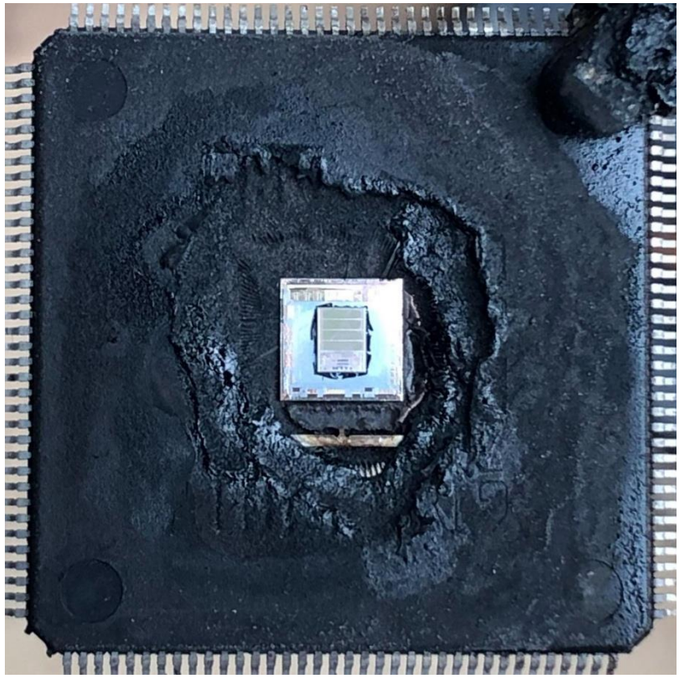


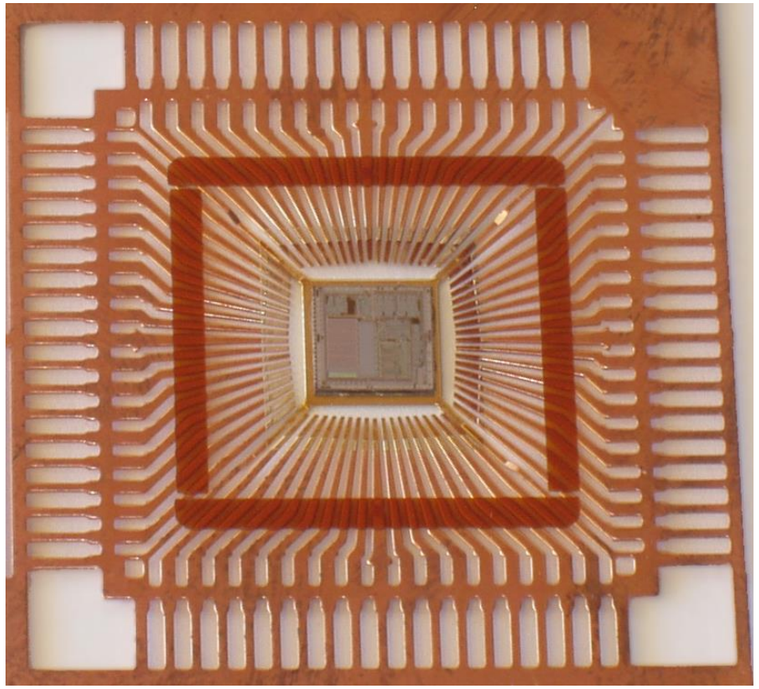
dimensions by Dániel Z. Aczél CC BY 3.0 / blue with shadow



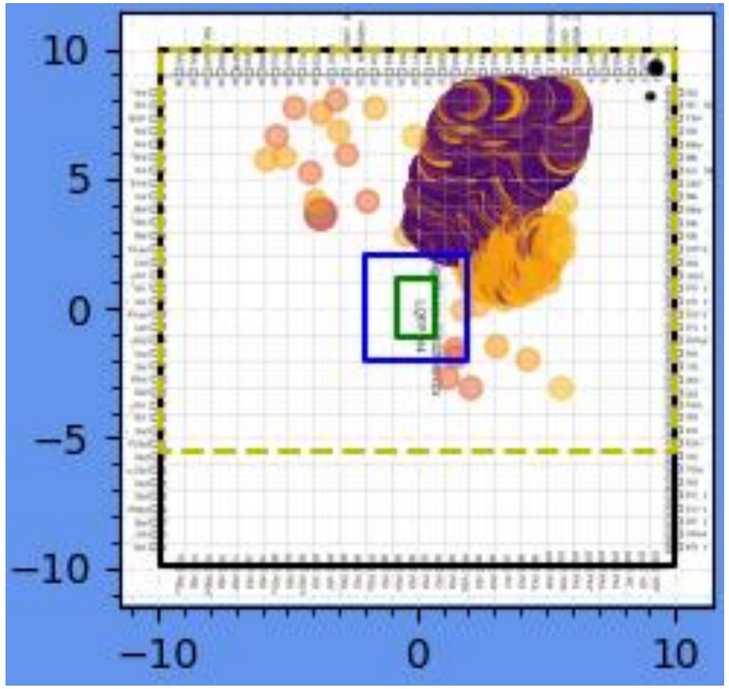
Why Spirals?

- *With QFP, all unique sensitivity behaviors can be observed in concentric rings ~25 to 50% the distance between die edge and pins*
- *Distinct sensitivity regions form radial lobes aligned with lead frame* – “flower petals”



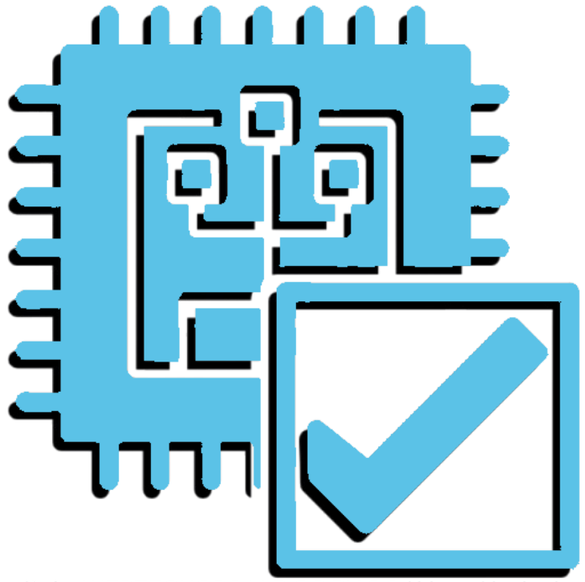


TQFP Leadframe © NobbiP. CC BY-SA 2.5 / Cropped



Calibrate Pulse Power & VCC

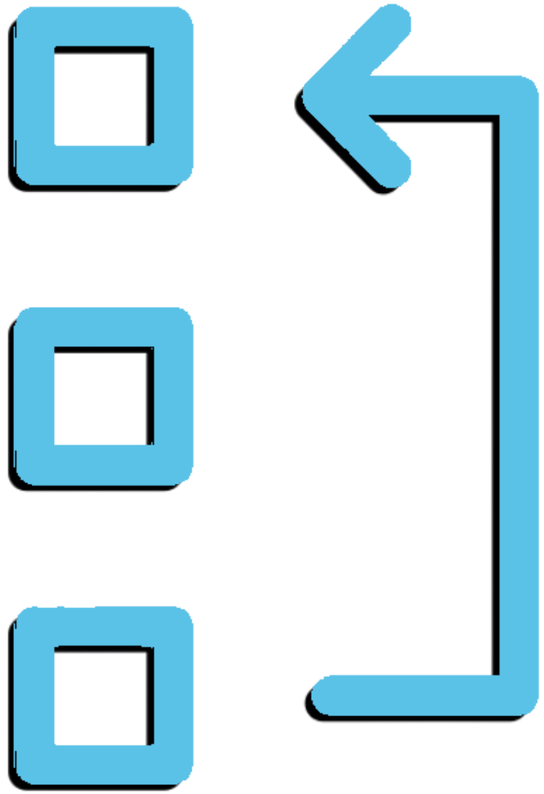
- Seek to one sensitive spot & collect data…
- Calibrate
  - Dial it back: *strong glitches* will preclude *useful* influence!
  - Independently reduce P & Increase VCC: *Note their ranges of influence*
  - Minimum VCC? - Consider % of zero current states vs continuation
    - 1.62V: 0.62% Brownout, 0.62% Unreadable PC
    - 1.61V: 33.33% , 38.27%
    - 1.51V: 53.70%, 98.15% (minimum operating voltage)
  - Sweet spot == ‘balanced’ mix of low current vs continuation
- Each sensitive area may have different optimum P & VCC



verify by WEBTECHOPS LLP CC BY 3.0 / blue with shadow

Program Counter Logging

- Sweet spots == modified code paths
- Characterizes level of glitch influence
  - Unreadable? (brownout or lockup – extreme influence)
  - In fault ISR? (significant influence)
  - Is PC still in expected code loop? (no or low influence)
- Also illuminates possible attacks…
  - In user writable RAM / buffer controllable by user?
  - Random place (existing useful code – get lucky?)

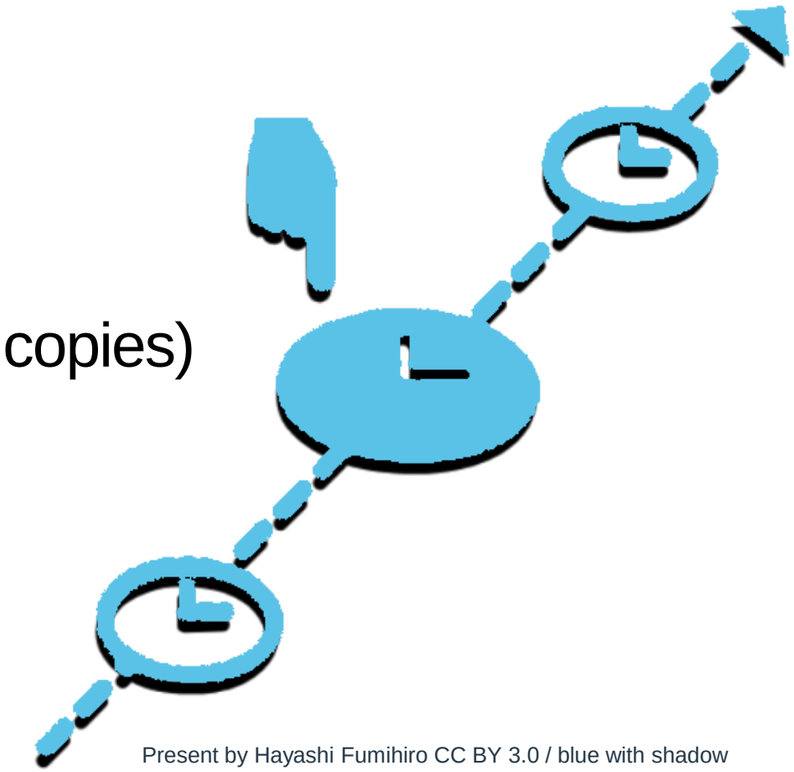


List by VERA CC BY 3.0 / blue with shadow


Perfecting the Glitch

Perfecting the Glitch – Trigger

- A stable trigger is key to reproducibility, repeatability, and progression
  - Trigger -> Delay -> Glitch: Hard timing is required on nS scale!
  - Manual triggering is useless
  - Reducing trigger jitter can reduce (D)elay window
- Sooner is better than later
  - Leverage boot code processes (register config, memory copies)
  - Multitasking / multiprocessing are your enemies!



Present by Hayashi Fumihiro CC BY 3.0 / blue with shadow

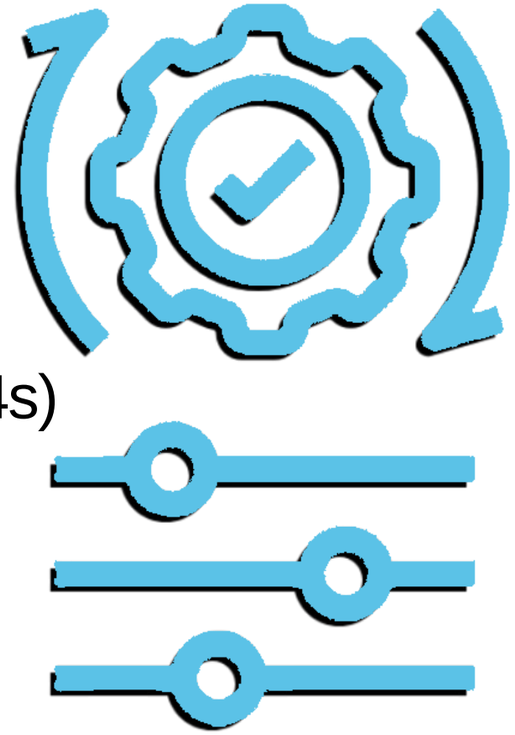
Perfecting the Glitch – Delay

- ARM fault status registers contain PC of fault (see Memfault's blog)
- *ARM faults associate (D)elay with exact location of a strong glitch's influence in code!*
  - Seek to one sensitive spot
  - Use *strong glitches* and log fault PCs…
  - Establish a range of (D)elay around your glitch target PC
- Without reading ARM fault status registers…
  - (D)elay range is much larger, increasing brute force time

Stopwatch by Andre Buand CC BY 3.0 / blue with shadow

Perfecting the Glitch – Optimization

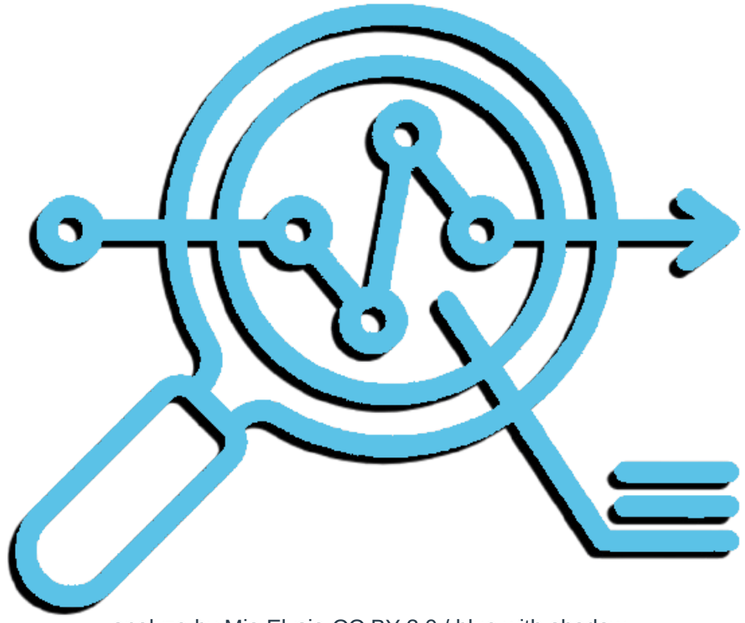
- Glitch rate optimization is key! (It's brute force, of course!)
- Reduce iterated parameter count and range
  - Collapse overlapping effects / correlated dimensions
    - Z & VCC can generally remain fixed and P can vary
- Classify glitch results faster (zero current vs protocol response)
- Minimize EMP charging time
  - PicoEMP: 5kHz @ 1.5% duty cycle (0.38s) or 8.7kHz @ 3.5% (0.24s)
- Parallelize operations (move while charging EMP)
- Optimize code and measure timing in logs



optimize by Kamin Ginkaew CC BY 3.0 / blue with shadow

Perfecting the Glitch – Analysis

- Visual analysis is a must!
- Is an observation unambiguous?
  - Frame in the context of SoC subsystems...
    - Reset: Triggered brownout vs jumped to ISR and triggered watchdog?
- Utilize an unlocked development board
  - Unlocked behavior vs locked
  - Extract and analyze available ROM (bootloader) code
  - Prior versions of firmware unlocked or published?



analyze by Mia Elysia CC BY 3.0 / blue with shadow

Perfecting the Glitch – Got Lost?

- Try a new known sensitive region
- Rescan and recalibrate when…
  - Lack of any glitch influence
  - Changing EM injector or probe tips
  - Changing or realigning targets
  - Observed anomalies
- Getting unstuck…
  - Precisely controlling variables key!
  - Is there a control or a measurement problem?
  - Measure more *or* more accurately…

lost by Bailey CC BY 3.0 / blue with shadow

Attack Results

The Attack – Data Analysis

Pin1 = Upper right

Main die

Flash die

Scanned area

Dot color = Current draw

ACK = Successful glitch

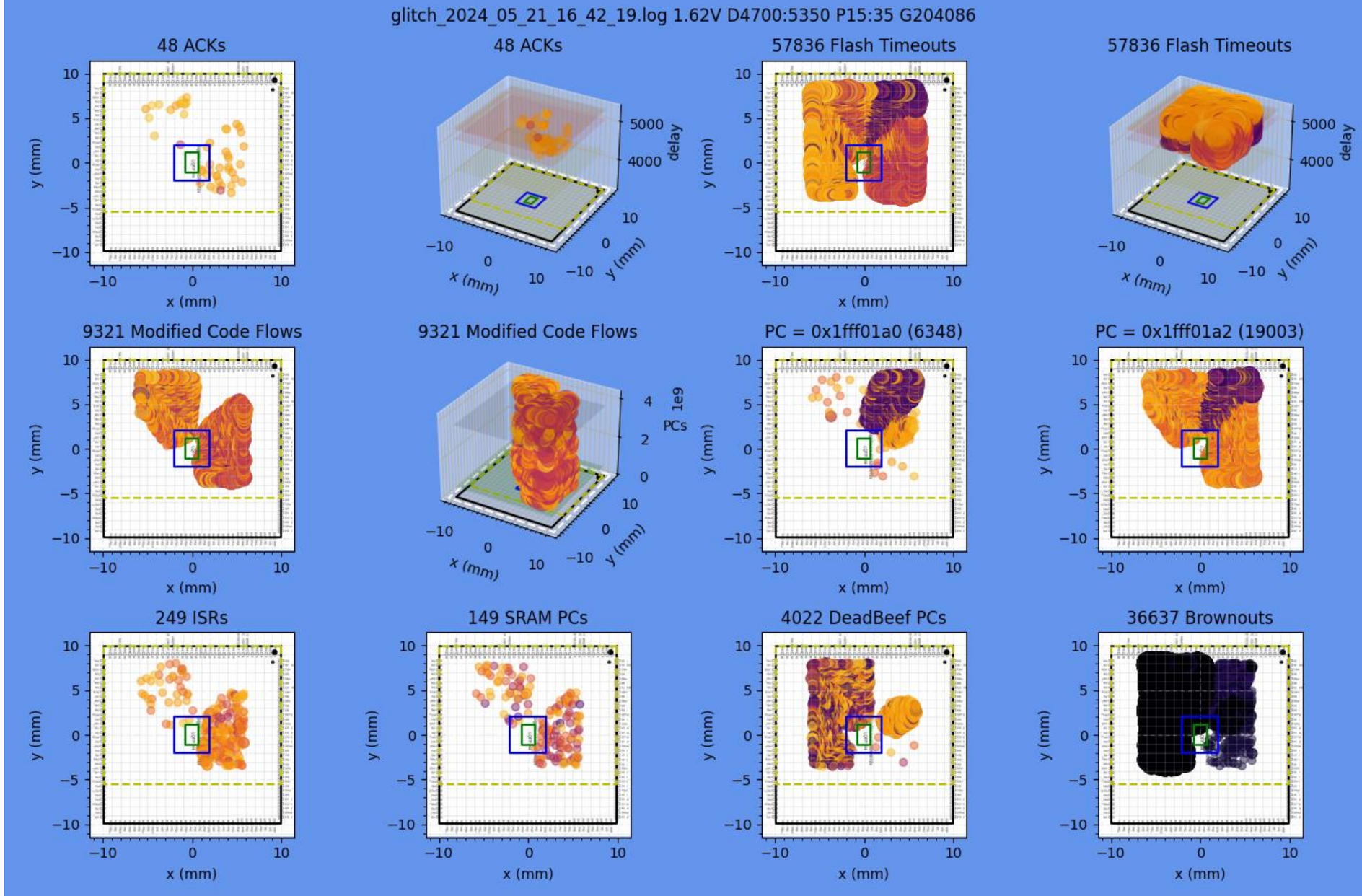
Flash Timeout = BL no response

0x1fff01a0 = NMI

0x1fff01a2 = Hard Fault

ISRs = Other IRQs

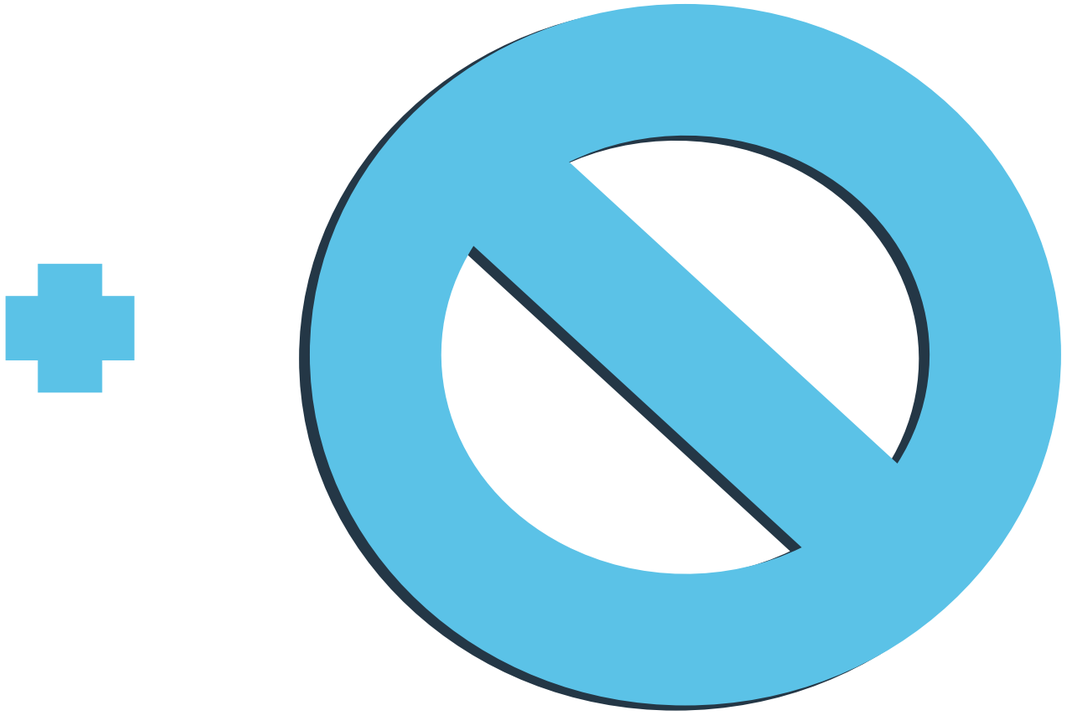
DeadBeef = Unreadable



The Attack – Results?

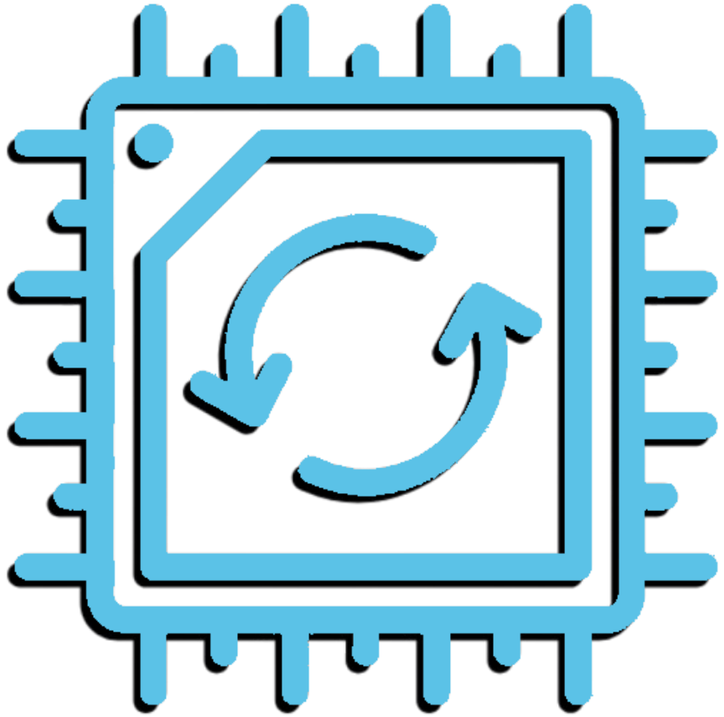
- ~10M glitches later, we successfully bypassed the security check…
- Bus Fault upon flash read (after Memory Read Length parameter is input)
- We confirmed the flash is disabled in bootloader mode, even without SWD attached

success by Edy Subiyanto CC BY 3.0 / blue with shadow



A Pivot to… The Second Attack – SRAM

- We Observed
  - SRAM contents persistent after HW reset
  - Logged some PC jumps to SRAM
- The Second Attack
  - NOP Slide + dump routine injected to SRAM via SWD
  - Reset and glitch early during reboot



chip by muhamad afiffudin CC BY 3.0 / blue with shadow

- Good data collection and a reliable, consistent rig made a pivot easy

The Second Attack – Save it All!

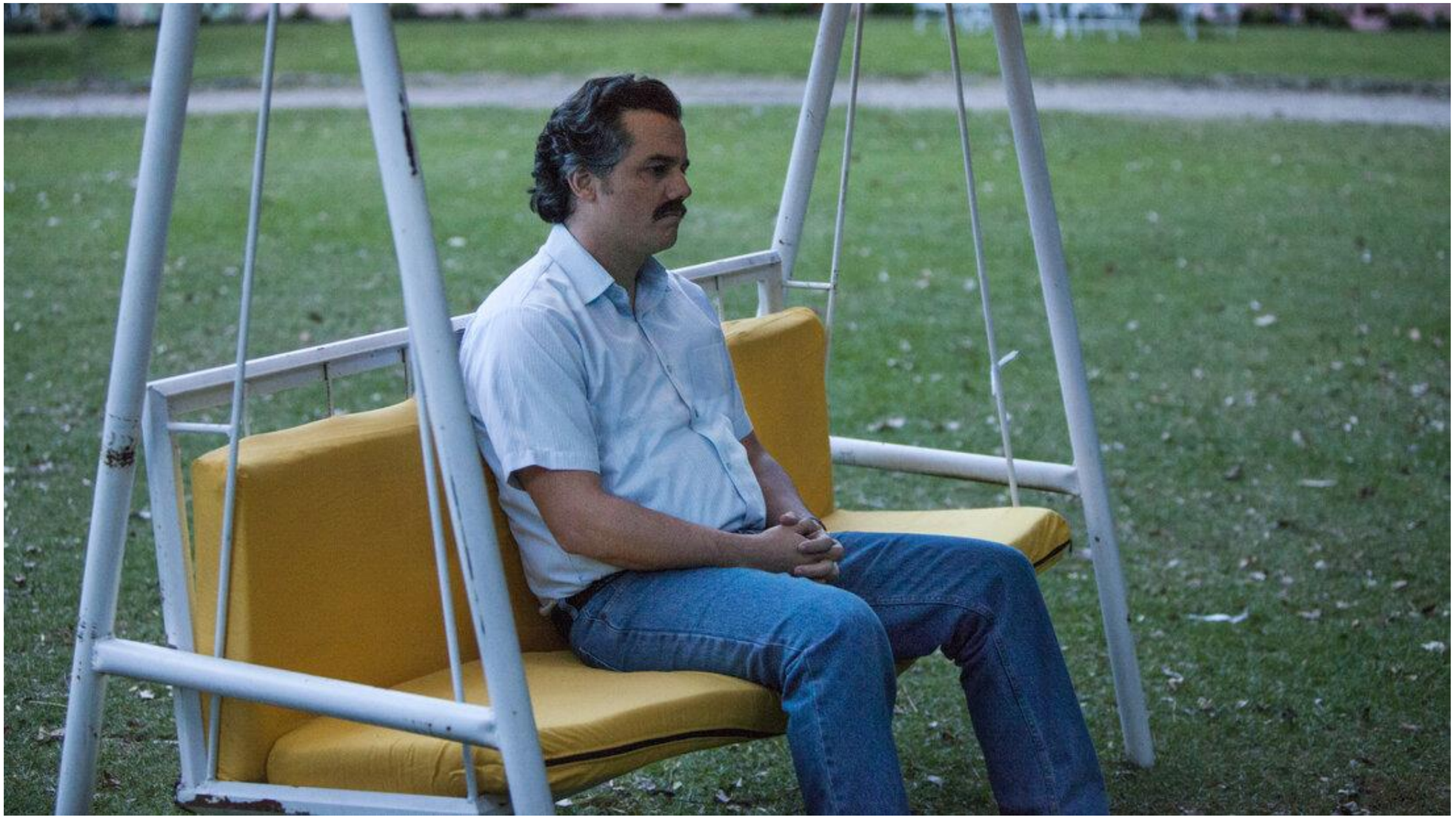
```c
// User Flash
uint32_t flash_address = 0x08000000;
uint32_t flash_end = 0x08100000;
while (flash_address < flash_end) {
    uint8_t byte = *(uint8_t *) flash_address;
    UART5_SendByte(byte);
    flash_address++;
}
```

```c
// Backup SRAM
uint32_t bk_sram_address = 0x40024000;
uint32_t bk_sram_end = 0x40025000;
while (bk_sram_address < bk_sram_end) {
    uint8_t byte = *(uint8_t *) bk_sram_address;
    UART5_SendByte(byte);
    bk_sram_address++;
}
```

```c
// RTC Registers
uint32_t RTC_address = 0x40002800;
uint32_t RTC_end = 0x4000289D;
while (RTC_address < RTC_end) {
    uint32_t word = *(uint32_t *) RTC_address;
    UART5_SendWord(word);
    RTC_address +=4;
}
```

Floppy by Vectorstall CC BY 3.0 / blue with shadow

The Second Attack – Wait for it…



The Second Attack – Success!

[2024-06-14 06:47:50,468] g: v2.00_x3.80_y-1.40_z0.00_d7.833 B0.000+ Z0.022 P288 Vb2.01 Ib0.050 F0.648 Vc2.01 Ic0.050 J0.669- PC080052c2 LPC00000000 LLR08004285 SOK -R2.200 G4.151 C4.325

[2024-06-14 06:47:51,636] NOP Slide detected, extracting flash...

Log Key:

v = Voltage

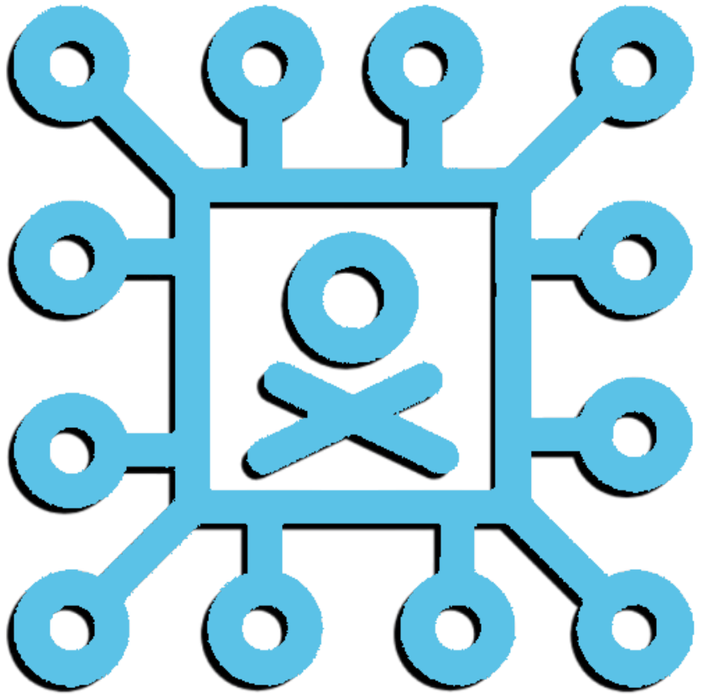
x, y, z = Position (mm) (origin is center of chip, pin 1 upper right)

d = Delay from trigger (ms) (RST release)

P = Pulse power (ns) (time EMP switch is closed)

V[x] = Voltage at stage x

I[x] = Current at stage x



circuit hack by Hai Studio CC BY 3.0 / blue with shadow

A Bonus – RDP1 Bypass

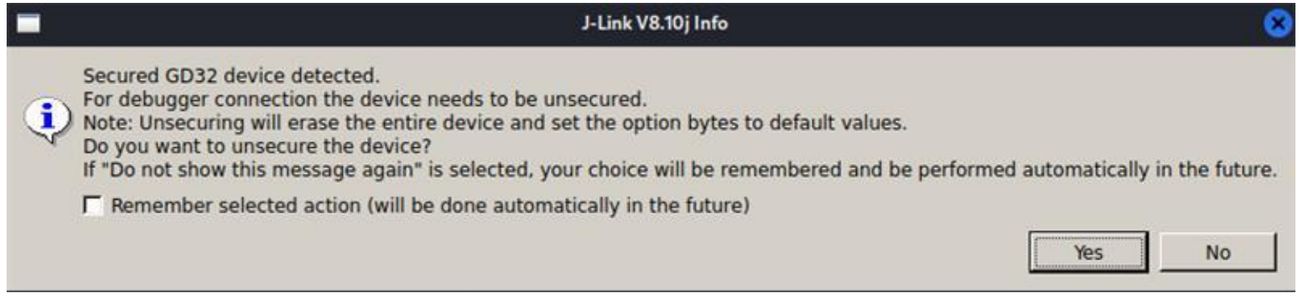
- Bypass found!
  - Boot in Flash Mode
  - Connect to device with RDP1 protections via SWD
  - Inject firmware extraction code into SRAM
  - Set PC to start of extraction code
  - Disconnect SWD
  - Execution continues and flash is extracted!
- Important to exercise mitigation edge cases!
  - Vary sequence, timing, modes, parameters, etc

bonus by Yosua Bungaran CC BY 3.0 / blue with shadow

Mitigations

Bypass Mitigations?

- GD *may* patch in follow-on IC tape-outs, but EM-FI mitigations are not so easy!
- Segger (J-Link) pseudo-mitigated this bypass, but don't be fooled!
  - J-Link v7.94m (03/06/2024) introduced a *feature* that blocks all SWD access to RDP1 locked GD devices without a forced erase!
  - Prior J-Link applications allow bootloader code access, SRAM R/W, stepping, and execution!
  - Appears to be implemented in the host sw, not the J-Link fw...
  - Obfuscation != Security (not cool, Segger!)
- Lesson: try old fw versions and alternate debug tools…



smile by Evgeni Moryakov CC BY 3.0 / blue with shadow

Conclusions

Conclusions

- Fault injection is a brute force exercise, but…
- There are optimizations and methods that will save you *significant* time
  - P, Z, VCC correlation
  - PicoEMP charge rate optimization
  - ARM fault status PC & D association
- Prepare yourself and don’t be discouraged!

hack by Sunardi CC BY 3.0 / blue with shadow

Sound Bytes

- The GigaDevice GD32F407 in RDP1 is vulnerable to a simple bypass
- The GigaDevice GD32F407 in RDP1 is vulnerable to EM-FI
- EM-FI: Calibrate, Scan, Tune, Analyze, Document, Optimize, Repeat…

Sound Wave by Yayat Dayat CC BY 3.0 / blue with shadow


Q&A

Contact Us

- Jonathan_Andersson@trendmicro.com
- Thanos_Kaliyanakis@trendmicro.com

https://www.zerodayinitiative.com

@thezdi, @thezdibugs

@thezdi

https://www.youtube.com/c/ZeroDayInitiative

PGP https://www.zerodayinitiative.com/documents/zdi-pgp-key.asc
Fingerprint: 743F 60DB 46EA C4A0 1F7D B545 8088 FEDF 9A5F D228