RAFT: Hardware-assisted Dynamic Information Flow Tracking for Runtime Protection on RISC-V

Yu Wang¹, Jinting Wu¹, Haodong Zheng¹, Zhenyu Ning²,¹, Boyuan He³, Fengwei Zhang¹

¹Southern University of Science and Technology
²Hunan University
³Huawei Technologies Co., Ltd.

October, 16, 2023
Dynamic Taint Analysis

```
struct SAMPLE{
    char buffer[20];
    int private_data;
};
int test()
{
    struct SAMPLE s;
    s.private_data = 5; // source ①
    signed char input_char;
    int i = 0;
    while ((input_char = fgetc(fp)) != EOF) // source ②
    {
        s.buffer[i] = input_char;
        i++;
    }
    int temp = s.private_data;
    process(temp); // sink ③
    return 0; // sink ④
}
```
High Performance Overhead

- Software-based DTA tools often suffer from unbearably high performance overhead
  - Due to Dynamic Binary Instrumentation (DBI) or Virtual Machine (VM)
  - Imposing ~4x or even ~10x slowdown

- A research interest emerged in developing hardware accelerators to improve the performance of DTA
Hardware-assisted DTA

```
mv t2, t1
sb t2 0x8000200
lb t3 0x8000200
lb t1 0x8000210
```
Hardware-assisted DTA on RISC-V

- Utilizing a coprocessor to perform analysis logic on RISC-V

**Problem 1**
- Non-negligible performance overhead (~20%)
  - Mainly coming from frequent memory operations
  - Reasonable for program analysis but is still unacceptable when protecting *time-critical applications* at runtime (e.g., medical applications and vehicle control units)
Hardware-assisted DTA on RISC-V

- Utilizing a coprocessor to perform analysis logic on RISC-V

**Problem 2**

- High memory overhead
  - For example, a flat, fixed-size structure: directly mapping the tags of every memory address into shadow memory requires massive storage space

![Diagram showing memory mapping and storage space comparison]

- 8 GiB Memory >> 1 GiB Shadow Memory
- Massive Storage Space

Shadow Memory

0x100108
0x1000c8

8 GiB Memory >> 1 GiB Shadow Memory
Goals

- A flexible hardware-assisted DTA framework on RISC-V to provide runtime protection for embedded applications *without delay to the programs*
- A new tag-storage mechanism with hybrid byte/variable granularity to *reduce the size of tag storage*
Architecture Overview

RISC-V Rocket Microprocessor

RoCC Interface
- Commit Log
  - -- inst
  - -- pc
  - -- addr
  - -- data
- Command
  - -- inst
  - -- rs1
  - -- rs2
  - -- rd
- Core Interrupt
- Coprocessor Interrupt
- Memory Request
- Memory Response

Filter Unit
- E
- D
- Full

Interrupt Manager

Control Unit
- Fetch
- Decode
- Inquiry

Tag Rules Pool
- Tag Propagation
- ALU
- Load/Store
- Sink

Tag Checking
- Security Violation

RAFT
- Tag Storage
  - Shadow Register File
  - Tag Storage File

Components

RAFT
Observation

- The way to access an allocated heap memory is usually through the start address stored on the stack.

Design

- Utilizing the information on the stack to represent the tags of heap data.
Tag-storage Mechanism

Tag Storage File (TSF)

Tagging program memory
- 2-bit tag:
  - 1 bit: taint or non-taint
  - 1 bit: address or data

Shadow Register File (SRF)

Tagging general-purpose registers
- 64-bit tag:
  - 1 bit: taint or non-taint
  - 1 bit: address or data
  - 62 bits: the location of the address stored on the stack
Collecting the runtime information about the main core and committing it to the coprocessor
Filtering out DTA-unrelated instructions that do not involve tag propagation

- Manually instrumenting two custom instructions before and after the block of unrelated instructions
Control Unit

- Controlling analysis logic
  1. Dequeue an instruction and Decode it
  2. Query tag propagation rules
  3. Perform DTA operations

\[
\text{Tag}(rd) \leftarrow \text{Tag}(rs1);
\]
\[
\text{Tag}(rd) \leftarrow \text{Tag}(rs1) \lor \text{Tag}(rs2);
\]
\[
\text{Tag}(rd) \leftarrow \text{Tag}(\text{Mem}[rs1 + \text{offset}]);
\]
\[
\text{Tag}(\text{Mem}[rd + \text{offset}]) \leftarrow \text{Tag}(rs1);
\]
• Pipelining the above tasks
• Moving tags from shadow memory to the coprocessor
• Managing the coprocessor interrupt
  • Security violation (e.g., return address comes from the system input)
  • Instruction Queue is full
Implementation

- Implementing our prototype based on PHMon (USENIX Security’20)
  - An efficient programmable hardware monitor to enforce an event–action monitoring model
  - Utilizing it to flexibly configure tag propagation rules and trace the runtime information
- Raft is deployed on the RISC-V Rocket emulator and Xilinx Kintex-7 FPGA KC705 evaluation board
- Custom instructions (RISC-V ISA)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint rs1, rs2</td>
<td>Mark the taintedness</td>
</tr>
<tr>
<td>src rs1, rs2</td>
<td>Mark heap pointer variables</td>
</tr>
<tr>
<td>arg rd, rs1</td>
<td>Assist tag propagation</td>
</tr>
<tr>
<td>open/close zero, zero</td>
<td>Filter out DIFT-unrelated instructions</td>
</tr>
<tr>
<td>sink rs1, rs2</td>
<td>Perform security checks</td>
</tr>
<tr>
<td>base fp/gp, rs2</td>
<td>Pass the frame pointer and global pointer to the coprocessor</td>
</tr>
</tbody>
</table>

- Toolchain (LLVM 12.0.1) and OS support (Linux v5.4)
Performance Evaluation

Comparison against Traditional Storage

- **Shadow Memory**: 22.54% overhead on CoreMark and 26.37% overhead on NBench
- **RAFT**: <0.1% overhead on CoreMark and NBench

- **RAFT** effectively cuts down the performance overhead from >20% to <0.1% with our new tag-storage mechanism
Performance Evaluation

- SPEC CINT 2006: 0.13% overhead on average

- We set up a main core with a peak instruction processing rate 2x and 3x that of the coprocessor and rerun the CoreMark
  - RAFT still introduces negligible performance overhead
**Hardware Resource Cost**

- An extra usage of 34.04% LUTs and 93.17% FFs
- 2.31% power overhead

<table>
<thead>
<tr>
<th>Whole System</th>
<th>Slice LUTs</th>
<th>Slice Registers</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without RAFT</td>
<td>58,442</td>
<td>29,445</td>
<td>3.46 W</td>
</tr>
<tr>
<td>With RAFT</td>
<td>78,355</td>
<td>56,879</td>
<td>3.54 W</td>
</tr>
<tr>
<td>%</td>
<td>+ 34.07%</td>
<td>+ 93.17%</td>
<td>+ 2.31%</td>
</tr>
</tbody>
</table>
• A medical embedded application OpenSyringePump
  • Detecting a non-control data attack by exploiting a buffer overflow vulnerability
• 5 known CVEs

<table>
<thead>
<tr>
<th>ID</th>
<th>CVE ID</th>
<th>Program</th>
<th>Vulnerability</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CVE-2009-4496</td>
<td>Boa</td>
<td>Information Leakage</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>CVE-2014-8503</td>
<td>Size</td>
<td>Buffer Overflow</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>CVE-2016-3186</td>
<td>Gif2tiff</td>
<td>Buffer Overflow</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>CVE-2018-17100</td>
<td>Ppm2tiff</td>
<td>Integer Overflow</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>CVE-2010-0001</td>
<td>Gzip</td>
<td>Integer Underflow</td>
<td>✓</td>
</tr>
</tbody>
</table>
Conclusion

- A flexible hardware-assisted DTA framework that provides runtime protection for embedded applications without delay to the programs
- A new storage mechanism with hybrid byte/variable granularity
- A prototype on the RISC-V Rocket emulator and FPGA development board

https://github.com/Compass-All/Raft

Thanks!
You can reach me at 12032879@mail.sustech.edu.cn for follow-up questions

I am looking for PhD positions in System Security