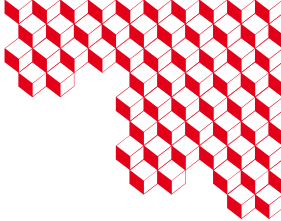


Enhancing the RISC-V Trace Encoder to verify the control-flow and code execution integrity

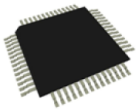
Anthony ZGHEIB, Olivier POTIN, Jean-Baptiste RIGAUD, Jean-Max DUTERTRE and Pierre-Alain MOELLIC

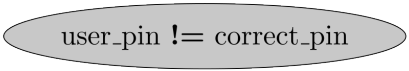
RISC-V Summit Europe, Barcelona, Spain

June 6, 2023

Context - Fault Injection Attacks (FIA)

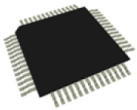
- FIA are effective threats affecting the behavior of a program.

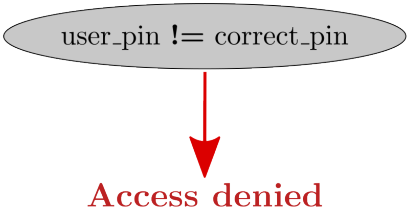




Context - Fault Injection Attacks (FIA)

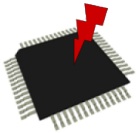
- FIA are effective threats affecting the behavior of a program.

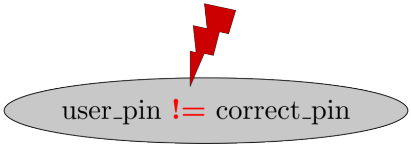




Context - Fault Injection Attacks (FIA)

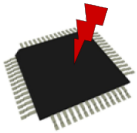
- FIA are effective threats affecting the behavior of a program.

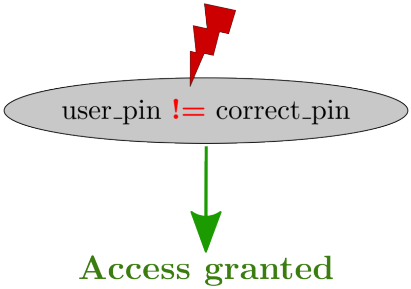




Context - Fault Injection Attacks (FIA)

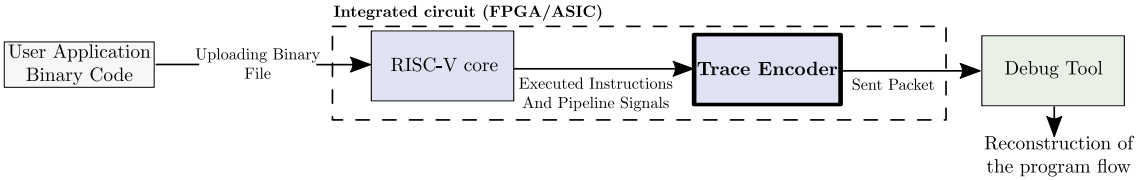
- FIA are effective threats affecting the behavior of a program.





Context - Trace Encoder (TE)

- It compresses, at runtime, the sequence of discontinuities executed by the RISC-V core, **representing the program behavior**, into trace packets.
- Used by developers for debug purposes.
- Embedded debug module designed by RISC-V foundation [6].



Context - TE - Example

- A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, *e.g:* a return instruction.

PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)

Context - TE - Example

- A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, *e.g:* a return instruction.

Context - TE - Example

- A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, *e.g:* a return instruction.

PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	fa0796e3	bnez a5,32c
0x384	00000793	li a5,0
0x388	00078513	mv a0,a5
0x38c	02010113	addi sp,sp,32
0x390	**00008067**	**ret**
0x3ac	00050793	mv a5,a0

Sent Packet

Reported_Address=**0x374**

Sent Packet

Branches=**1**, Branch_Map=**1**, Reported_Address=**0x3ac**

Context - TE - Example

- A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, *e.g:* a return instruction.

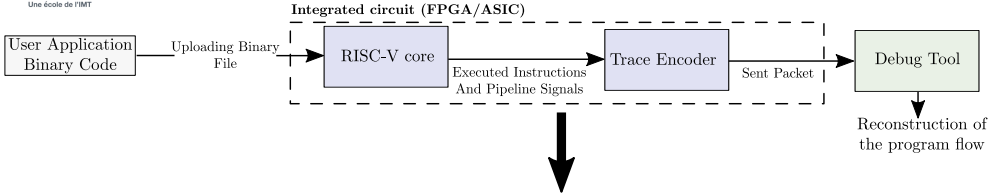
PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)
0x378	fff78713	addi a4,a5,-1
0x37c	00e12223	sw a4,4(sp)
0x380	fa0796e3	bnez a5,32c
0x384	00000793	li a5,0
0x388	00078513	mv a0,a5
0x38c	02010113	addi sp,sp,32
0x390	**00008067**	**ret**
0x3ac	00050793	mv a5,a0

Sent Packet
Reported_Address=**0x374**

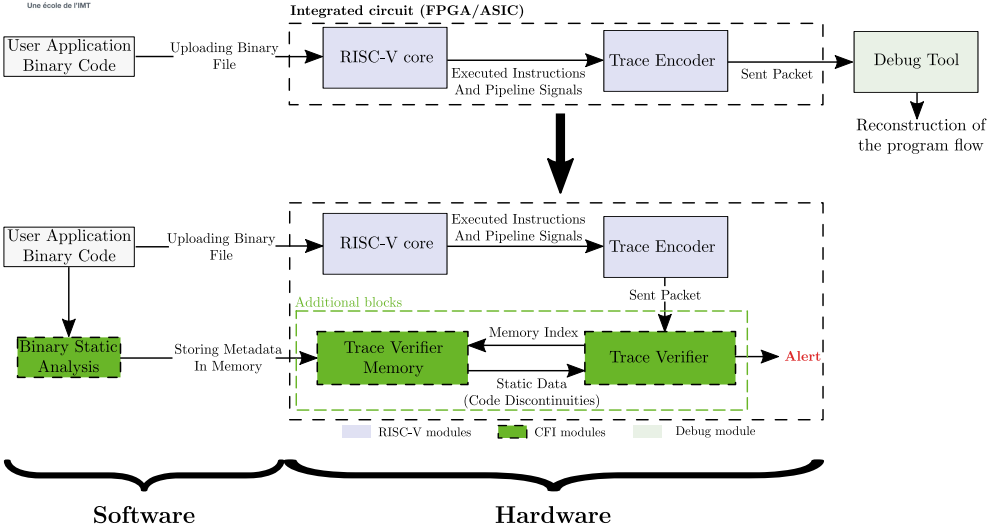
Sent Packet
Branches=**1**, Branch_Map=**1**, Reported_Address=**0x3ac**

- Is it possible to use the TE to **check the program behavior** and detect **FIA**?

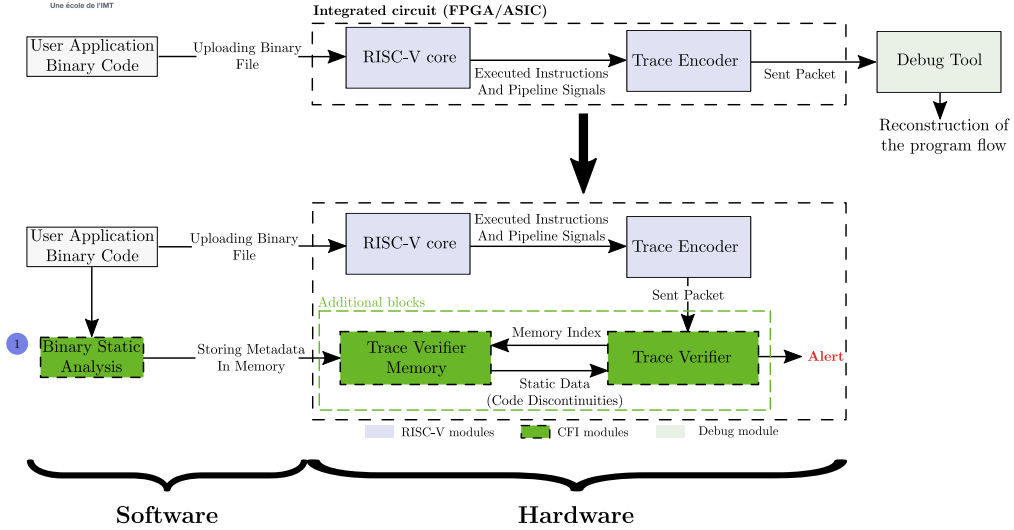
TE-based verification solution



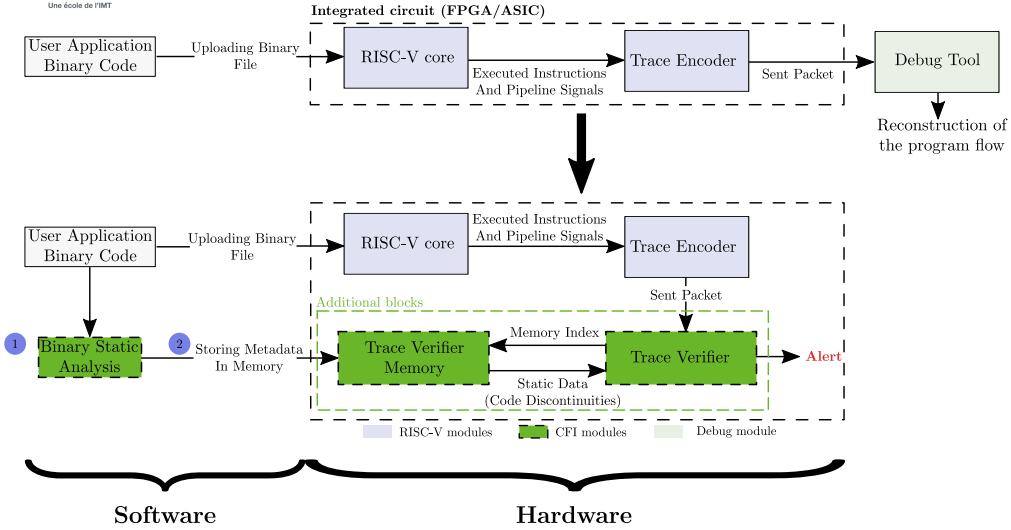
TE-based verification solution



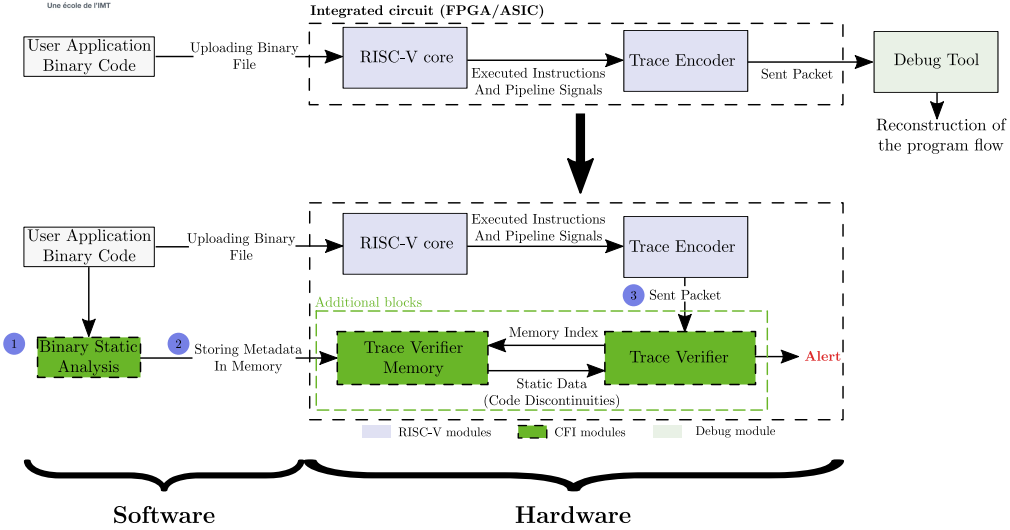
TE-based verification solution



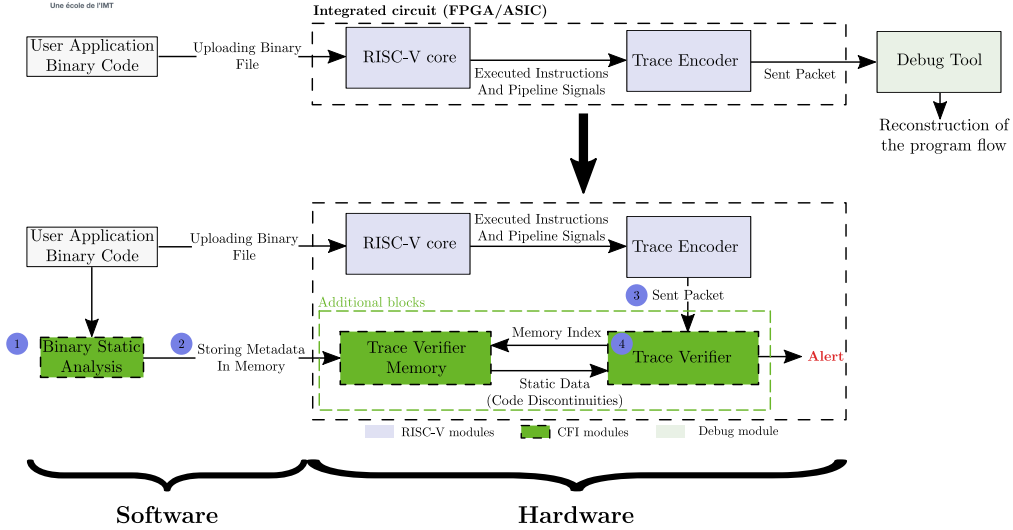
TE-based verification solution



TE-based verification solution

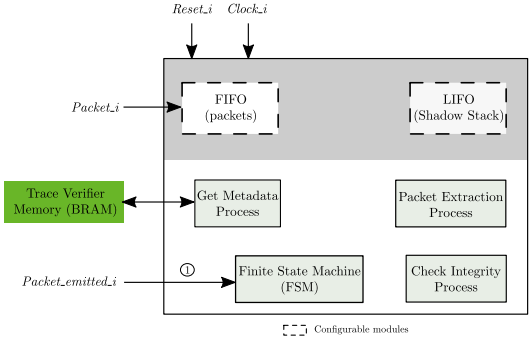


TE-based verification solution



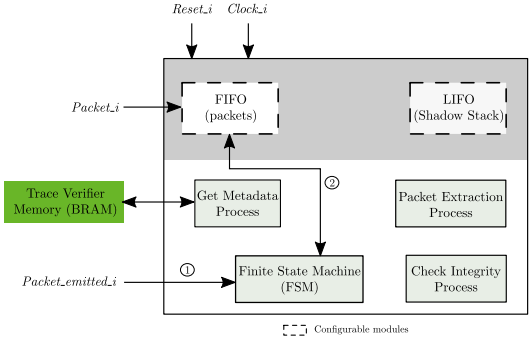
TE-based Solution features

- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.



TE-based Solution features

- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.

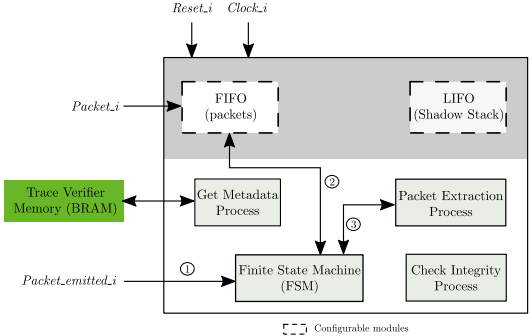


TE-based Solution features

- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.

PACKET CONTENT

Branches =1
Branch_map=1
Address=0x3ac



TE-based Solution features

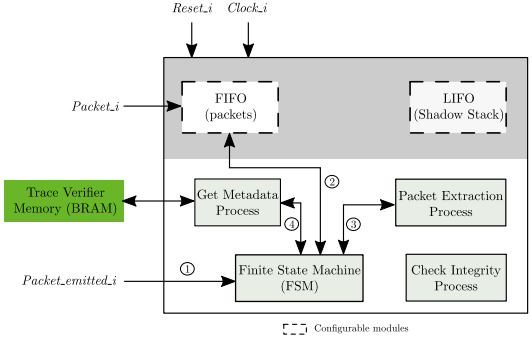
- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.

PACKET CONTENT

Branches =1
Branch_map=1
Address=0x3ac

BRAM

Index :	Content		
3E	: 000000380 (PC)	fa0796e3 (Instruction)	0036 (J Index)
3F	: 00000390 (PC)	00008067 (Instruction)	xxxx (Unused Index)



TE-based Solution features

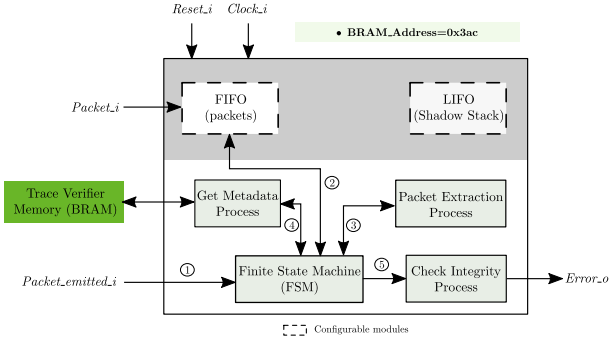
- Verification process starts when a packet is sent.
- Navigation through static data and constitution of the program's followed path.

PACKET CONTENT

Branches =1
Branch_map=1
Address=0x3ac

BRAM

Index :	Content		
3E	: 000000380 (PC)	fa0796e3 (Instruction)	0036 (J Index)
3F	: 00000390 (PC)	00008067 (Instruction)	xxxx (Unused Index)



TE-based Solution features

Covered threats

- Changing the return address of a function – Backward Edge Attacks (BEAs).
- Instruction skip on a function call.

TE-based Solution features

Covered threats

- Changing the return address of a function – Backward Edge Attacks (BEAs).
- Instruction skip on a function call.

Limitation

- Corruption of a **discontinuity instruction** (e.g bne => beq).
- Corruption of **any** non discontinuity instruction.
- Protection of the instructions execution till the first pipeline stage (Fetch stage).

TE-based Solution features

Covered threats

- Changing the return address of a function – Backward Edge Attacks (BEAs).
- Instruction skip on a function call.

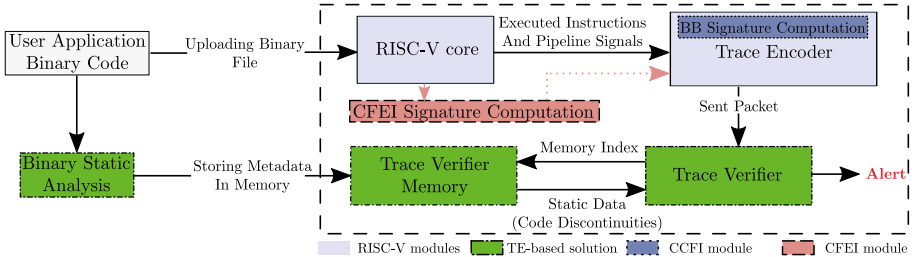
Limitation

- Corruption of a **discontinuity instruction** (e.g bne $=>$ beq).
- Corruption of **any** non discontinuity instruction.
- Protection of the instructions execution till the first pipeline stage (Fetch stage).

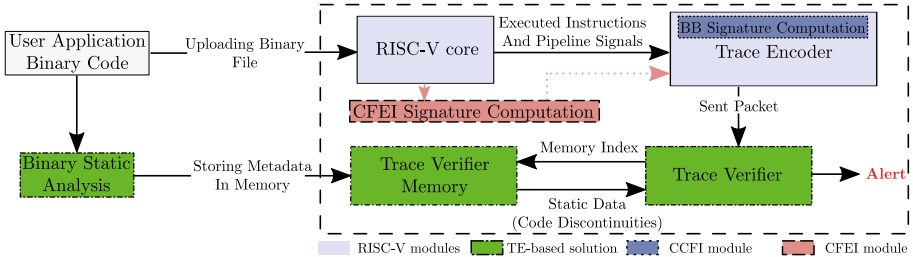
What do we propose?

- Enhancing the **TE standard** to cover more fault models.

Enhancing the TE standard to improve fault coverage



Enhancing the TE standard to improve fault coverage



- A packet is now sent after **each** discontinuity instruction.
- Signature modules are added to the circuit.
- TE module adapted while respecting the retro-compatibility.

Enhancing the TE standard to improve fault coverage

Example - Code and Control-Flow Integrity (CCFI) mode

- A packet is sent after **each** discontinuity with a Basic Block (BB)[1] signature.

PC	Instruction	Assembly Code
0x22c	**00008067**	**ret**
0x374	00412783	lw a5,4(sp)

Sent Packet

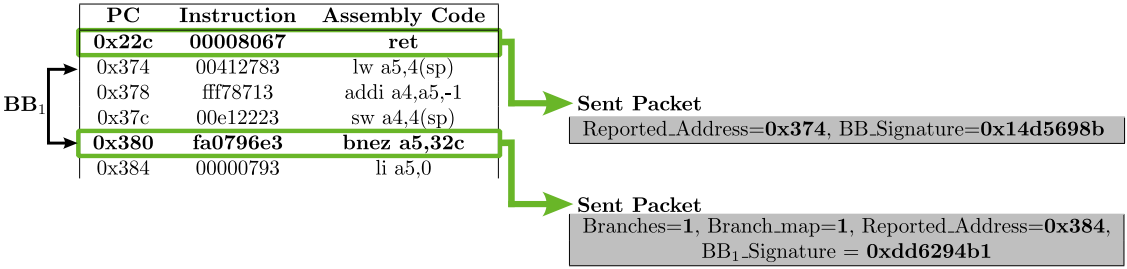
Reported_Address=**0x374**, BB_Signature=**0x14d5698b**

[1] BB is a set of successive instructions where their execution is done consecutively and in order.

Enhancing the TE standard to improve fault coverage

Example - Code and Control-Flow Integrity (CCFI) mode

- A packet is sent after **each** discontinuity with a Basic Block (BB)[1] signature.



[1] BB is a set of successive instructions where their execution is done consecutively and in order.

Enhancing the TE standard to improve fault coverage

Example - Code and Control-Flow Integrity (CCFI) mode

- A packet is sent after **each** discontinuity with a Basic Block (BB)[1] signature.

	PC	Instruction	Assembly Code
	0x22c	**00008067**	**ret**
BB_1	0x374	00412783	lw a5,4(sp)
	0x378	fff78713	addi a4,a5,-1
	0x37c	00e12223	sw a4,4(sp)
	0x380	**fa0796e3**	**bnez a5,32c**
BB_2	0x384	00000793	li a5,0
	0x388	00078513	mv a0,a5
	0x38c	02010113	addi sp,sp,32
	0x390	**00008067**	**ret**
	0x3ac	00050793	mv a5,a0

Sent Packet
Reported_Address=**0x374**, BB_Signature=**0x14d5698b**

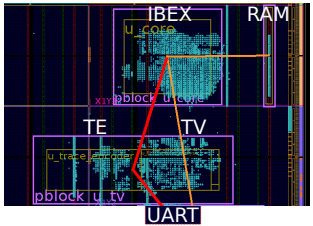
Sent Packet
Branches=**1**, Branch_map=**1**, Reported_Address=**0x384**, BB_1_Signature = **0xdd6294b1**

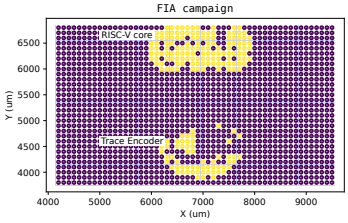
Sent Packet
Reported_Address=**0x3ac**, BB_2_Signature=**0x041caa95**

[1] BB is a set of successive instructions where their execution is done consecutively and in order.

Results

- We simulated FIA and experimental EMFI on RV32IMC instructions and **IBEX RISC-V core** on FPGA.





	SFC	BEA	SBI	CDI	CI	PIE	VL
TV	✓	✓	(✗)	✗	✗	✗	↗ ↗
TV for CFI	✓	✓	✓	✓	✗	✗	↘
TV for CCFI	✓	✓	✓	✓	✓	✗	↘
TV for CFEI	✓	✓	✓	✓	✓	✓	↘

- SFC: Skip on Function Calls
- BEA: Backward Edge Attack
- SBI: Skip on Branch Instructions
- CDI: Corruption of a Discontinuity Instruction
- CI: Corruption of any Instruction
- PIE: Protection of Instruction Execution in the micro-architecture.
- VL: Verification Latency

Conclusion

- The TE compliant solution [11] detects skip attacks on function calls / returns.
- Solutions **with the TE standard enhancement** detect corruption of any instruction (CCFI) [10] and attacks on microarchitectural signals (CFEI) [12].

Conclusion

- The TE compliant solution [11] detects skip attacks on function calls / returns.
- Solutions **with the TE standard enhancement** detect corruption of any instruction (CCFI) [10] and attacks on microarchitectural signals (CFEI) [12].

- **No software overhead** as the TV implementation does not modify the user code nor the RISC-V compiler.

Perspectives

- The TE compliant solution [11] detects skip attacks on function calls / returns.
- Solutions **with the TE standard enhancement** detect corruption of any instruction (CCFI) [10] and attacks on microarchitectural signals (CFEI) [12].

- **No software overhead** as the TV implementation does not modify the user code nor the RISC-V compiler.

- Check how indirect calls could be treated in our approach.
- Enable the branch prediction feature.
- Upgrade the TE-based solutions to handle interruptions and core exceptions.

Thank you for your attention!

This research is part of the Projet COFFI: ANR-18-CES39-003

References I

[1] Thomas Chamelot, Damien Couroussé, and Karine Heydemann. "SCI-FI: control signal, code, and control flow integrity against fault injection attacks". In: *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2022, pp. 556–559.

[2] Jean-Luc Danger et al. "CCFI-Cache: A Transparent and Flexible Hardware Protection for Code and Control-Flow Integrity". In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018 21st Euromicro Conference on Digital System Design (DSD). Aug. 2018, pp. 529–536. DOI: 10.1109/DSD.2018.00093.

References II

[3] Asmit De et al. “Hardware Assisted Buffer Protection Mechanisms for Embedded RISC-V”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (Dec. 2020). Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 4453–4465. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.2984407.

[4] Ruan De Clercq et al. “SOFIA: software and control flow integrity architecture”. In: *Computers & Security* 68 (2017), pp. 16–35.

[5] Leila Delshadtehrani et al. “Nile: A programmable monitoring coprocessor”. In: *IEEE Computer Architecture Letters* 17.1 (2017), pp. 92–95.

[6] RISC-V. *Working Draft of the RISC-V Processor Trace Specification*. URL: https://github.com/riscv/riscv-trace-spec (visited on 11/29/2020).

References III

[7] Olivier Savry, Mustapha El-Majihi, and Thomas Hiscock. “Confidaent: Control FLow protection with Instruction and Data Authenticated Encryption”. In: *2020 23rd Euromicro Conference on Digital System Design (DSD)*. 2020 23rd Euromicro Conference on Digital System Design (DSD). Aug. 2020, pp. 246–253. DOI: 10.1109/DSD51259.2020.00048.

[8] Mario Werner et al. “Sponge-based control-flow protection for IoT devices”. In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 214–226.

[9] Shaza Zeitouni et al. “Atrium: Runtime attestation resilient under memory attacks”. In: *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2017, pp. 384–391.

References IV

[10] Anthony Zgheib et al. “A CCFI Verification Scheme Based on the RISC-V Trace Encoder”. In: *Constructive Side-Channel Analysis and Secure Design: 14th International Workshop, COSADE 2023, Munich, Germany, April 3–4, 2023, Proceedings*. Springer. 2023, pp. 42–61.

[11] Anthony Zgheib et al. “A CFI Verification System based on the RISC-V Instruction Trace Encoder”. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, pp. 456–463. DOI: 10.1109/DSD57027.2022.00067.

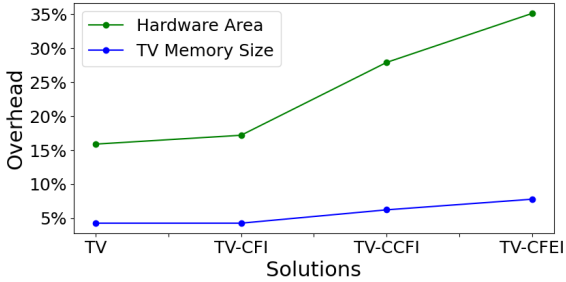
[12] Anthony Zgheib et al. “CIFER: Code Integrity and control Flow verification for programs Executed on a RISC-V core”. In: *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2023, pp. 100–110.

Comparison of our solution with related works

Solution	FIXER [3]	NILE [5]	SOFIA [4]	SCFP [8]	CONFIDAENT [7]	CCFI-Cache [2]	ATRIUM [9]	SCI-FI [1]	TE-CFI [11]	**TE-CCFI [10]**	**TE-CFEI [12]**
No User Code Modification	✗	✗	✗	✗	✗	✗	✓	✗	✓	✓	✓
No Compiler Modification	✓	✗	✓	✗	✗	✗	✓	✗	✓	✓	✓
No Pipeline Modification	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓	✓
No Performance Overhead	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Backward Edge Protection	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Forward Edge Protection	✓	✗	✗	✓	✓	(✗)	✗	✗	✗	✗	✗
Code Integrity	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
Code Execution Integrity	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓
Code Confidentiality	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗

Comparison of our solution with related works

Solution	FIXER [3]	NILE [5]	SOFIA [4]	SCFP [8]	SCI-FI [1]	CONFIDAENT [7]	CCFI-Cache [2]	ATRIUM [9]	TE-based CFI [11]	**TE-CCFI [10]**	**TE-CFEI [12]**
Code Size (%)	N/A	N/A	141	19.8	25.4	<36	<30	0	0	**0**	**0**
Performance (%)	1.5	<3	110	9.1	17.5	<36	32	<22.7	0	**0**	**0**
Hardware Area (%)	2.9	15	28.2	N/A	<23.8	N/A	10	<20	<17.1	**<27,9**	**<35,1**



Our CFI Methodology

3 Steps

- Static analysis of the binary application code (1).
- Generation of metadata related to these instructions (2).
- CFI Verification with external HW module (3).

Our CFI Methodology

3 Steps

- Static analysis of the binary application code (1).
- Generation of metadata related to these instructions (2).
- CFI Verification with external HW module (3).

(1)

```
int isabsequal (int x, int y)
{
   if(x == y)
      return 1;
   else if (x == -y)
      return -1;
   else
      return 0;
   end if;
}
```

(2)

RAM

Index :	Content			
1F :	00000080 (PC)	0010006F (Instruction)	0079 (J Index)	0FFF (Unused Index)
...				
79 :	00000914 (PC)	01BD5863 (Instruction)	007B (Index if Br Taken)	007A (Index if Br Not Taken)
...				

Our CFI Methodology

3 Steps

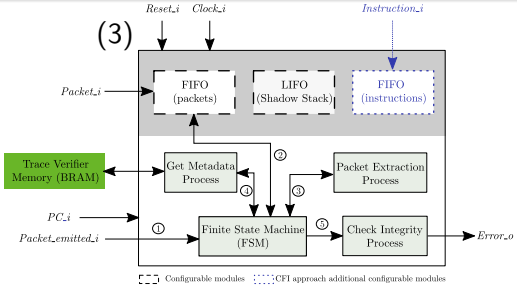
- Static analysis of the binary application code (1).
- Generation of metadata related to these instructions (2).
- CFI Verification with external HW module (3).

(2)

RAM

Index :	Content			
1F :	00000080 (PC)	0010006F (Instruction)	0079 (J Index)	0FFF (Unused Index)
...				
79 :	00000914 (PC)	01BD5863 (Instruction)	007B (Index if Br Taken)	007A (Index if Br Not Taken)
...				

(3)



FIA campaign