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
FIA are effective threats affecting the behavior of a program.

\[ \text{user\_pin} \neq \text{correct\_pin} \]
FIA are effective threats affecting the behavior of a program.
FIA are effective threats affecting the behavior of a program.

user_pin != correct_pin
FIA are effective threats affecting the behavior of a program.

```
user_pin != correct_pin
```

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

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x22c</td>
<td>00008067</td>
<td>ret</td>
</tr>
<tr>
<td>0x374</td>
<td>00412783</td>
<td>lw a5,4(sp)</td>
</tr>
</tbody>
</table>
A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, e.g.: a return instruction.

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x22c</td>
<td>00008067</td>
<td>ret</td>
</tr>
<tr>
<td>0x374</td>
<td>00412783</td>
<td>lw a5,4(sp)</td>
</tr>
</tbody>
</table>

Sent Packet
Reported Address=0x374
A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, e.g: a return instruction.

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

**Sent Packet**
- Reported Address=0x374

**Sent Packet**
- Branches=1, Branch Map=1, Reported Address=0x3ac
A packet is sent when an unpredictable discontinuity (target address is not known from the binary code) is executed, e.g., a return instruction.

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

Is it possible to use the TE to check the program behavior and detect FIA?
TE-based verification solution
TE-based verification solution

Integrated circuit (FPGA/ASIC)

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Debug Tool

Reconstruction of the program flow

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Additional blocks

Binary Static Analysis

Storing Metadata In Memory

Trace Verifier Memory

Memory Index

Static Data (Code Discontinuities)

Trace Verifier

Alert

Software

Hardware

RISC-V Summit Europe 2023, Barcelona, Spain

June 6, 2023
TE-based verification solution

Integrated circuit (FPGA/ASIC)

User Application Binary Code

RISC-V core

Trace Encoder

Debug Tool

Reconstruction of the program flow

1. Binary Static Analysis

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Sent Packet

Trace Encoder

Trace Verifier Memory

Trace Verifier

Alert

Software

Hardware

Additional blocks

Storing Metadata In Memory

Memory Index

Static Data (Code Discontinuities)
TE-based verification solution

Integrated circuit (FPGA/ASIC)

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Debug Tool

Reconstruction of the program flow

User Application Binary Code

Uploading Binary File

1. Binary Static Analysis

2. Storing Metadata In Memory

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Trace Verifier

Memory Index

Static Data (Code Discontinuities)

Trace Verifier

Alert

Additional blocks

Software

Hardware

RISC-V Summit Europe 2023, Barcelona, Spain
TE-based verification solution

Integrated circuit (FPGA/ASIC)

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Debug Tool

Reconstruction of the program flow

User Application Binary Code

Uploading Binary File

1. Binary Static Analysis

2. Storing Metadata In Memory

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Trace Verifier Memory

Memory Index

Static Data (Code Discontinuities)

Trace Verifier

Alert

Software

Hardware

Additional blocks
TE-based verification solution

Integrated circuit (FPGA/ASIC)

User Application Binary Code

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Debug Tool

Reconstruction of the program flow

User Application Binary Code

Uploading Binary File

RISC-V core

Executed Instructions And Pipeline Signals

Trace Encoder

Sent Packet

Additional blocks

Trace Verifier Memory

Memory Index

Static Data (Code Discontinuities)

Trace Verifier

Alert

Software

Hardware
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**

<table>
<thead>
<tr>
<th>Index</th>
<th>Content</th>
<th>PC</th>
<th>Instruction</th>
<th>J Index</th>
<th>Unused Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>3E</td>
<td>000000380</td>
<td></td>
<td>fa0796e3</td>
<td>0036</td>
<td></td>
</tr>
<tr>
<td>3F</td>
<td>00000390</td>
<td></td>
<td>0008067</td>
<td>XXXX</td>
<td></td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Index</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>3E</td>
<td>000000380 PC fa0796e3 Instruction 0036 J Index</td>
</tr>
<tr>
<td>3F</td>
<td>00000390 PC 0008067 Instruction xxxx Unused Index</td>
</tr>
</tbody>
</table>

![Diagram showing TE-based Solution features](image)
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.
**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

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

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x22c</td>
<td>00008067</td>
<td>ret</td>
</tr>
<tr>
<td>0x374</td>
<td>00412783</td>
<td>lw a5,4(sp)</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>PC</th>
<th>Instruction</th>
<th>Assembly Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x22c</td>
<td>00008067</td>
<td>ret</td>
</tr>
<tr>
<td>0x374</td>
<td>00412783</td>
<td>lw a5,4(sp)</td>
</tr>
<tr>
<td>0x378</td>
<td>fff78713</td>
<td>addi a4,a5,-1</td>
</tr>
<tr>
<td>0x37c</td>
<td>00e12223</td>
<td>sw a4,4(sp)</td>
</tr>
<tr>
<td>0x380</td>
<td>fa0796e8</td>
<td>bnez a5,32c</td>
</tr>
<tr>
<td>0x384</td>
<td>00000793</td>
<td>li a5,0</td>
</tr>
</tbody>
</table>

\(^1\) 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.

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

$^1$ BB is a set of successive instructions where their execution is done consecutively and in order.
We simulated FIA and experimental EMFI on RV32IMC instructions and IBEX RISC-V core on FPGA.

<table>
<thead>
<tr>
<th></th>
<th>SFC</th>
<th>BEA</th>
<th>SBI</th>
<th>CDI</th>
<th>CI</th>
<th>PIE</th>
<th>VL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TV</td>
<td>✓</td>
<td>✓</td>
<td>(X)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TV for CFI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>TV for CCFI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>TV for CFEI</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **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

- Solutions with the TE standard enhancement detect corruption of any instruction (CCFI) [10] and attacks on microarchitectural signals (CFEI) [12].

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.

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 II


## Comparison of our solution with related works

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No User Code Modification</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Compiler Modification</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Pipeline Modification</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Performance Overhead</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backward Edge Protection</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward Edge Protection</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>(x)</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Integrity</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Execution Integrity</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Confidentiality</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of our solution with related works

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size (%)</td>
<td>N/A</td>
<td>N/A</td>
<td>141</td>
<td>19.8</td>
<td>25.4</td>
<td>&lt;36</td>
<td>&lt;30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Performance (%)</td>
<td>1.5</td>
<td>&lt;3</td>
<td>110</td>
<td>9.1</td>
<td>17.5</td>
<td>&lt;36</td>
<td>32</td>
<td>&lt;22.7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hardware Area (%)</td>
<td>2.9</td>
<td>15</td>
<td>28.2</td>
<td>N/A</td>
<td>&lt;23.8</td>
<td>N/A</td>
<td>10</td>
<td>&lt;20</td>
<td>&lt;17.1</td>
<td>&lt;27.9</td>
<td>&lt;35.1</td>
</tr>
</tbody>
</table>

![Graph showing overhead and TV memory size](image)

Anthony ZGHEIB
RISC-V Summit Europe 2023, Barcelona, Spain
June 6, 2023 19 / 12
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)  
```c
int isabsequal (int x, int y)  
{
    if(x == y)  
        return 1;
    else if (x == -y)  
        return -1;
    else  
        return 0;
    end if;
}
```

(2)  
RAM

<table>
<thead>
<tr>
<th>Index</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F</td>
<td>00000080</td>
</tr>
<tr>
<td>...</td>
<td>PC</td>
</tr>
<tr>
<td>79</td>
<td>0000914</td>
</tr>
<tr>
<td>...</td>
<td>PC</td>
</tr>
</tbody>
</table>
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).

### RAM

<table>
<thead>
<tr>
<th>Index</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F</td>
<td>00000080 0010006F 0079 0FFF</td>
</tr>
<tr>
<td>...</td>
<td>PC Instruction J Index Unused Index</td>
</tr>
<tr>
<td>79</td>
<td>00000914 01BD5863 007B 007A</td>
</tr>
<tr>
<td>...</td>
<td>PC Instruction Index if Br Taken Index if Br Not Taken</td>
</tr>
</tbody>
</table>
FIA campaign

- Pulse generator
- Oscilloscope
- Motorized XYZ stage
- UART Nexys Board
- Artix-7 FPGA
- GPIO

Anthony ZGHEIB

RISC-V Summit Europe 2023, Barcelona, Spain

June 6, 2023