SCI-FI: Control Signal, Code, and Control-Flow Integrity against Fault Injection Attacks

Thomas Chamelot[1], Damien Couroussé[1], Karine Heydemann[2]

[1] *Université Grenoble Alpes, CEA, List* F-38000 Grenoble, France, firstname.lastname@cea.fr
[2] *Sorbonne Université, CNRS, LIP6* F-75005, Paris, France, firstname.lastname@lip6.fr

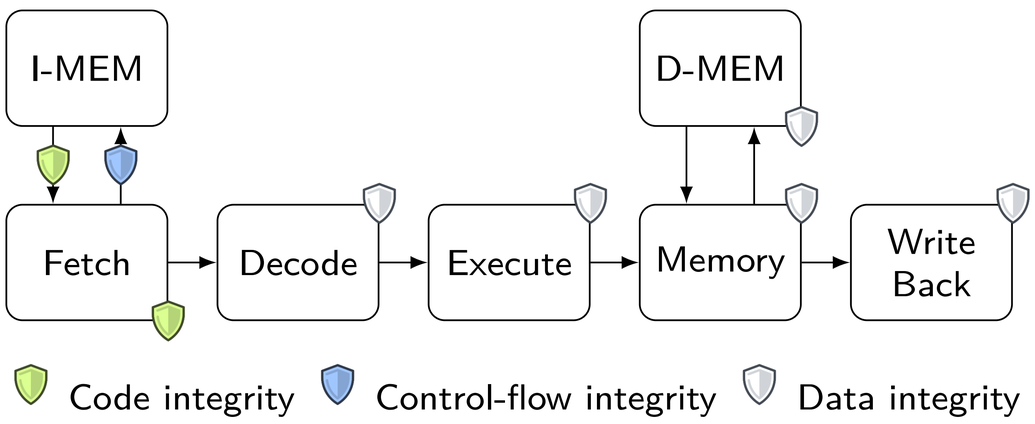
This work was partially funded by the French National Research Agency (ANR) under grant agreement ANR-18-CE39-0003

Context

Fault Injection Attacks

An attacker performs a fault injection attack by using power or clock glitch, EM pulse or laser beam to perturb an integrated circuit.

Required Security Properties



Problem

It has been shown that some vulnerabilities exist at the microarchitecural level [1].

Contributions

Protecting the Pipeline Control Path

New security property: Execution integrity

- SCI-FI combines code and control-flow integrity properties with execution integrity
- SCI-FI achieves execution integrity by protecting the pipeline's control signals

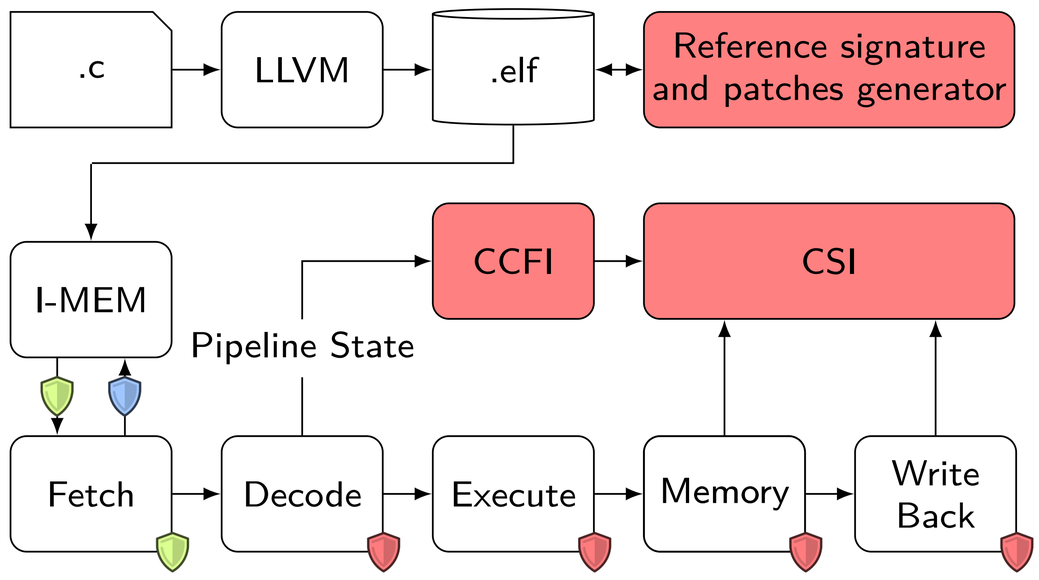
Principles

1. Data-independent control signals outputted by Decode are gathered into a so-called pipeline state Σ
2. The CCFI module enforces code and control-flow integrity and execution integrity for Decode and Execute stages
   (a) Computes signature from current pipeline state and previous signature
   $$S_i = f(\Sigma_i, S_{i-1})$$
   (b) Updates signature to generate collision for instructions with multiple predecessors after a taken branch
   $$S^{'} = u(S, patch)$$
   (c) Verifies runtime signatures against reference signatures located after dedicated control-flow instructions
3. The CSI module enforces execution integrity
   (a) Duplicate signals from the pipeline state
   (b) Checks duplicated signals between pipeline stages
4. A dedicated tool generates reference signatures and patches at compile time

SCI-FI dedicated instructions

- Verification instructions load a reference signature immediately following in the program memory, and trigger the signature verification: `scifi.beq`, `scifi.bne`, `scifi.blt`, `scifi.bltu`, `scifi.bge`, `scifi.bgeu`, `scifi.jal`, `scifi.jalr`
- Load patch instructions fetch a patch value into the CCFI module: `scifi.ldp`

Architecture



Code transformation example with `memcpy` from libgcc.

```c
#include <stddef.h>

__attribute__((scifi_secured))
void *
memcpy(void* dest,
       const void* src,
       size_t len)
{
  char *d = dest;
  const char *s = src;
  while (len--)
    *d++ = *s++;
  return dest;
}
```

```
memcpy:
  scifi.ldp    0(patch_base)
  scifi.beqz   a2, .LBB0_3
  scifi.signature
  mv           a3, a0

.LBB0_2:
  scifi.ldp    4(patch_base)
  lb           a4, 0(a1)
  addi         a2, a2, -1
  addi         a1, a1, 1
  addi         a5, a3, 1
  sb           a4, 0(a3)
  mv           a3, a5
  scifi.bnez   a2, .LBB0_2
  scifi.signature

.LBB0_3
  scifi.ret
  scifi.signature
```

Experimental Evaluation

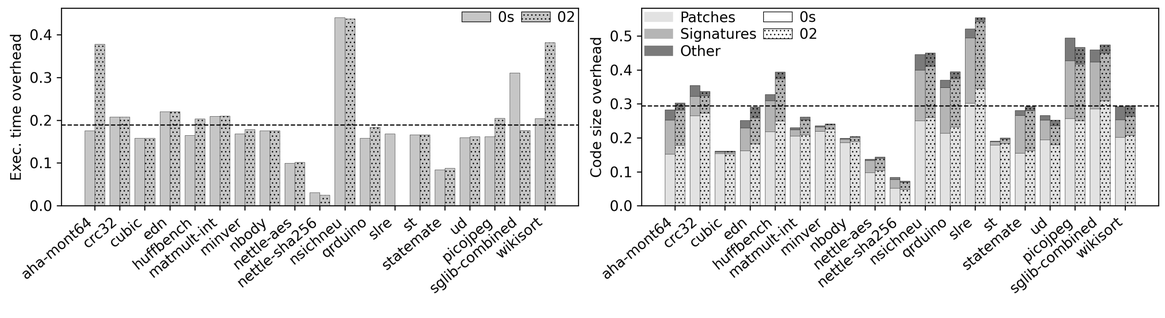
RISC-V RV32I CV32E40P

ASIC implementation 28-FDSOI @ 400MHz

- CRC32: +6.5%
- CBC-MAC Prince: +23.8%

Software evaluation using LLVM 12 with Newlib on Embench-IOT

- Code size: +29.4%
- Execution time: +18.4%



Conclusion

Security Properties

Code, control-flow and execution integrity and additionally code authenticity with CBC-MAC as the signature function

Overheads

Similar to existing state-of-the-art counter-measures for code and control-flow integrity

Future Work

Support for more complex architectures and more complex software (OOP, OS, ...)

You can learn more about SCI-FI in [2]!

Bibliography

[1] Laurent, J. et al. Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures. *Design, Automation & Test in Europe Conference & Exhibition* (2019).
[2] Chamelot T. et al. SCI-FI: Control Signal, Code, and Control Flow Integrity against Fault Injection Attacks. *Design, Automation & Test in Europe Conference & Exhibition* (2022).