Hardware-Assisted Program Execution Integrity: HAPEI

Ronan Lashermes, Hélène Le Bouder and Gaël Thomas

INRIA, IMT-Atlantique, DGA

November 29th, 2018

NordSec 2018, Oslo

Content

Section 1

Introduction

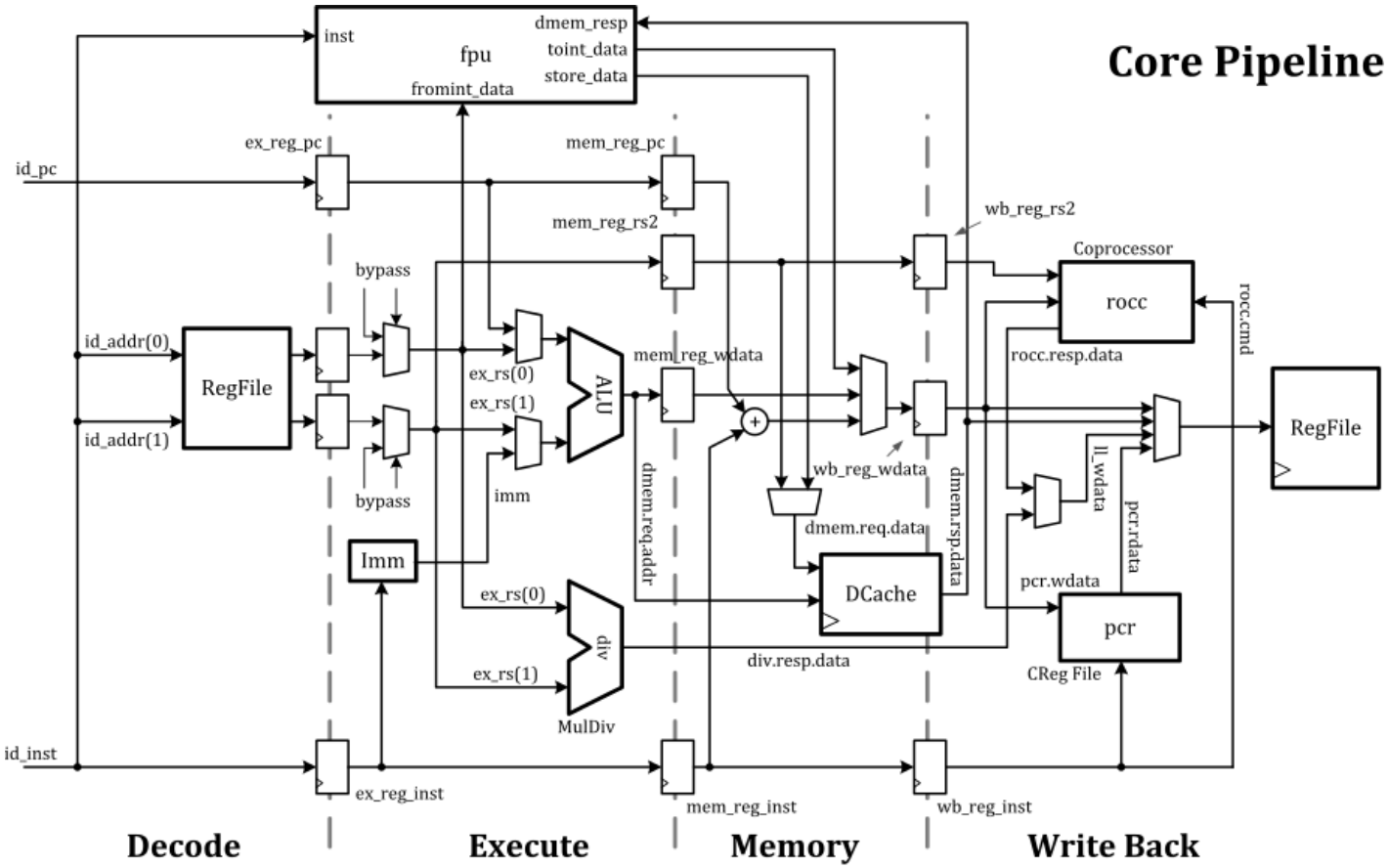
Instructions

```
08000598 <blink_wait>:
8000598:   b580        push    {r7, lr}
800059a:   b082        sub    sp, #8
800059c:   af00        add    r7, sp, #0
800059e:   4b0b        ldr    r3, [pc, #44]    ; (80005cc <blink_wait+0x34>)
80005a0:   603b        str    r3, [r7, #0]
80005a2:   2300        movs    r3, #0
80005a4:   607b        str    r3, [r7, #4]
80005a6:   e005        b.n    80005b4 <blink_wait+0x1c>
80005a8:   687b        ldr    r3, [r7, #4]
80005aa:   f503 03f4     add.w    r3, r3, #7995392    ; 0x7a0000
80005ae:   f503 5390     add.w    r3, r3, #4608    ; 0x1200
80005b2:   607b        str    r3, [r7, #4]
80005b4:   687a        ldr    r2, [r7, #4]
80005b6:   683b        ldr    r3, [r7, #0]
80005b8:   429a        cmp    r2, r3
80005ba:   d3f5        bcc.n    80005a8 <blink_wait+0x10>
80005bc:   f7ff ffe2     bl    8000584 <fast_trig_up>
80005c0:   2003        movs    r0, #3
80005c2:   f000 f8af     bl    8000724 <wait>
80005c6:   3708        adds    r7, #8
80005c8:   46bd        mov    sp, r7
80005ca:   bd80        pop    {r7, pc}
80005cc:   e00be00c     .word    0xe00be00c
```

The software abstraction is a lie

Ok, it works for simple architectures

What is hardware ?



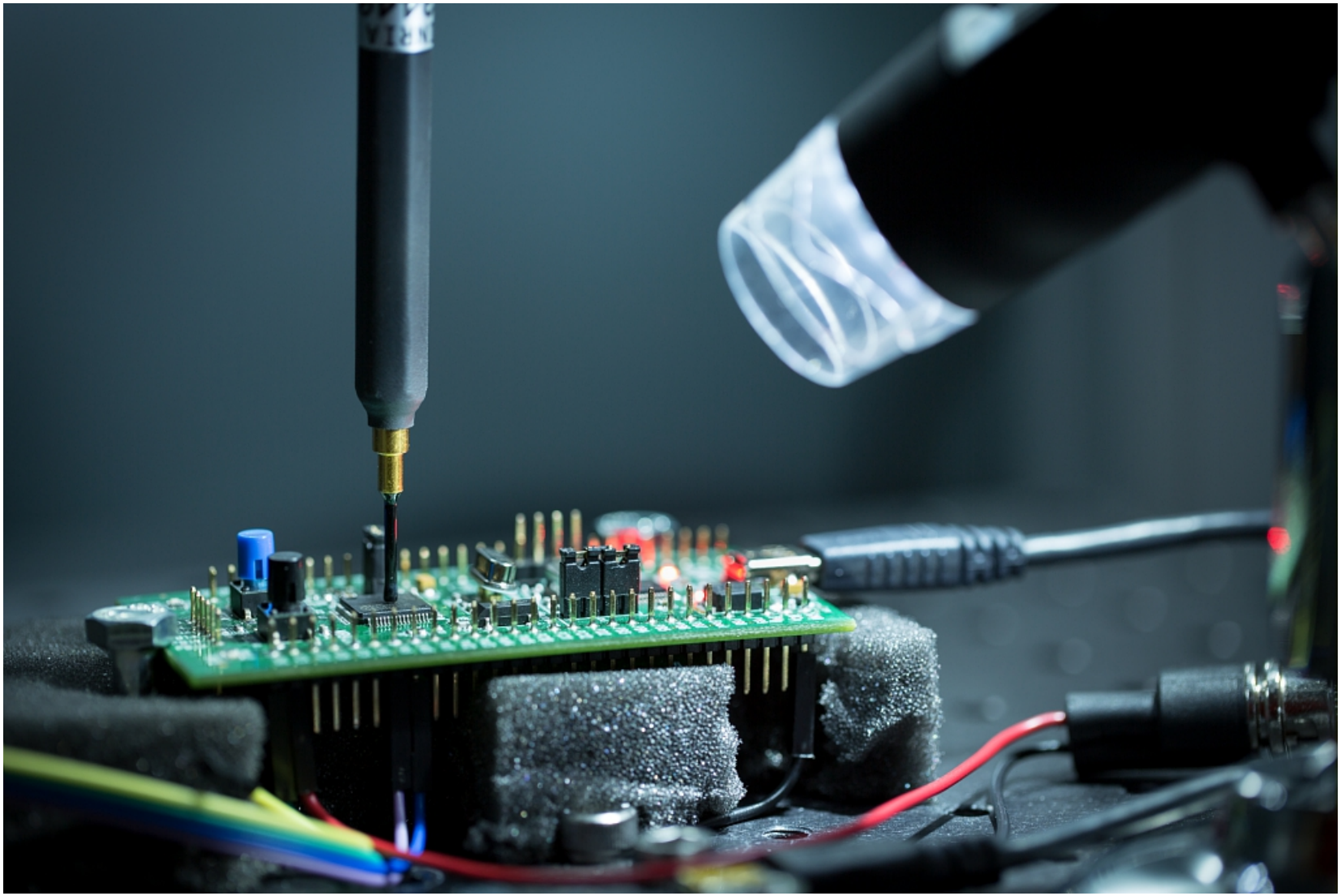
Rocket core (RISC-V)

Fallacies about hardware

1. Memory access is $O(1)$ in time $\Rightarrow$ Cache timing attacks.
2. Instructions are executed in order $\Rightarrow$ Spectre/Meltdown/...
3. Program integrity is guaranteed

Hardware fault attacks

©Inria / Photo C. Morel



©Inria / Photo C. Morel

Fault activated backdoor

Source code

```
void blink_wait()
{
  unsigned int wait_for = 3758874636;
  unsigned int counter;
  for(counter = 0; counter < wait_for; counter += 8000000);
}
```

Assembly

```
08000598 <blink_wait>:
push    {r7, lr}
sub     sp, #8
add     r7, sp, #0
ldr     r3, [pc, #44]     ; (80005cc <blink_wait+0x34>)
...
adds    r7, #8
mov     sp, r7
pop     {r7, pc}
.word    0xe00be00c ; @80005cc, 0xe00be00c = 3758874636
```

Fault activated backdoor

Source code

```
void blink_wait()
{
  unsigned int wait_for = 3758874636;
  unsigned int counter;
  for(counter = 0; counter < wait_for; counter += 8000000);
}
```

Assembly

```
08000598 <blink_wait>:
push    {r7, lr}
sub     sp, #8
add     r7, sp, #0
ldr     r3, [pc, #44]     ; (80005cc <blink_wait+0x34>)
...
adds    r7, #8
mov     sp, r7
nop
b backdoor
```

The previous application could have been proven correct.

Section 2

Ensuring program execution integrity

Several integrities

- **Instructions Integrity (II)**: executed instructions belong to the program.
- **Control Flow Integrity (CFI)**: only authorized control flow (jumps, branches, ...).
- **Data Integrity (DI)**: program data cannot be tampered with.
- Program Integrity = II + CFI

Several attack models

- **Code Injection Attacks (CIA)**: an attacker tries to divert the control flow to execute its own malicious payload.
- **Code-Reuse Attacks (CRA)**: an attacker tries to execute a malicious payload composed by a sequence of legitimate pieces of programs (often called widgets).
- **Hardware Fault Injection (HFI)**: the attacker can edit the program, at runtime, by modifiyng data or instruction values.

SOFIA: Instruction Set Randomization[1]

Control Flow Integrity

Encrypt instructions, with program state encoding:

$$i' = E_k(PC_{prev}||PC) \oplus i$$

Example

1: i1 '
2: i2 '
3: i3 '

To decrypt i_3: $i_3 = E_k(2||3) \oplus i'_3$

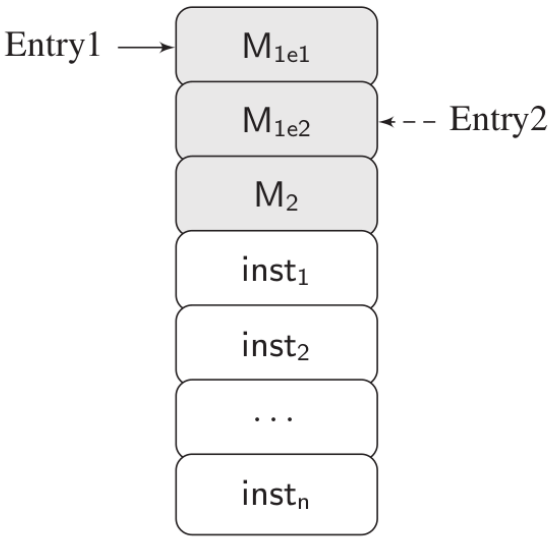
Nice if all instructions have only 1 predecessor... If not we have a special case to deal with.

[1] "SOFIA: Software and control flow integrity architecture", de Clercq et al., 2016

SOFIA: Instruction Set Randomization[2]

System Integrity

Compute a MAC for all 6-instructions blocks, placed at the beginning of the block.



2-predecessors

Compute the two corresponding MACs and adapt the control flow to the entry point.

[2] "SOFIA: Software and control flow integrity architecture", de Clercq et al., 2016

HAPEI: Hardware-Assisted Program Execution Integrity

Goal

Another solution to the CFI&II problem, yet inefficient (for now).
Instruction Set Randomization technique.

1) PC is not the program state

$$acc_n = HMAC_k(acc_{n-1}||i_{n-1}).$$

Initial state can be $acc_0 = HMAC_k(IV)$. k device specific secret key.

2) Encrypt with program state (as in SOFIA)

Encrypt:

$$i'_n = C(acc_n) \oplus i_n.$$

C is a compression fonction: the size of acc_n must be decided according to security requirements.

HAPEI

3) 2-predecessors

The state of the program before a 2-predecessors instruction must be an invariant depending on the 2 possible states. Program state values $\in \mathbb{F}_{2^b}$.
Encrypt:

$$\{\Sigma = a_1 \oplus a_2, i'_n = C(a_1 \cdot a_2) \oplus i_n\}.$$

Decrypt:

$$i_n = C\left(acc_n \cdot (acc_n \oplus \Sigma)\right) \oplus i'_n.$$

Σ gives no information away on a_1 or a_2 if both stay secret.

HAPEI

3) 2-predecessors

The state of the program[illegible]ction must be an invariant depending on t[illegible]tate values $\in \mathbb{F}_{2^b}$.
Encrypt:

$$\{\Sigma = a_1 \oplus a_2, i'_n = C(a_1 \cdot a_2) \oplus i_n\}.$$

Decrypt:

$$i_n = C\left(acc_n \cdot (acc_n \oplus \Sigma)\right) \oplus i'_n.$$

Σ gives no information away on a_1 or a_2 if both stay secret.

HAPEI

4) n-predecessors (cycles allowed in CFG)

The state of the program before a n-predecessors instruction must be a random invariant (rebase). We must be able to project all legitimate program states to this rebased value, and reject illegitimate values.

HAPEI

4) n-predecessors (cycles allowed in CFG)

The state of the program before a n-predecessors instruction must be a random invariant (rebase). We must be able to project all legitimate program states to this rebased value, and reject illegitimate values.

Solution: use projection into subgroups of $\mathbb{F}_{2^b}$. Subgroup of size r exists $\forall r | 2^b - 1$.
Example: $5|2^{16} - 1$, so there is a cyclic subgroup $\{\mu, \mu^2, \mu^3, \mu^4, \mu^5 = 1\}$ for some $\mu \in \mathbb{F}_{2^b}$ with $\mu^r = 1$.

HAPEI

4) n-predecessors (cycles allowed in CFG)

The state of the program before a n-predecessors instruction must be a random invariant (rebase). We must be able to project all legitimate program states to this rebased value, and reject illegitimate values.

Solution: use projection into subgroups of $\mathbb{F}_{2^b}$. Subgroup of size r exists $\forall r | 2^b - 1$.
Example: $5 | 2^{16} - 1$, so there is a cyclic subgroup $\{\mu, \mu^2, \mu^3, \mu^4, \mu^5 = 1\}$ for some $\mu \in \mathbb{F}_{2^b}$ with $\mu^r = 1$.

Encrypt (5-predecessors): $a_1, a_2, \ldots, a_5$. Choose random $c \in \mathbb{F}_{2^b}$.
Compute polynomial P of degree 4 such that:

$$P(a_i) = c \cdot \mu^i.$$

Store $\{P, i'_n = C(c^r) \oplus i_n\}$.

HAPEI

Decrypt

$$i_n = C\left(P(acc_n)^r\right) \oplus i'_n.$$

Works because $\forall i$,

$$P(a_i)^r = \left(c \cdot \mu^i\right)^r = c^r \cdot (\mu^r)^i = c^r.$$

Exponentiation by r required to keep degree of P minimal (but not equal to constant).

HAPEI

Why the exponentiation? Memory efficiency and security

It is possible to devise a polynomial to map directly from all a_i to c (and 1 to 1). But then, the polynomial gives information on its roots.
$\forall a, deg(gcd(P[x] - a, x^{2^b} - x)) \leq 5$, but $deg(gcd(P[x] - c, x^{2^b} - x)) = 5$ for the correct roots.

HAPEI

Why the exponentiation? Memory efficiency and security

It is possible to devise a polynomial to map directly from all a_i to c (and 1 to 1). But then, the polynomial gives information on its roots.
$\forall a, deg(gcd(P[x] - a, x^{2^b} - x)) \leq 5$, but $deg(gcd(P[x] - c, x^{2^b} - x)) = 5$ for the correct roots.

Polynomial Exponentiation Trick

$deg(gcd(P[x] - c \cdot \mu^i, x^{2^b} - x)) \leq 4$ but no information leaks as easily.
But it means that illegitimate program states can be accepted ! It supposes that the attacker cannot control the program state value.

CHIP-8 implementation

CHIP-8

It is an Instruction Set Architecture (ISA) for a video games 8-bit virtual machine (from the 1970s). Extremely simple ISA ($\approx$ 30 instructions). ROMs (video games binaries) are freely available on internet.

Our implementation

Two implementations: the reference and the hardened one. A special key press modifies the next opcode with a random valid one.

Demo time !

Hardening memory usage for a set of CHIP-8 roms.

ROM name	ROM byte size	Instructions count	Polynomials count	Field elements	Polynomials byte size
INVADERS	1283	202	28	99	1584
GUESS	148	49	8	25	400
KALEID	120	59	10	32	512
CONNECT4	194	67	5	19	304
WIPEOFF	206	101	15	47	752
PONG2	264	126	19	60	960
15PUZZLE	384	116	17	54	864
TETRIS	494	189	32	106	1696
BLINKY	2356	856	84	310	4960
VBRIX	507	218	27	93	1488
SYZYGY	946	414	44	149	2384
BRIX	280	134	17	57	912
TICTAC	486	194	23	89	1424
MAZE	34	13	3	10	160
PUZZLE	184	87	10	34	544
BLITZ	391	121	15	47	752
VERS	230	103	24	73	1168
PONG	246	117	18	57	912
UFO	224	106	15	48	768
TANK	560	236	42	139	2224

HAPEI

Limitations

- Indirect branches: '**ADD PC, PC, R1**',
- dynamic libraries,
- system calls,
- load time/run time relocation...

The whole architecture (hardware, software, ...) has to be designed with security in mind.

Section 3

Conclusion

Where the difficulty lies

To prove a solution secure in an abstract model is not enough.

It must be implemented without flaws at all abstractions levels.

This is not a hardware problem...

... it is a cross abstraction problem.

- Information leaks over networks: you can perform cache timing attacks on a distant server.
- RowHammer faults on a co-hosted virtual machine.
- Protocol prover : “cryptographic primitives are supposed unbreakable”.

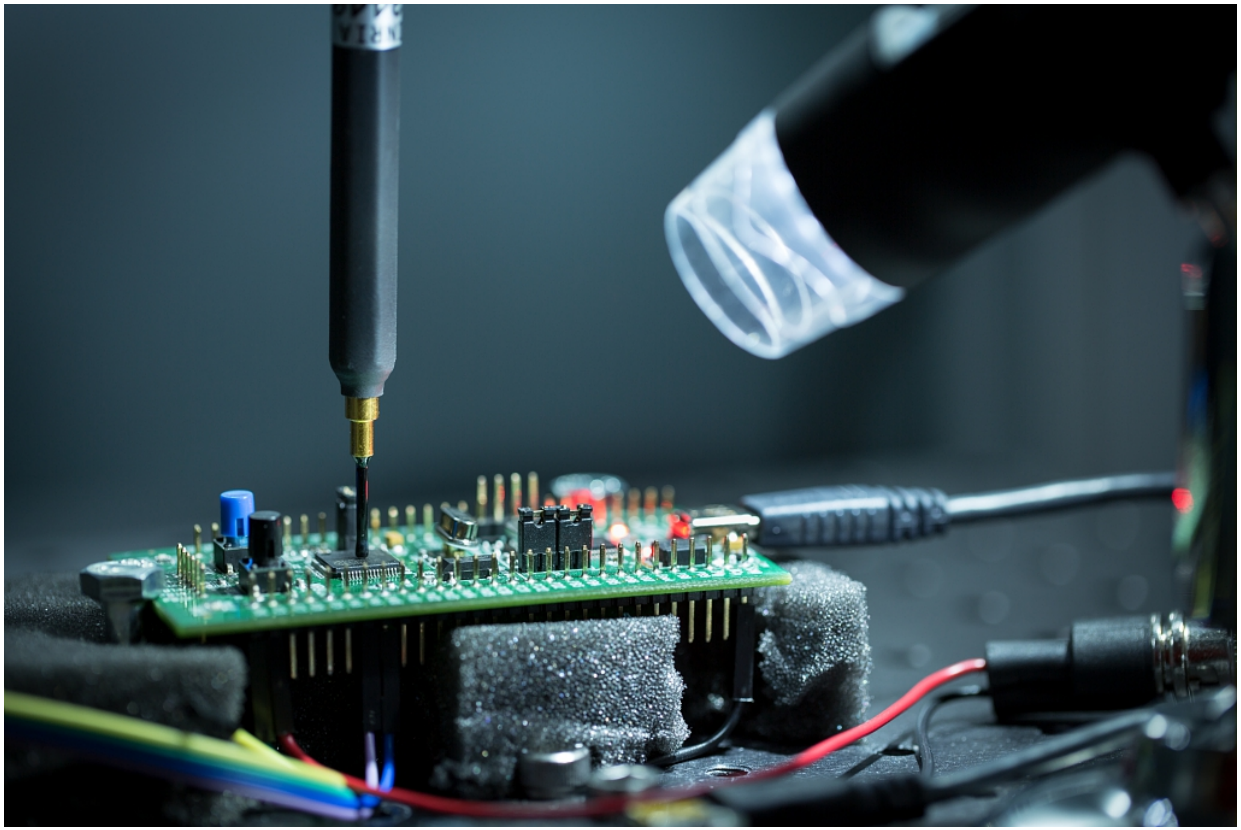
Technologies like *Trustzone* or *SGX* do not help in that regard.

Conclusion

Cross abstraction vulnerabilities reflect our human inability to grasp complexity.
This abstraction layers representation is an illustration of the sociological dimension of systems design.
We need new tools to ensure security across layers.

Thank you!

Any questions?



© Inria / Photo C. Morel