

Today I'm going to talk about how fault injection attacks can be used to exploit bug free code in embedded systems.

Fault Injection is a local attack on hardware, which requires physical access.

For reasons I'll cover later, it's often the foot-in-the-door which can lead to remote attacks later.

This talk is intended as an introduction to the subject, aimed at those with little or no knowledge of FI.

Abstract

Embedded systems are everywhere, automating more and more of our everyday lives. Our cars, phones, games consoles, industrial controllers and IoT devices increasingly require security mechanisms to protect their security configurations, and in some cases, stored secrets, such as cryptographic keys, debug/flash protection access mechanisms, firmware images, and AI models. For a long time, local, physical attacks on general purpose microcontrollers were considered out of scope during threat analysis, but the increase in value of breaking the device security protections, the decrease in cost of the attacks, and the increase in awareness of such attacks, means that we're in a transitional state regarding protection against fault-injection.

Description

This talk will introduce attendees to fault-injection, a local attack category which is often used as the first step in the attack chain for embedded systems, and in some cases can also lead to remote attacks. It will cover the techniques which attackers use to generate

security violations such as bypassing read protection, secure boot, or debug protection in embedded systems, even when the code is completely free of bugs. You will learn about the attacker motivations, tools and techniques, as well as the methods used to harden devices against these attacks, and how increased public awareness, certification, and regulation is changing the landscape. You will see how the cost of the equipment needed is often very low, and learn how you can begin your “glitching” journey for under £20. We will look at the fault-injection mitigations added in the Raspberry Pi Pico 2, and consider their efficacy - there is currently a \$20,000 bug bounty available for breaking these protections leading to recovery of a secret stored in the One-Time-Programmable flash memory. We shall also touch upon side-channel analysis, which can recover cryptographic keys in use through measurement and analysis of tiny power fluctuations, or even by using a coil to pick up electro-magnetic emanations.

Keywords:

Embedded Systems

Microcontrollers

Hardware Attacks

Fault-Injection

Voltage Fault Injection (VFI)

Electro-Magnetic Fault Injection (EMFI)

Clock Fault Injection (CFI)

Risk Assessment

Threat Modelling

Automotive

Industrial Control Systems

IoT

Mitigation Strategies

Raspberry Pi Pico 2

Side-Channel Analysis

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

Today we'll cover:

Who I am

What fault injection is

Different types of FI attacks

Examples of why and where FI attacks are used

How this can compromise security goals

Live demo

Mitigation techniques

If time, other hardware attacks

What we'll cover



- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

:~\$ whoami

@barsteward
Bluesky: @barsteward.bsky.social
Mastodon: @barsteward@infosec.exchange
RIP Twitter: @barsteward

I'm employed to torture silicon chips until they give up their secrets, or agree that they work for me now.

Hardware penetration testing of claims about

- Secure Boot
- Flash Read Protection
- Debug Protection
- (and side-channel analysis resistance)

This talk does not represent the views of my employer!



I work for a silicon vendor who sells microcontrollers for use in cars, industrial control systems, IoT devices etc.

My job is focused on hardware penetration testing, to determine whether the microcontrollers we sell are resistant to fault injection and side channel analysis attacks. Basically, I'm running the internal red team for hardware attacks on our products. My team represents all you hackers out there in the audience; we try to identify vulnerabilities before you lot get a chance to.

In hardware manuals and marketing material, various claims are made about the normal operation of the devices, and it's my job to try to disprove those claims, so fixes can be made before we release the chip, and to teach others in the organisation about how to protect against attacks on these.

These claims are things like

- *Secure Boot, where only signed images will be executed*
- *Flash Read Protection, where it should only be possible to read the flash memory contents after authentication, or in certain lifecycle management states*
- *Debug Protection, where debugging should only be possible after authentication, or in certain lifecycle management states*

The image is what Deepai.org thinks I do in the lab!

Morph image used under licence: <https://creativecommons.org/licenses/by/4.0/>

Image source:

Science Museum Group. Model of 'Morph'. 1999-5162 Science Museum Group Collection Online. Accessed 21 November 2024.

<https://collection.sciencemuseumgroup.org.uk/objects/co8180635/model-of-morph>.

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

What The F (I) ?

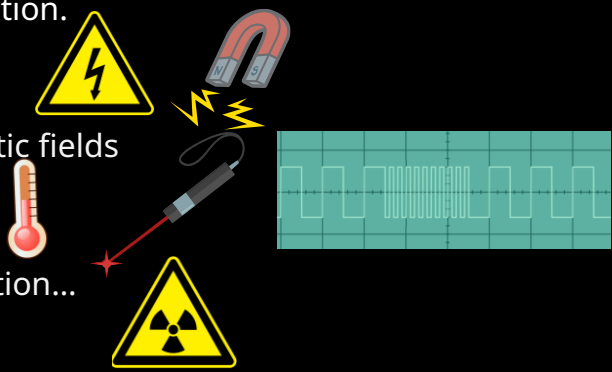
What is Fault Injection anyway?

What is Fault Injection (FI)?

Fault Injection (FI) is a class of hardware attacks in which the device is stressed in an unusual way to make it malfunction.

Extremes of

- Voltage
- Electromagnetic fields
- Clock speed
- Temperature
- Light
- Ionizing radiation...



The diagram illustrates various fault injection methods targeting a hardware device. It features a central green circuit board with a white square wave pattern. Surrounding the board are several icons: a yellow lightning bolt warning symbol (top left), a red horseshoe magnet with yellow lightning bolts (top right), a red thermometer (middle left), a red laser pointer (middle right), and a yellow radiation warning symbol (bottom center). A yellow lightning bolt also connects the magnet to the circuit board.

Fault Injection (FI) is a class of hardware attacks in which the device is stressed in an unusual way to make it malfunction.

There are many different ways to cause a chip to malfunction; extremes of voltage, electromagnetic fields, clock speeds, temperature, light, ionizing radiation...

These attacks require physical access to the hardware

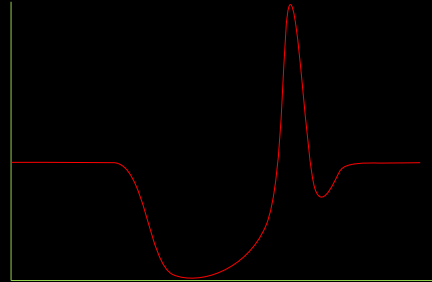
Fault injection is often referred to as “glitching”

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

Voltage Fault Injection (VFI)

Glitches are introduced to the power rails, by briefly changing the supplied voltage, or by driving one or more pins to an incorrect voltage.



With Voltage FI, we cause brief “glitches” to the power of the device.

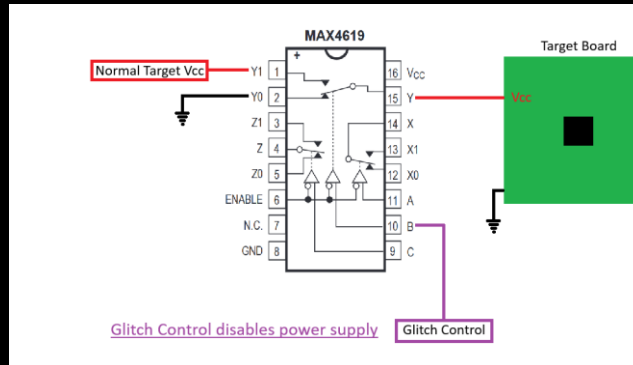
This can be by briefly interrupting the power supply, or applying an incorrect voltage, or shorting a pin.

Sometimes board modifications are needed; for example, removing capacitors can make the device less stable and more vulnerable to glitching.

<https://giphy.com/gifs/season-3-the-simpsons-3x24-xT5LMWOExnRmXt2vFS>

Voltage Fault Injection (VFI)

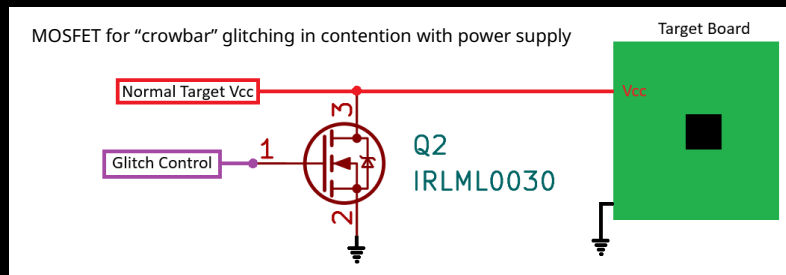
Glitches are introduced to the power rails, by briefly changing the supplied voltage, or by driving one or more pins to an incorrect voltage.



Low entry-level cost – a cheap multiplexer such as the MAX4619 can be used to control the power supplied.

Voltage Fault Injection (VFI)

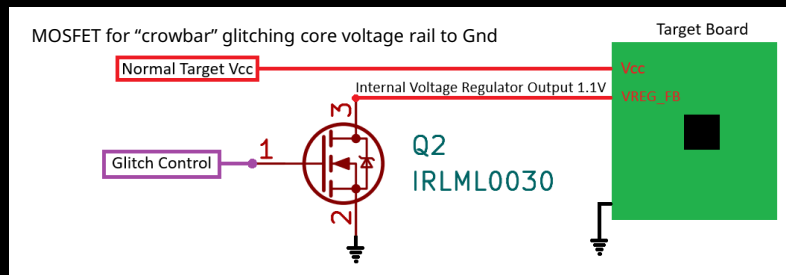
Glitches are introduced to the power rails, by briefly changing the supplied voltage, or by driving one or more pins to an incorrect voltage.



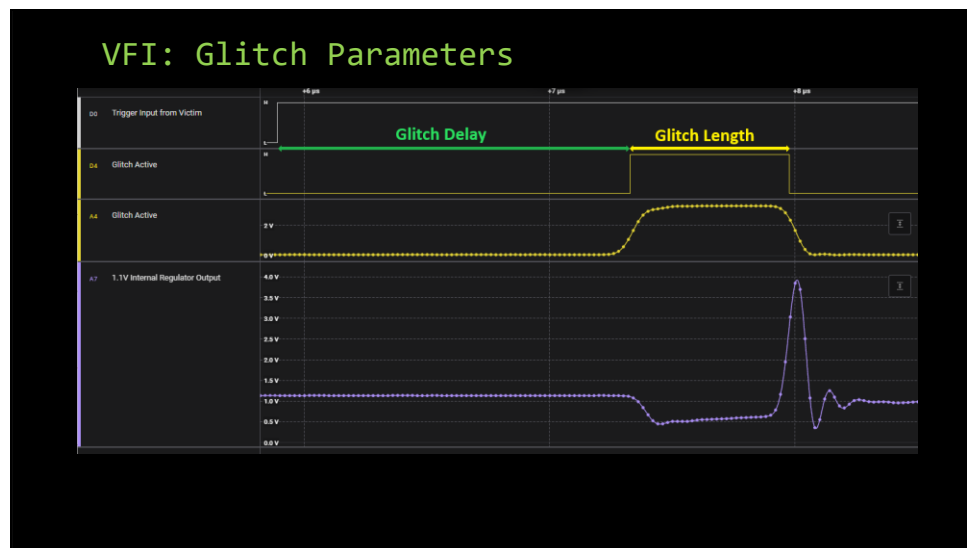
Another method is called "Crowbar Glitching";
Here, the power supply is briefly shorted to ground when the MOSFET transistor is active.

Voltage Fault Injection (VFI)

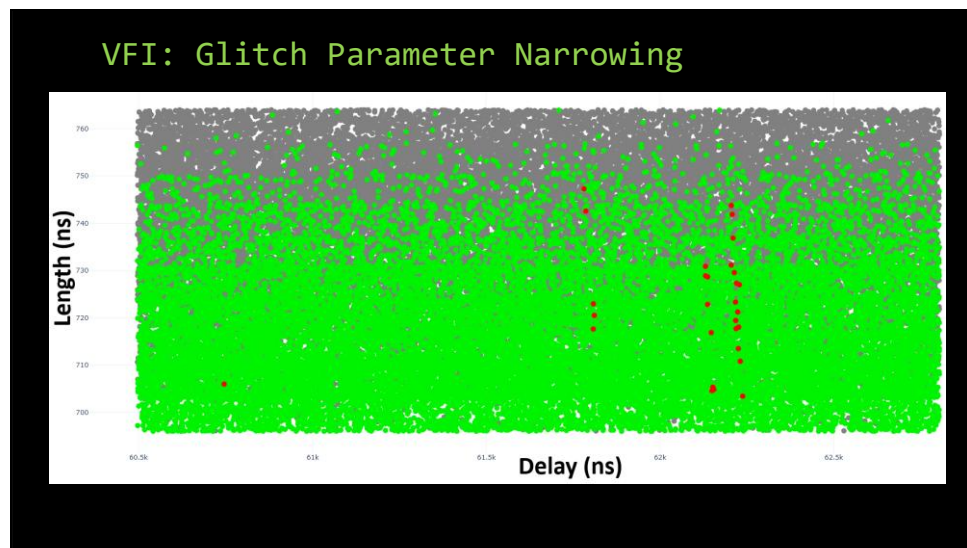
Glitches are introduced to the power rails, by briefly changing the supplied voltage, or by driving one or more pins to an incorrect voltage.



This example shows crowbar glitching the internal regulator output of the chip, rather than the power supply itself, to directly affect the chip's core CPU voltage.



Once wired up, we have two glitch timing parameters to explore;
The glitch delay is the time we should wait after receiving a trigger input, before we activate the glitch
The glitch length is the length of time that the glitch is active for.
By tuning these timings we adjust when and how hard we hit the chip with a glitch.



It can be tricky to find the “Goldilocks Zone”. This is a plot of results, showing how we search for the right combination of glitch delay and glitch length.

Green represents normal device operation – the chip was unaffected by the glitch

Grey is where the glitch caused the device to stop responding – we hit it too hard

Red is a successful attack



Parameter narrowing is an iterative process which allows us to tune for a higher attack success rate.

These parameters are affected by many things, including board capacitance, length of power wires, thickness and length of glitch wire connection...

@TODO Update image and show just reds

Electro-Magnetic Fault Injection (EMFI)

An EM pulse is delivered from a coil close to the chip



[NewAE ChipSHOUTER \(CW520\)](#)
~£3.5K

Riscure EMFI Transient Probe
much more expensive

Homemade circuitry/coil
<£100 but high voltages involved, so not advised!

Generates wide range of glitch effects in target device; a real life "magic wand":
Corruption of reads/write values, program flow alteration, influencing of compare operations...
occasional release of magic smoke and chip self-destruct!

Electro-Magnetic Fault Injection, or EMFI, causes device malfunction by blasting it with a short EM pulse from a coil which is placed near the device.

EMFI is my favourite attack as it requires no board modification, and generates a wide range of effects inside the device.

Effects such as

Corruption of reads/writes,

program flow alteration,

influencing of compare operations...

occasional release of magic smoke and chip self-destruct!

In addition to the delay and length parameters, EMFI also introduces options for the coil voltage (normally 150V – 500V), and positional dependency of the coil relative to the device

I'm using a cheap (~£250) converted 3D printer (Creality Ender 3) as an XYZ stage for positioning the EMFI probe,
Which is the NewAE ChipSHOUTER (~£3500)

NewAE also sell a kit to make a much cheaper, less powerful version, the PicoEMP, for around £50

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- ➔ • Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

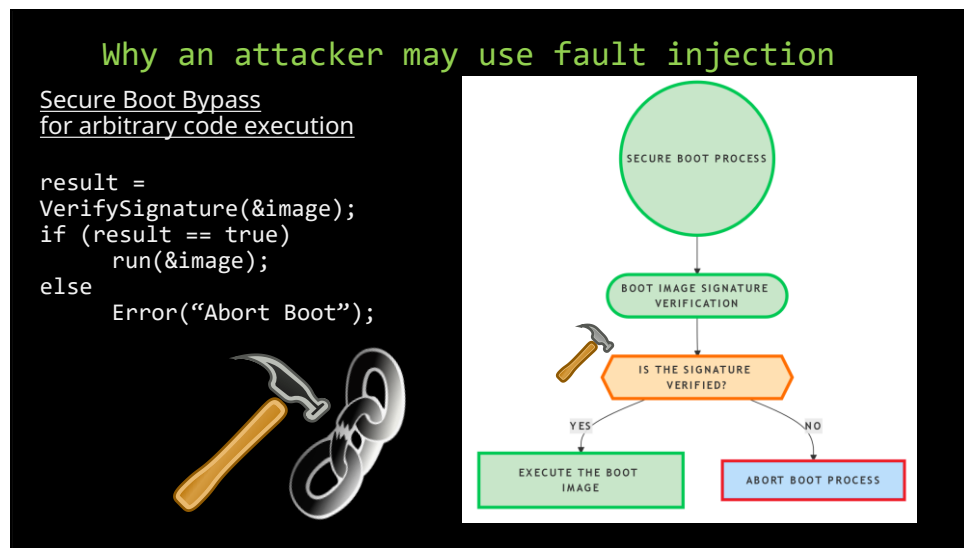
Why The F (I) ?

Why use Fault Injection anyway?

Let's consider what assets there are to protect in an embedded microcontroller

- *Intellectual Property*
 - *Device firmware*
 - *AI models*
 - *Trade secrets – undocumented access protocols*
- *Cryptographic keys*
 - *Signing keys*
 - *Root keys for hardware root of trust – particularly if symmetric*
 - *Key installation keys*
 - *Backend API keys*
- *Access to hardware/restricted functionality*
 - *Avoidance of software limits on features*
 - *DRM avoidance*
 - *Arbitrary code execution – “modded” firmware*

Additionally, obtaining the firmware allows for binary analysis to identify code vulnerabilities, which could lead to a scalable remote attack



Secure Boot Bypass:

It's great to have a chain of trust from a hardware root key and to use strong cryptography for signature checks, but if you can corrupt the result comparison operation with a fault, then this breaks the chain of trust.

From a manufacturer's point of view, often the attacker is the owner of the product, looking to bypass restrictions imposed by the manufacturer. Sometimes this is to unlock restricted features, sometimes this is to bypass DRM protections.


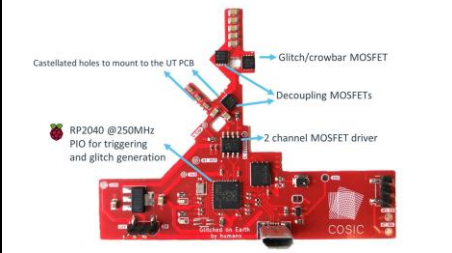
Gaining execution can allow an attacker leverage the full device capabilities; this may include use of cryptographic keys for signature generation, so in metering for example, could subvert non-repudiation of electricity / gas readings.

This means that there are often financial motivations for the attacker.

Why an attacker may use fault injection

Secure Boot Bypass

Lennert Wouters: Glitched on Earth by Humans: A Black-Box Security Evaluation of the SpaceX Starlink User Terminal
<https://github.com/KULEuven-COSIC/Starlink-FI>
<https://www.youtube.com/watch?v=NXqLMmGwJm0>



Lennert Wouters bypassed Secure Boot on the SpaceX Starlink User Terminal using voltage fault injection.

Here we see the modchip PCB which automates this attack, to achieve arbitrary code execution.

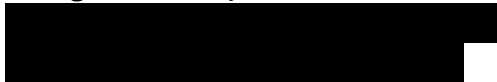
Custom quad core ARM Cortex-A53 without documentation or open samples

Lennert Wouters: Glitched on Earth by Humans: A Black-Box Security Evaluation of the SpaceX Starlink User Terminal

<https://github.com/KULEuven-COSIC/Starlink-FI>

<https://www.youtube.com/watch?v=NXqLMmGwJm0>

A Bright New Day for Broadband — Starlink (51016637753).jpg



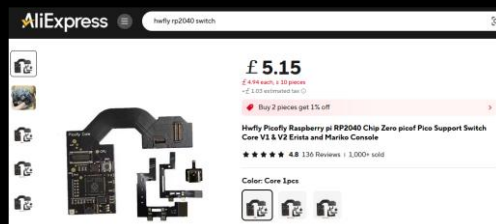
Why an attacker may use fault injection

Secure Boot Bypass

Nintendo Switch modchip for running custom firmware

<https://www.retrosix.wiki/picofly-hwfly-rp2040-nintendo-switch>

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



Secure Boot Bypass:

Games consoles are often targets for FI; there are also Nintendo Switch modchips commercially available, which use Voltage FI to run custom firmware:

<https://www.retrosix.wiki/picofly-hwfly-rp2040-nintendo-switch>

<https://www.aliexpress.com/item/1005007386986743.html>

https://upload.wikimedia.org/wikipedia/commons/7/70/Nintendo_Switch_OLED-Modell_%28BeatEmUps%29_20211001_08.png

<https://commons.wikimedia.org/wiki/User:PantheraLeo1359531>

This file is licensed under the Creative Commons Attribution 3.0 Unported license:

<https://creativecommons.org/licenses/by/3.0/deed.en>

Why an attacker may use fault injection

Read Protection Bypass
Yifan Lu was successful in dumping (reading) the ROM of the Playstation Vita using the NewAE ChipWhisperer to perform voltage fault injection.



Injecting Software Vulnerabilities with Voltage Glitching, Yifan Lu
<https://arxiv.org/abs/1903.08102>

Debug / Read Protection Bypass:

In some systems, debug capabilities of the chip are disabled, with the intention of preventing anyone being able to connect with a debugger and read/write memory. Sony put a lot of effort into trying to protect the PS Vita, after previous failure to protect the PSP / PS3. The PS Vita firmware was dumped through the use of voltage FI. Often FI attacks are foot-in-the-door local physical attack, in order to extract the firmware. This can then be analysed for flaws, which can allow an attacker to pivot to a logical or network attack.

Injecting Software Vulnerabilities with Voltage Glitching, Yifan Lu
<https://arxiv.org/abs/1903.08102>

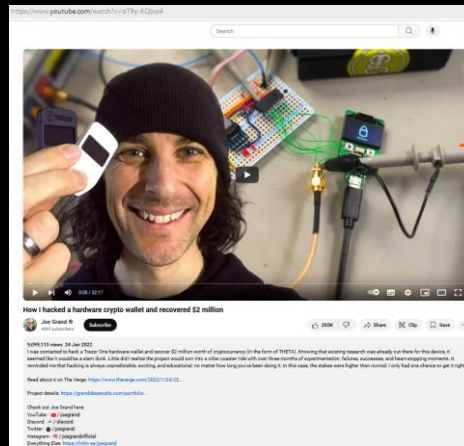
Image: Public Domain:

<https://upload.wikimedia.org/wikipedia/commons/thumb/b/b4/PlayStation-Vita-1101-FL.jpg/1024px-PlayStation-Vita-1101-FL.jpg>

Why an attacker may use fault injection

Read Protection Bypass

Joe Grand recovered \$2m of THETA cryptocurrency from a Trezor One hardware wallet, using voltage FI
<https://www.youtube.com/watch?v=dT9y-KQbqi4>



Read Protection Bypass:

There are financial motivations too:

A stored secret in the Trezor One cryptocurrency hardware wallet was recovered by using voltage fault injection to bypass the read protection mechanism.

Joe Grand has a great video showing his attack to recover \$2m of THETA when the owner had forgotten his password.

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

How The F (I) ?

How does Fault Injection cause security violations?

How FI affects a device

Affects program flow, security settings and internal variables. Mainly a transient effect, but can lead to permanent data changes too.

- **Instruction skipping...**
 - Most prevalent effect
- **Data in flight corruption...**
 - Misread of stored value, address/data bus corruption
- **Out of order operation...**
 - Read operation may complete early, before data fetch is complete
- **Op Code Corruption...**
 - Use of incorrect register
 - The "Jungle Jump" – program counter gets set to incorrect address and execution continues from there!

FI generally causes transient faults; brief errors which aren't permanent, but in some cases, permanent changes can occur.

Affects security settings and internal variables

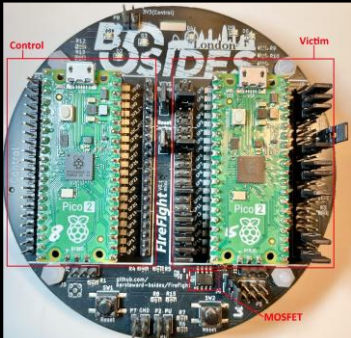
- *Instruction skipping – most prevalent effect*
 - *Early termination of loops*
 - *Skipping of conditional branches*
- *Data in flight corruption – misread of stored value, address/data bus corruption*
 - *Use of incorrect data for comparison operations*
- *Out of order operation...*
 - *Read operation may complete early, before data fetch is complete*

- *Data corruption*
- *Mainly transient effect, but can lead to permanent data changes too*
 - *FI generally causes transient faults; brief errors which aren't permanent, but in some cases, permanent changes can occur.*
- *Op Code Corruption*
 - *A mov data instruction may become a branch instruction*
 - *Use of incorrect register*
 - *The “Jungle Jump” – program counter gets set to incorrect address and execution continues from there! This effect is by far the hardest to mitigate against.*

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- Mitigation techniques & standardisation
- Other attacks

Voltage FI Demo: FIreFIght



This demo will (hopefully!) show a Pi Pico 2 performing voltage glitching on another Pi Pico 2, with the aid of a MOSFET (crowbar glitching the 1.1V internal regulator output)

On the left side we have the “Control” board, and on the right, the “Victim” board.

All PCB gerber/production files and source code for this are available on github: <https://github.com/barsteward-bsides/FireFight>
(Thanks to AsFaBw for the PCB design work)

On to my demo, which I’ve called FIreFIght.

This is a Pi Pico 2 vs another Pi Pico 2:

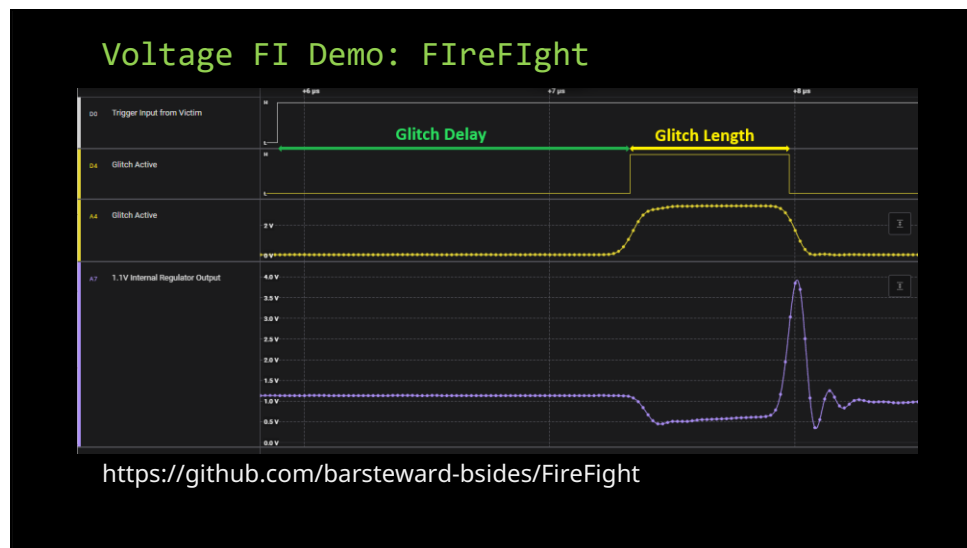
On the left side we have the “Control” board, and on the right, the “Victim” board.

6 weeks ago this board and this particular demo weren’t planned, and then I heard my talk had been accepted, so a massive thank you to AsFaBw who did all the hard work on the PCB design/layout, whilst I was busy trying to get the code to work!

Each time the Victim board runs, it performs an AES encryption and transmits the ciphertext over UART.

The Control board controls the Victim’s power and RUN line, and using the incredible PIO feature of the Pi Pico2 for timings accurate to 6.66ns, performs the voltage glitch by controlling the MOSFET which briefly shorts the CPU core voltage line to Gnd.

Doing so will occasionally disturb the AES encryption calculation, and **sometimes** causes a very specific type of error.



As a reminder, we have *glitch Delay* and *glitch Length* parameters

Voltage FI Demo: FIREFight DFA

Just causing an instruction skip, or a misread didn't seem to have enough jeopardy for a live conference demo, so instead let's try something a bit more tricky...

Differential Fault Analysis (DFA) to try to recover the AES key by analysing the faulty ciphertext outputs

Round 8	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
1B	5A 19 A3 7A	BE D4 0A DA	BE D4 0A DA	00 B1 54 FA	EA B5 31 7F	EAD273218580BAD2312BF5687F8D292F
1B	41 49 E0 8C	83 38 E1 64	38 E1 64 83	51 C8 78 18	D2 8D 28 8D	
1B	42 DC 19 04	2C 86 D4 72	04 F2 2C 86	2F 89 6D 98	73 8A F5 29	
1B	B1 1F 65 8C	C8 C8 4D FE	FE C8 C8 4D	D1 FF CD EA	21 D2 68 2F	
Round 9	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
1B	EA 04 65 85	87 F2 4D 97	87 F2 4D 97	47 4D A3 AC	AC 19 28 57	AC7766F319ADC2128012941575C086E
1B	83 45 5D 96	CC 6E 4C 98	6E 4C 98 CC	37 D4 7D 9F	77 FA D1 5C	
1B	5C 33 98 B0	4A C3 46 E7	46 E7 43 C3	84 E4 3A 42	66 DC 29 88	
1B	F0 2D AD C5	BC D8 95 A8	A8 BC D8 95	ED A5 A8 BC	F3 21 41 6E	
Round 10	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
1B	E8 59 88 18	C9 C8 3D A6	C9 C8 3D A6		06 C9 E1 B6	D014F9A8C9EE2589E13F0CC8B6638CA6
1B	40 2E A1 C3	89 31 32 2E	31 32 2E 89		14 EE 3F 63	
1B	F2 38 13 42	89 87 7D 2C	7D 2C 89 87		F9 25 8C 8C	
1B	1E 84 E7 D2	72 5F 94 B5	B5 72 5F 94		A8 89 CB A6	
Output Ciphertext						
	39 B2 DC 19					
	25 DC 11 6A					
	84 89 85 88					
	1D F8 97 32					3925841D02DC09F8DC18597196A8B32

Instead of a simple “can we cause any error” demo, I’m aiming for something a bit more tricky:

AES Key Recovery using Differential Fault Analysis. This is where we use FI to cause errors in a cryptographic operation, and analyse the output to derive information about the key. The “Victim” performs an AES 128 encryption and transmits the ciphertext over UART, and the “Control” board performs the voltage glitch by controlling the MOSFET which briefly shorts the CPU core voltage line to Gnd, and collects and analyses the output.

Occasionally we will disturb the AES encryption calculation, and *sometimes*, if the timings are just right, causes a very specific type of error:

We aim to *disturb the value of a single byte at the end of Round 8*

<reference only – will not speak>

Control board uses the incredible PIO feature of the Pi Pico2 for timings accurate to 6.66ns,

A single byte error in either byte 0, 5, 10, or 15 will cause an error in the ciphertext at bytes 0, 7, 10, and 13

A single byte error in either byte 1, 6, 11, or 12 will cause an error in the ciphertext at bytes 3, 6, 9, and 12

A single byte error in either byte 2, 7, 8, or 13 will cause an error in the ciphertext at bytes 2, 5, 8, and 15

A single byte error in either byte 3, 4, 9, or 14 will cause an error in the ciphertext at bytes 1, 4, 11, and 14

Voltage FI Demo: FIReFIght DFA

Just causing an instruction skip, or a misread didn't seem to have enough jeopardy for a live conference demo, so instead let's try something a bit more tricky...

Differential Fault Analysis (DFA) to try to recover the AES key by analysing the faulty ciphertext outputs

Round 8	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
10	5A 19 A3 7A	8E D4 0A DA	8E D4 0A DA	00 B1 54 FA	EA 85 31 7F	EAD2732185B0BAD2312BF5607FBD292F
11	41 49 E0 BC	83 3B E1 64	3B E1 64 83	50 C8 76 18	D2 8D 28 8D	
12	42 DC 19 BA	2C 86 D4 F2	D4 F2 2C 86	2F 89 6D 99	73 BA F5 29	
13	B1 1F 65 BC	C8 08 AD FE	FE C8 08 AD	D1 FF CD EA	21 D2 68 2F	
Round 9	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
10	EA 84 65 85	67 F2 4D 97	67 F2 4D 97	47 4D A3 56	AC 19 28 57	AC7766F319ADC212B012941575C006E
11	82 45 5D 96	45 6E 4C 98	6E 4C 98 45	37 D4 7D 24	77 FA D1 3C	
12	5C 33 98 B9	44 C3 46 F2	46 F2 44 C3	94 E4 3A 6D	66 DC 29 8B	
13	F0 2D AD C5	8C D8 95 A6	A6 8C D8 95	ED A5 A5 49	F3 21 41 6E	
Round 10	Start of round	After Subbytes	After Shiftrows	After MixColumns	Round Key Value	
10	C8 59 88 03	C9 C8 3D 7C	C9 C8 3D 7C		DE C9 E1 86	D014F9ABC9EE2589E13F0CC8B6630CAG
11	49 3E A1 26	89 31 32 F9	31 32 F9 89	7A	14 EE 3F 63	
12	F2 38 13 80	89 87 7D 7A	7D 7A 89 87		F9 25 0C 0C	
13	1E 84 E7 2D	72 5F 94 D8	D8 72 5F 94	D8	A8 89 C8 A6	
Output ciphertext						
	39 B3 DC CA					
	25 DC C8 6A					
	8A 5F 85 88					
	70 F8 97 32					
3925847082DC3FFB0CC8B597CA6A8B32						

This will cause a pattern of 4 bytes to be affected in the ciphertext output. There are 4 different groups of these 4 byte patterns that we are interested in - we need at least 2 faulty ciphertexts for each 4 byte group, and the unaffected ciphertext output, to perform key recovery.

<reference only – will not speak>

Control board uses the incredible PIO feature of the Pi Pico2 for timings accurate to 6.66ns,

A single byte error in either byte 0, 5, 10, or 15 will cause an error in the ciphertext at bytes 0, 7, 10, and 13

A single byte error in either byte 1, 6, 11, or 12 will cause an error in the ciphertext at bytes 3, 6, 9, and 12

A single byte error in either byte 2, 7, 8, or 13 will cause an error in the ciphertext at bytes 2, 5, 8, and 15

A single byte error in either byte 3, 4, 9, or 14 will cause an error in the ciphertext at bytes 1, 4, 11, and 14



This is not a new attack and it heavily relies upon open source libraries such as @Doegox's PhoenixAES

I made minor modifications made to PhoenixAES and aeskeyschedule libraries to get them to run in uPython on the Pico 2

Now for the demo... let's see if we can achieve exactly this effect of corrupting a single byte at the end of round 8 of the encryption...

<https://github.com/SideChannelMarvels/JeanGrey/tree/master/phoenixAES>

What we'll cover

- Who am I?
- What is fault injection?
- Types of fault injection attacks
- Why/where are fault injection attacks used?
- How can fault injection compromise security goals?
- Voltage FI Demo / How you can try this yourself
- ➔ • Mitigation techniques & standardisation
- Other attacks

Mitigation Techniques

To protect the security goals of a system, numerous mitigation techniques can be employed:

- Software hardening techniques
 - Fail-safe default initialised values
 - Avoid trivial values for constants such as 0 and 1; maximise hamming distance
 - Repeated checks for comparisons
 - Checks that loops completed the correct number of iterations
 - Randomised timing delays, to make repeatability and attack parameter narrowing harder
 - Control flow integrity checks
 - ... See Riscure's "Fault Mitigation Patterns" whitepaper for more details: <https://www.riscure.com/publication/fault-mitigation-patterns/>

It's really tricky to write code that will fail safe during a hardware attack

- *Fail-safe default initialised values*
- *Avoid trivial values for constants such as 0 and 1; maximise hamming distance*
- *Repeated checks for comparisons*
- *Checks that loops completed the correct number of iterations*
- *Randomised timing delays, to make repeatability and attack parameter narrowing harder*
- *Control flow integrity checks*

Software mitigation patterns is a deep topic. Riscure have a fantastic whitepaper on this, but it does not go into too much depth regarding the cost of implementing these mitigations: <https://www.riscure.com/publication/fault-mitigation-patterns/>

Cost is a trade-off between code size, performance, and reliability (all of which are important in embedded systems), plus code complexity/maintainability, and development time.

Mitigation Techniques

To protect the security goals of a system, numerous mitigation techniques can be employed:

- Hardware techniques
 - Glitch resistant internal power circuitry
 - Glitch detectors
 - Voltage monitoring circuitry
 - Oscillator disturbance detection
 - Honeypot logic paths
 - Memory Protection Units to prevent code execution in restricted areas
 - Shielding
 - Control flow integrity mechanisms

Hardware mitigations can also be employed

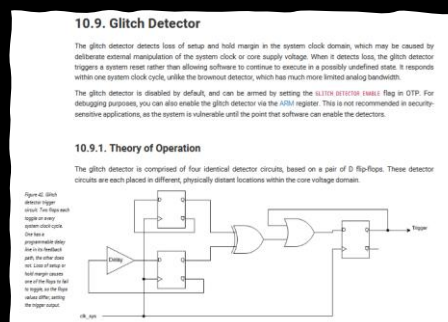
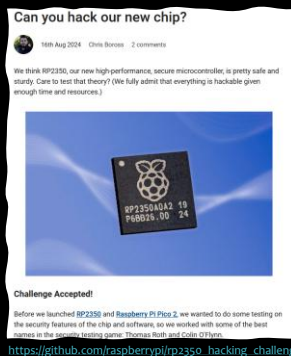
- *Glitch resistant internal power circuitry*
- *Glitch detectors*
 - *Voltage monitoring circuitry*
 - *Oscillator disturbance detection*
 - *Honeypot logic paths*
- *Memory Protection Units to prevent code execution in restricted areas*
- *Shielding*
- *Control flow integrity mechanisms*

Cost is size of additional silicon real estate needed, increase in manufacturing cost, and risk of false positives causing reliability issues

Mitigation Techniques: RP2350 (Raspberry Pi Pico 2)

The RP2350 chip used for the Pi Pico 2 includes a configurable glitch detector, and there's a ~~\$10,000~~ \$20,000 bug bounty for bypassing the chip security features and recovering a secret stored in the OTP flash memory

<https://www.raspberrypi.com/news/30000-badges-and-still-no-hack/>



Earlier this year the Raspberry Pi RP2350 chip was released (as used on the Pi Pico 2). This chip includes a variety of fault injection mitigations, and there is a bug bounty for bypassing these chip security features and recovering a secret stored in the OTP flash. In September they doubled the bounty to \$20,000.

https://github.com/raspberrypi/rp2350_hacking_challenge

<https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf>

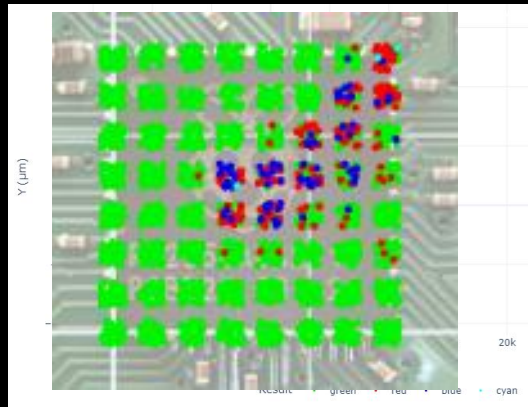
<https://www.raspberrypi.com/news/30000-badges-and-still-no-hack/>

Mitigation Techniques: RP2350 (Raspberry Pi Pico 2)

So how effective is the glitch detector?

This is an EMFI scan I did before enabling the glitch detector, on a simple nested for loop counter...

Red = Successful glitch
Green = Expected response
Blue = Device reboot
Cyan = Corrupted response



I took a look at this device to determine how effective the glitch detector is:
This shows the results of an EMFI campaign, against a nested for loop counter running on the device. Basically, I was trying to cause it to mis-count.
This was without the glitch detector enabled, as a baseline

Red = Successful glitch

Green = Expected response

Blue = Device reboot

Cyan = Corrupted response

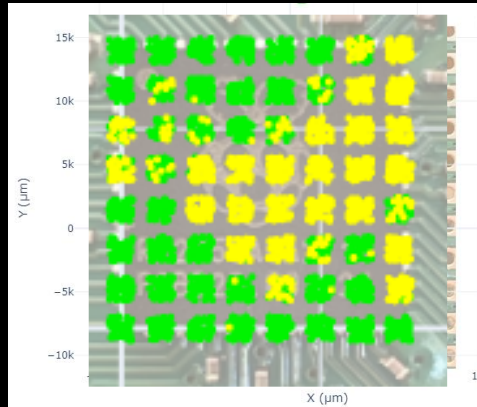
Mitigation Techniques: RP2350 (Raspberry Pi Pico 2)

So how effective is the glitch detector?

And with the glitch detector enabled...

Yellow = Glitch Detected
Green = Expected response
Red = Successful glitch

(1mm Coil, lower power,
plus a week of trying)



We see that where there were successful glitches, these attempts are now caught by the glitch detector.

But that's not the end of the story; after a further week of refinement, and using a smaller coil at a lower power,

I was able to corrupt the loop counter without triggering the glitch detector, albeit at a much lower success rate.

Mitigation Techniques: RP2350 (Raspberry Pi Pico 2)

However... on Dec 27th at 38C3, there's a talk claiming to have defeated the challenge

Aedan Cullen

Most of what I do is related to embedded systems, robotics, or efficient computing. Other fields, like security research, are just a byproduct of always learning how things work :)



Session

12-27

Hacking the RP2350

23:00

Aedan Cullen

60min

Raspberry Pi's RP2350 microcontroller introduced a multitude of new hardware security features over the RP2040, and included a Hacking Challenge which began at DEF CON to encourage researchers to find bugs. The challenge has been defeated and the chip is indeed vulnerable (in at least one way). This talk will cover the process of discovering this vulnerability, the method of exploiting it, and avenues for deducing more about the relevant low-level hardware behavior.

Security

Saai ZIGZAG

<https://fahrplan.events.ccc.de/congress/2024/fahrplan/talk/39HFD9/>

And what looks to be the final chapter of this story is that there's a talk at 38C3 in a couple of weeks where Aeden Cullen claims to have successfully defeated the challenge
<https://fahrplan.events.ccc.de/congress/2024/fahrplan/talk/39HFD9/>

Standards and certifications



Standards and certification schemes are influencing things:

- [Common Criteria \(ISO/IEC 15408:2022\)](#) is the certification standard for Smart Cards and high security devices, but this is very expensive to comply with.
- NIST [FIPS 140-3](#)
- Automotive standard [ISO/SAE 21434:2021](#) has forced automotive manufacturers to consider these types of attacks
- Certification schemes such as [ARM PSA](#) and [SESIP](#) have a number of levels, some of which require resistance to FI attacks
- The [EU Cyber Resilience Act](#) will enforce strict incident reporting rules, which may also influence product security decisions.

There are a number of standards and certification schemes which are influencing FI resistance:

Common Criteria is the certification standard for Smart Cards and high security devices, but this is very expensive to comply with, well into 6 figures.

For cryptographic modules, NIST have the FIPS 140-3 scheme

For automotive use, ISO/SAE 21434 has forced all levels of the supply chain from OEMs to Tier 2 suppliers to consider FI attacks in their threat modelling

Other certification schemes available (mainly used for using security as a distinguisher from the competition):

ARM PSA and SESIP, both have a number of security levels, with FI testing included in the higher levels.

The EU Cyber Resilience Act will also mandate product vulnerability reporting, so this may influence security decisions made during development.

CC ISO/IEC 15408:2022: <https://www.iso.org/standard/72891.html>

FIPS 140-3: <https://csrc.nist.gov/pubs/fips/140-3/final>


PSA Certified: <https://www.psacertified.org/>

SESIP: <https://www.trustcb.com/iot/sesip/>

ISO/SAE 21434:2021: <https://www.iso.org/standard/70918.html>

CRA: <https://www.european-cyber-resilience-act.com/>

What we'll cover

- Who am I?
 - What is fault injection?
 - Types of fault injection attacks
 - Why/where are fault injection attacks used?
 - How can fault injection compromise security goals?
 - Voltage FI Demo / How you can try this yourself
 - Mitigation techniques & standardisation
- 
- Other attacks

Invasive attacks(if time!)

- Decapsulation and optical read of ROM
- Micro-probing to connect to internal signals or to connect/disconnect internal lines
- Body Bias Injection: voltage glitch to the ground plane inside the chip – this raises the Gnd voltage and can cause localised data misreads due to the reduced potential difference between Gnd and core voltage.

There are other methods, often more invasive, to recover device secrets,

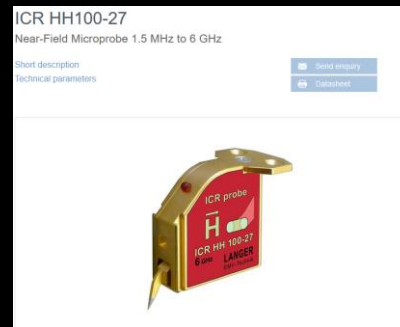
Decapsulation and optical read of ROM

Micro-probing to connect to internal signals or to connect/disconnect internal lines

Body Bias Injection: voltage glitch to the ground plane inside the chip – this raises the Gnd voltage and can cause localised data misreads due to the reduced potential difference between Gnd and core voltage.

Side Channel Analysis (if time!)

- Detection of tiny data dependant fluctuations in timing, power or electromagnetic emissions.
- It can be possible to fully recover a cryptographic key that's in use, by capturing and analysing the EM emissions from the chip, by placing a near-field microprobe close to the chip surface.



<https://www.langer-emv.de/en/product/near-field-microprobe-sets-icr-hh-h-field/26/icr-hh100-27-set-near-field-microprobe-1-5-mhz-6-ghz/768/icr-hh100-27-near-field-microprobe-1-5-mhz-to-6-ghz/101>

And side-channel analysis can use *tiny data dependant fluctuations in timing, power or electromagnetic emissions* to recover cryptographic keys.

SCA can also be useful for fault injection, when used to trigger the glitch, or in timing analysis during parameter refinement.

Conclusions

- It's hard (and costly) to protect against physical attacks on hardware if people can get access to the chip.
- These attacks are becoming more widely known/exploited and the tools are getting cheaper.
- Glitch detectors (and other mitigations) can make a huge difference to the difficulty and repeatability of a fault injection attack, but they're not perfect.
- There is more effort going into hardware and software protection mechanisms now too.
- System design that avoids storage of secrets is a great defence, but not always practical

- *It's hard (and costly) to protect against physical attacks on hardware if people can get access to the chip.*
- *These attacks are becoming more widely known/exploited and the tools are getting cheaper.*
- *Glitch detectors (and other mitigations) can make a huge difference to the difficulty and repeatability of a fault injection attack, but they're not perfect.*
- *There is more effort going into hardware and software protection mechanisms now too.*
- *System design that avoids storage of secrets is a great defence, but not always practical*

Code Credit

FireFIght control interface, including PIO glitch control: @barsteward

<https://github.com/barsteward-bsides/FireFight>

DFA Key recovery library phoenixAES: Philippe Teuwen @doegox

<https://github.com/SideChannelMarvels/JeanGrey/tree/master/phoenixAES>

AES key schedule library aeskeyschedule: Marcel Nageler @fanoster

<https://github.com/fanosta/aeskeyschedule>

Other Credits

PCB Design: AsFaBw <https://github.com/AsFaBw>

ChipSHOUTER EMFI probe: Colin O'Flynn @oflynn.com (NewAE)

<https://www.newae.com/> (Check out the ChipWhisperer too)

Incredible patience: My wife

[Fortunately, no link!](#)

Image Credits

- Morph image used under licence: <https://creativecommons.org/licenses/by/4.0/>
Science Museum Group. Model of 'Morph'. 1999-5162 Science Museum Group Collection Online.
<https://collection.sciencemuseumgroup.org.uk/objects/co8180635/model-of-morph>
- AI image of chip torture <https://deepai.org/>
- Homer Simpson light switch <https://giphy.com/gifs/season-3-the-simpsons-3x24-xT5LMW0ExnRmXt2vFS>
- MAX4619 image under fair usage from <https://www.analog.com/media/en/technical-documentation/data-sheets/MAX4617-MAX4619.pdf>
- Starlink Dish image
[https://commons.wikimedia.org/wiki/File:A_Bright_New_Day_for_Broadband_%E2%80%94_Starlink_\(51016637753\).jpg](https://commons.wikimedia.org/wiki/File:A_Bright_New_Day_for_Broadband_%E2%80%94_Starlink_(51016637753).jpg) Steve Jurvetson from Los Altos, USA, [Creative Commons Attribution 2.0](#)
- Starlink Modchip image used with permission from Lennert Wouters [modchip.jpg](#):
<https://github.com/KULeuven-COSIC/Starlink-FI>
- Joe Grand screenshot from YouTube used under fair usage
<https://www.youtube.com/watch?v=dT9y-KQbqi4>
- Nintendo Switch image by PantheraLeo1359531:
https://upload.wikimedia.org/wikipedia/commons/7/70/Nintendo_Switch_OLED-Modell_%28BeatEmUps%29_20211001_08.png
Licensed under the [Creative Commons Attribution 3.0 Unported license](#)
- All other images and clipart unrestricted or included under fair usage

These slides, along with the FireFight PCB design files, and the embedded FireFight code for the DFA demo are available at:

<https://github.com/barsteward-bsides/FireFight>
(Definitely not a Rick-Roll)

This presentation copyright @barsteward 2024
Released under CC BY 4.0

<https://creativecommons.org/licenses/by/4.0/>

- Bluesky: @barsteward.bsky.social

- Mastodon: @barsteward@infosec.exchange

- ~~RIP~~ Twitter: @barsteward

