RT-LIFE

Portable RISC-V Interface for Real-Time Lightweight Security Enforcement
Tight Resource Constraints limit IoT Security
Security Considerations
Scenario

Real-Time embedded IoT device with potentially harmful MMIO periphery.

Embedded Device
RISC-V Processor Core
FE DE EX ME WB

Dangerous Memory-Mapped IO Write Access

Smart Lock
Industrial Robot
Rocket Launcher
Security Arms Race (1/2) – Code Injection

int ORIGINAL_main()
{
    int b = 0;
    if (b) {
        get(R);
    }
    return 0;
}

int ATTACKER_main()
{
    float evil = 1.0;
    if (evil) {
        destroy_world();
    }
    return 0;
}

Existing mitigations: Memory Management Unit, Memory Protection Unit, Data Execution Prevention, Read-Only Memory, Address Space Layout Randomization
Security Arms Race (2/2) – Code Reuse

Without injecting new malicious code, runtime-behavior of existing code is tampered.

Static write protection inadequate.

Runtime-dynamic integrity required.

Control Flow (CF) Instruction

Legal Next Program Counter

Attacker-chosen illegal Program Counter
1. Search code gadgets
2. Exploit memory error(s)
3. Manipulate return address (on stack) to concatenate gadgets

int calledFromMain()
{
    int b=0;
    if(b) {
        get(R);
    }
    return 0;
}
Code Reuse - Jump Oriented Programming (2/3)

- Manipulate *heap* memory to tamper indirect jumps and concatenate gadgets

- FSM Logic constraining the Control Flow Graph
- Pointer integrity
- Artificial intelligence

...
• Tamper data values to slightly manipulate Control Flow (CF) without violating the Control Flow Graph
• Repurpose rarely used memory for virtual registers

Branch

• Fine granular control flow integrity
• Data Flow Integrity
• Data Invariant Integrity

Many different attacks and *countermeasures* in industry and research!
Software / Hardware Partitioning

1. Software-only: Running additional Standard-ISA checking instructions

2. Enhanced Pipeline: Implementing Non-Standard-ISA checking instructions

3. Hardware-Assisted:

   - Embedded Device
     - Main Processor Core
       - FE, DE, EX, ME, WB
     - Guard Processor
       - FE, DE, EX, ME, WB

4. Specialized Hardware:

   - Embedded Device
     - Main Processor Core
       - FE, DE, EX, ME, WB
     - Special Hardware
     - Saved Area
Locations of hardware Integrity modules

1. In-Pipeline
   - ++ Lowest latency
   - -- Lowest portability

2. On-Chip
   - + Slightly increased latency
   - + Good portability

3. Off-Chip
   - ++ Best Portability
   - -- Very long latency
   - -- Throughput limitation
Related Work:

- RISC-V Formal Interface
- RISC-V Debug Specification
- RISC-V Trace Specification

Monitoring: 
One-way ticket, passively monitors violations
Cannot prevent evil instructions in flight
Timing: Monitoring vs. In-Time Enforcement (2/2)

Prevention: Closed loop operation, can actively prevent violations from taking effect.
Preventing the Worst-Case Attack

Timing is critical!
Prevention with RT-LIFE

RISC-V Core

RT-LIFE TAP

FE DE EX ME WB

RT-LIFE Actors

Stall or Reset via RT-LIFE

DExIE Enforcement Unit

Capture Latency (CL)

Decision Latency (DL)

Capturing status signals from uncommitted instructions via RT-LIFE
Architecture: Interface signal specification

RISC-V Core

Control Flow:
PC, Instruction, Next PC

Mem Store:
PC, Address, Size, Data

Reg Write:
PC, Register, Data

Control:
Stalls, Reset

DExIE Enforcement Unit
Design Space Exploration
Covering 6 RISC-V cores

Select a RISC-V Core!

- Piccolo
- Orca
- Taiga
- Flute
- PicoRV32
- VexRiscv

RT-LIFE: Capture

RT-LIFE: Control

DExIE Enforcement Unit
Microarchitecture (1/6)
Piccolo
Flute is similar to Piccolo, but separates more pipeline stages.
Microarchitecture (3/6)

Orca

[Diagram of the Orca microarchitecture with labeled stages: Fetch, Decode(1), Decode(2), Execute, Writeback, Cache, Writethrough, Bus. Connections indicate data flow and dependencies between these stages.]
Microarchitecture (4/6)

PicoRV32

Diagram showing the flow of instructions and data through the PicoRV32 microarchitecture, with nodes labeled `fetch`, `ldrs`, `ldmem`, `stmem`, `exec`, `shift`, `trap`, and `SoS`. The diagram outlines the flow of operations such as `Reg. Write: PC, Target Reg., Data`, `CF: PC, Instruction`, and `Mem. Store: PC, Address, Size, Data` towards the `DExIE` module. The flow is indicated by arrows and nodes.
Microarchitecture (5/6)

Taiga

![Diagram of Taiga microarchitecture with components and data flow]

- Fetch
- Pre-Decide
- Decode, Issue
- Branch Unit
- Load Store Unit
- ALU
- Bus
- Cache
- Taiga: Writeback, Retire
- CFS
- CF: PC, Instr, Next PC
- Mem. St.: Address, Size, Data
- SoS
- Reg. Write: PC, Target Reg., Data
- DExIE

Christoph Spang, Florian Meisel, Andreas Koch | TU Darmstadt | RT-LIFE: Real-Time Lightweight Integrity Enforcement Interface | 21
Microarchitecture (6/6)

VexRiscv
Evaluation
Evaluation (1/4)
Maximum Clock Frequency in MHz

- **Flute**: 90 MHz, 82 MHz
- **Orca**: 215 MHz, 215 MHz
- **Piccolo**: 140 MHz
- **PicoRV**: 340 MHz, 340 MHz
- **Taiga**: 145 MHz
- **VexRiscv**: 235 MHz, 230 MHz, 135 MHz

Plain, With RT-LIFE, With RT-LIFE & DExIE
Evaluation (2/4)
Look Up Tables (LUTs)
Evaluation (3/4)
Register Usage in Bit

![Bar Chart]

- **Plain**: Bars in brown
- **With RT-LIFE**: Bars in red
- **With RT-LIFE & DExIE**: Bars in blue

<table>
<thead>
<tr>
<th>Program</th>
<th>Plain</th>
<th>With RT-LIFE</th>
<th>With RT-LIFE &amp; DExIE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flute</td>
<td>11,851</td>
<td>11,382</td>
<td>18,389</td>
</tr>
<tr>
<td>Orca</td>
<td>4,096</td>
<td>4,077</td>
<td>3,515</td>
</tr>
<tr>
<td>Piccolo</td>
<td>8,301</td>
<td>10,696</td>
<td>17,744</td>
</tr>
<tr>
<td>PicoRV</td>
<td>3,568</td>
<td>3,568</td>
<td>3,094</td>
</tr>
<tr>
<td>Taiga</td>
<td>6,950</td>
<td>3,089</td>
<td>5,333</td>
</tr>
<tr>
<td>VexRiscv</td>
<td>3,520</td>
<td>3,464</td>
<td>7,340</td>
</tr>
</tbody>
</table>
Evaluation (4/4)
BRAM Usage in Kilobyte

![BRAM Usage in Kilobyte Chart]

- **Plaintext**: BRAM Usage in Kilobyte
- **Diagram**: Bar chart showing BRAM usage for different benchmarks with and without RT-LIFE and DExIE.
Conclusion

RT-LIFE: Real-Time Lightweight Integrity Enforcement Interface

Tap early pipeline stages → Capture uncommitted instructions

Enforcement with long and dynamic Decision Latency → Prevent evil instructions in flight from taking effect

High portability and Low resource overhead
Future Work

- Attack Prevention for Out-of-order Cores
- Reduced Capture Latency via Branch Prediction
- Dynamic Attack Responses via Low-latency Interrupts
- Runtime-Dynamic Security Enforcement Units
Open Source

RT-LIFE on GitHub

https://github.com/esa-tu-darmstadt/RT-LIFE

Made with TaPaSCo

https://github.com/esa-tu-darmstadt/tapasco