Raven: A Novel Kernel Debugging Tool on RISC-V

Fengwei Zhang

COMPASS Lab

Southern University of Science and Technology

Outline

- Motivation
- Design & Implementation
- Case Study
- Performance Evaluation
- Limitations
- Future Directions
- Conclusion

Existing Debugging Approaches on RISC-V

Software Debugging

- Require Hypervisor
  - QEMU, KVM, etc.
- Intrusive Injecting **`ebreak`**
  - Breaks integrity
- Tied to Specific OS
  - kGDB, WinDBG, etc.

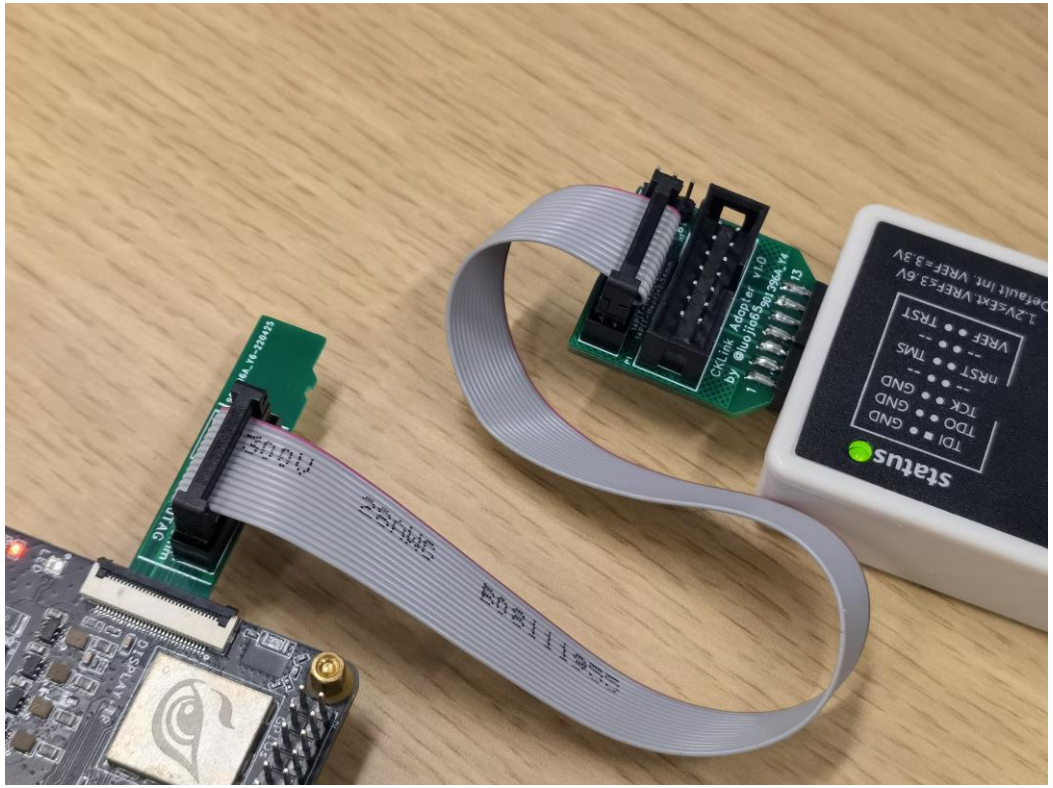
Hardware Debugging

- Vendor Restriction on JTAG
  - No debugging port
- Divergent Implementation
  - JLink, CKLink, etc.
- Expensive Debugger
  - JLink: ~500 USD
  - CKLink: ~300 USD

Example: Nezha D1

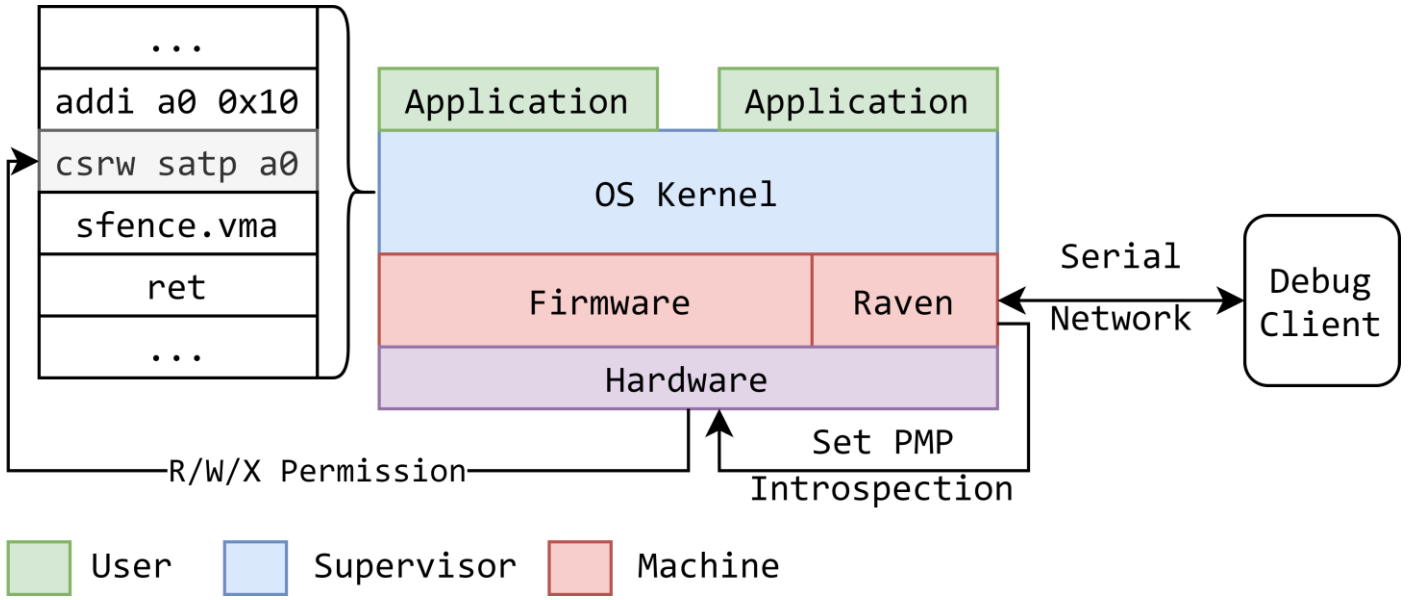
A RISC-V SoC with XuanTie C906 single core 64-bit CPU

- Special debugging probe called CKLink (incompatible to JLINK)
- Debugging port is hidden in SD slot (special adapter needed)



Design Overview

- Non-invasive Debugging
  - Use PMP instead of ebreak
- No Hypervisor
  - Based on baremetal firmware
- No Special Hardware
  - Software does the heavy lifting



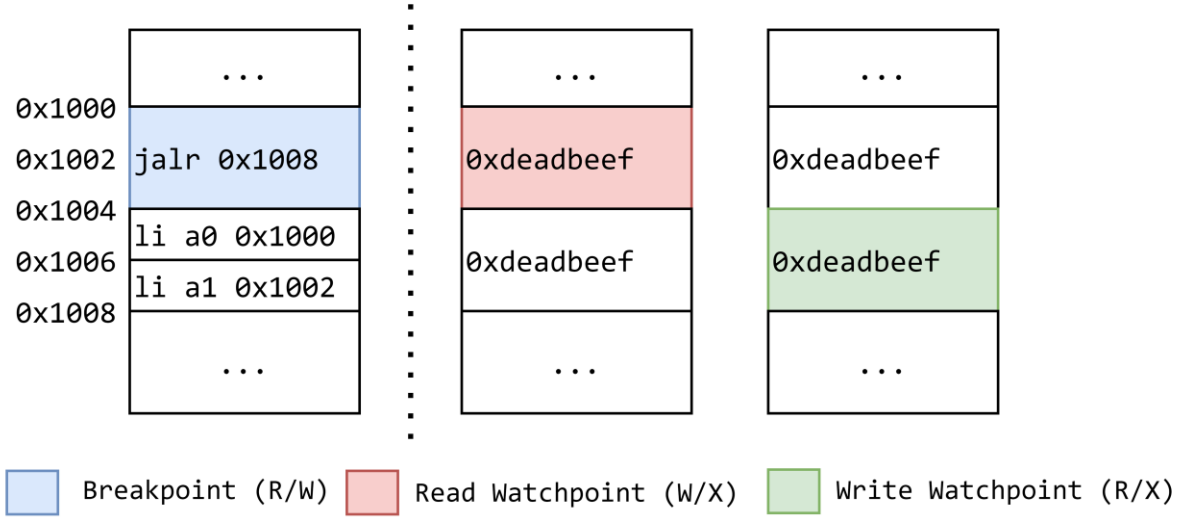
What is PMP?

A physical memory protection mechanism of RISC-V.

- Granularity: 4 bytes~4 kilobytes
- Permission: R/W/X restrictions in S/U modes
- Violation  Exception

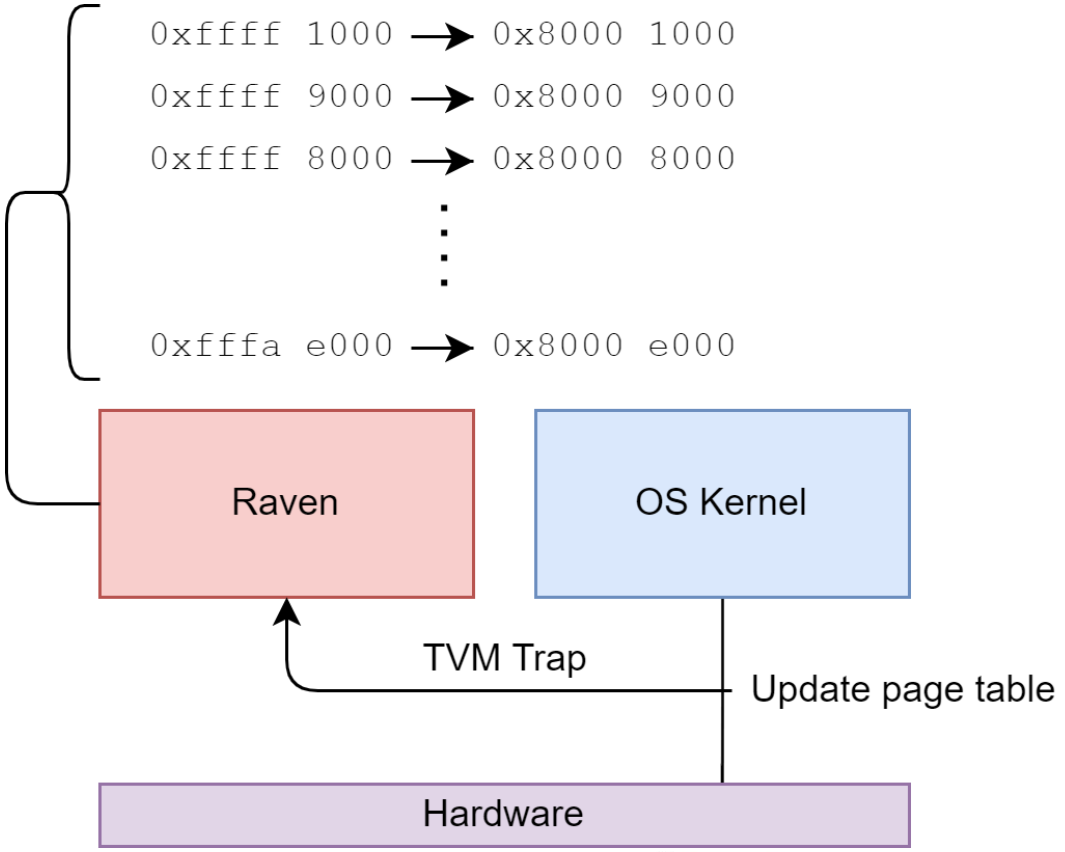
PMP as Debugging Primitives

- Breakpoint
  - Set instruction as non-executable to trap into firmware
- Watchpoint
  - Set data as non-readable/non-writable to have R/W watchpoints



Page Table Synchronization

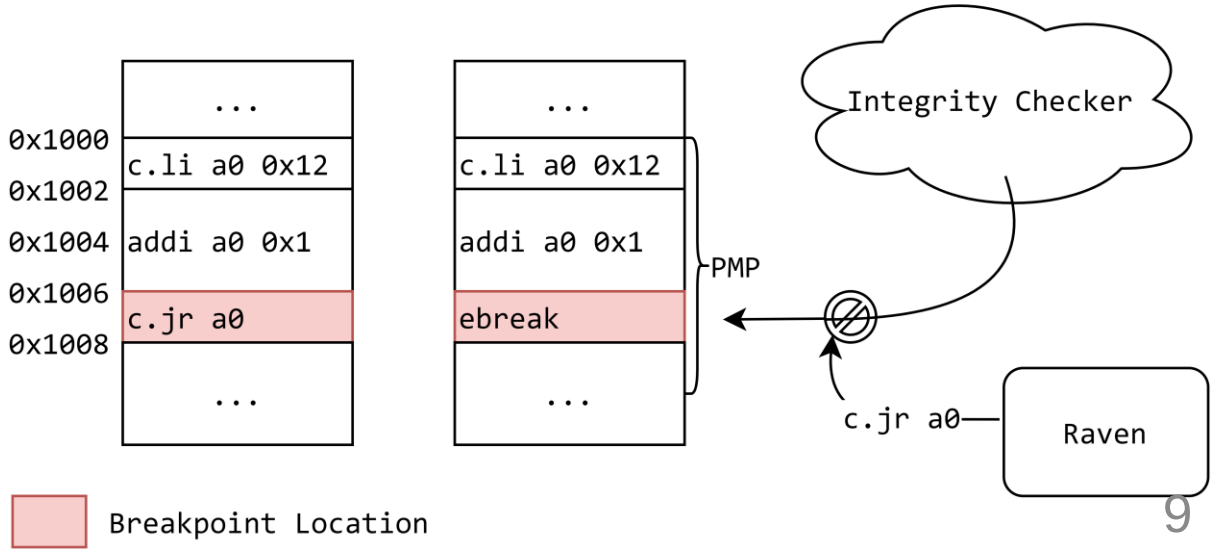
- PMP only recognizes physical address.
  - We leverage TVM (Trap Virtual Memory) to perform synchronization
- TVM will trap **`sfence.vma`** and page table updates.
  - Raven uses this trap to look up physical address and config PMP.



Coarse Granularity Solution

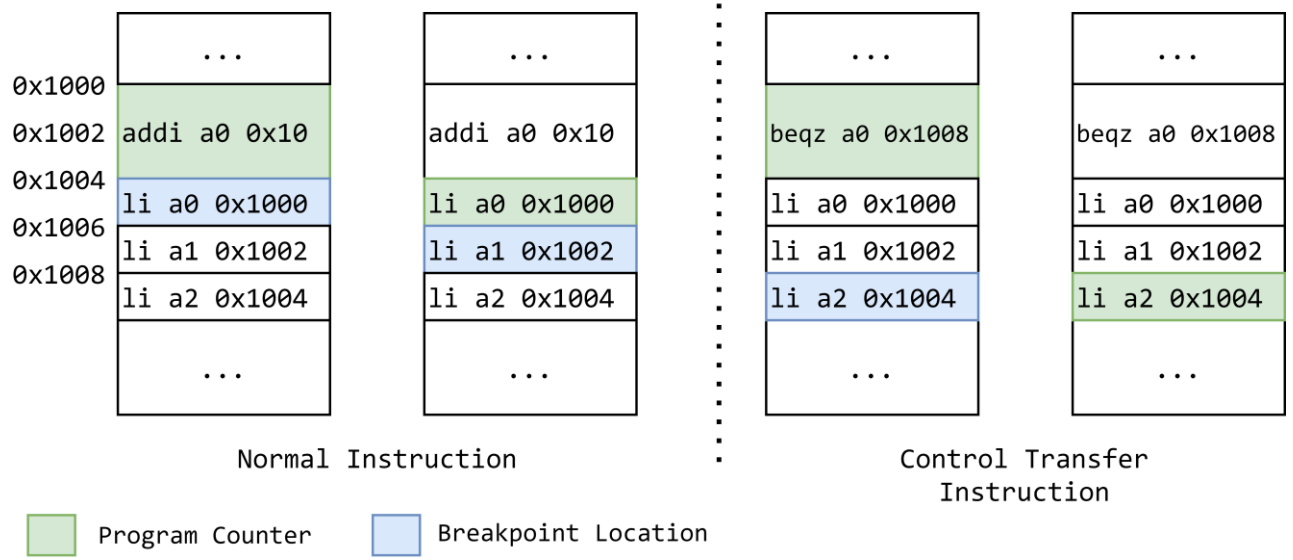
- Granularity varies, not all can be used as breakpoints
- Fallback to **`ebreak`** without breaking integrity

Board	# PMP	Granularity
QEMU Virtboard	16	4 byte
HiFive Unleashed	8	4 byte
HiFive Unmatched	8	4 kilobyte
HiFive Rev B	8	4 byte
Allwinner Nezha D1	8	4 kilobyte



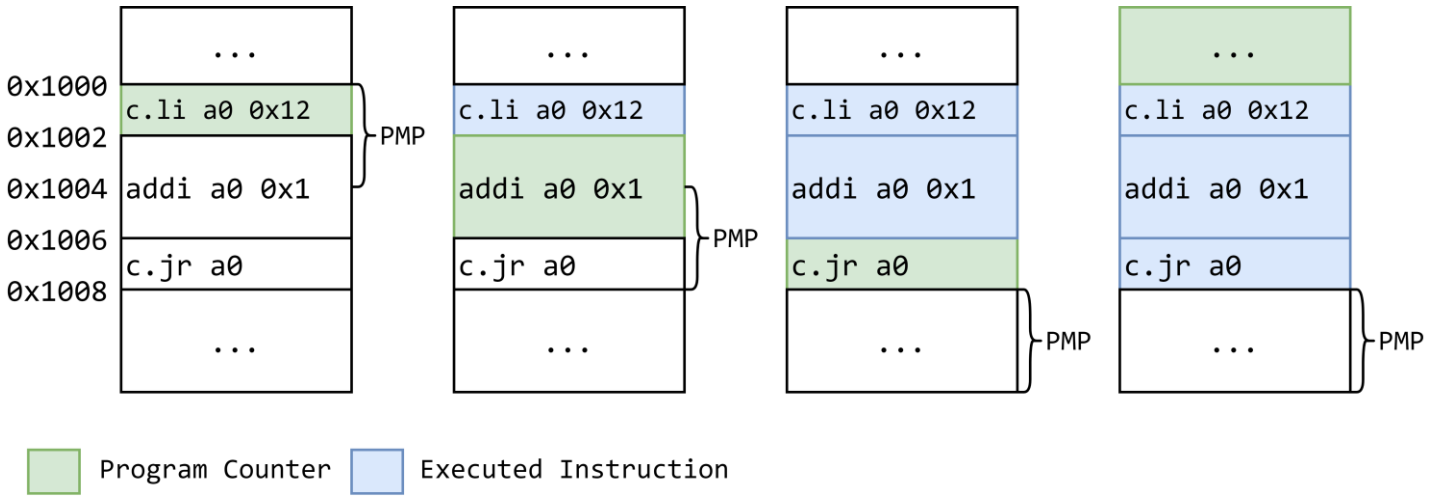
Primitives to Single Stepping

- Normal Instruction
  - Setup breakpoints following PC
- Jump Instruction
  - Decode and predict its destination



Hidden Instructions

- Finest granularity: 4 bytes
- Instruction length: 2 bytes
  (“C” Extension ISA)



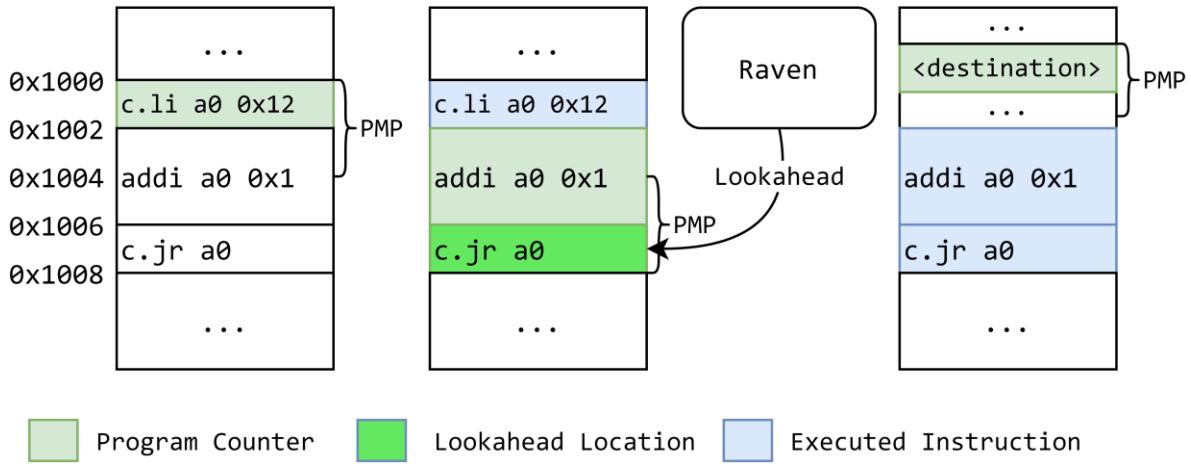
This leads to the “hidden” instructions

Look-ahead Technique

Look-ahead happens when

1. Instruction & PMP misaligned
2. The instruction is a jump

Similar tricks can be used for asynchronous events like IRQ.



Functions of Raven

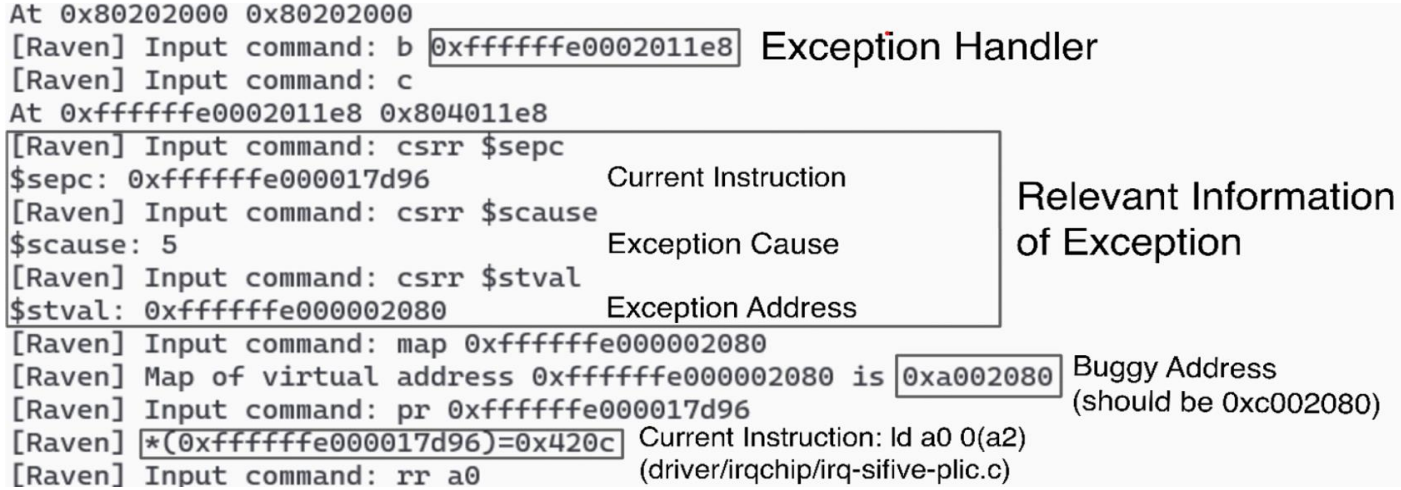
- Raven supports most debugging function of a hardware debugger.
- Making it easy to integrate Raven with frontends like GDB

Command Format	Description
b <address>	Set a breakpoint at <address>
w <address>	Set a watch point at <address>
pr (pw) <address>	Read(Write) memory content at <address>
rr (rw) <reg>	Read(Write) register content of <reg>
map <address>	View the memory mapping of <address>
csrr (csrw) <csr>	Read(Write) control status register of <csr>
s	Single-step execution
c	Continue execution after a breakpoint
<GPIO Switch>	Send an external interrupt to halt the kernel

Case Study: Buggy Device Tree

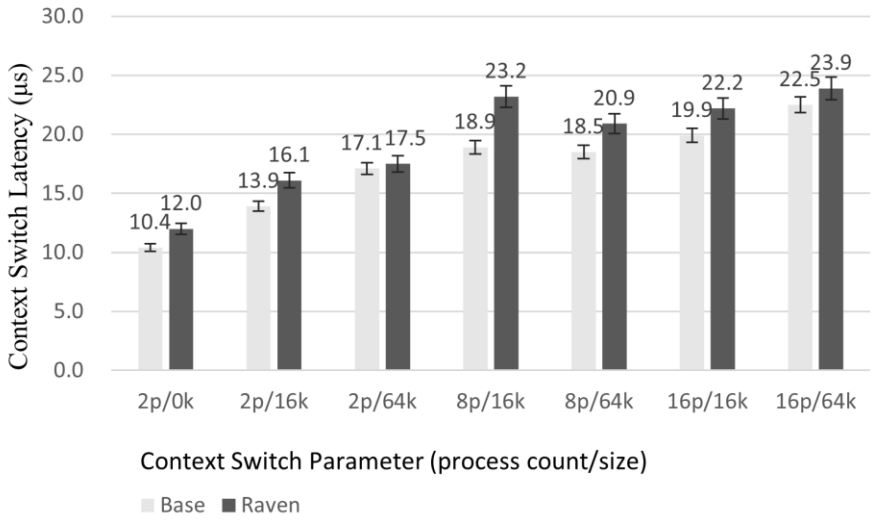
Steps

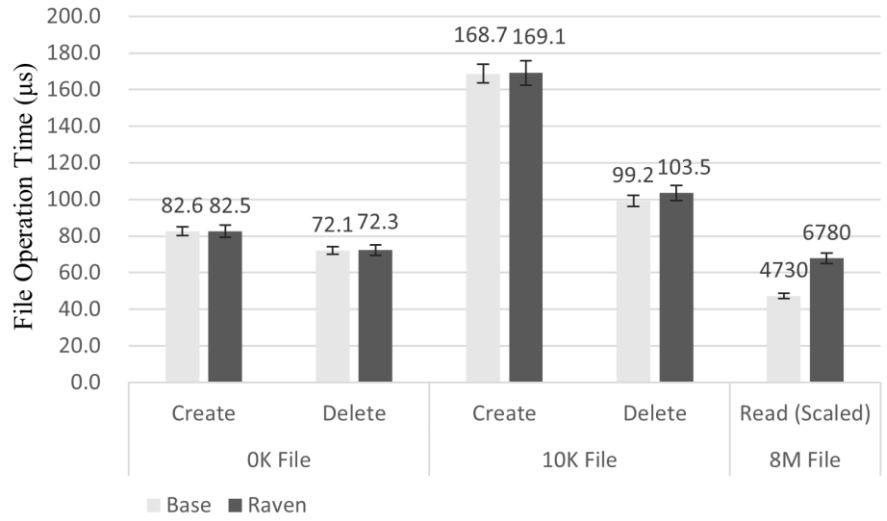
1. Craft a buggy device tree
2. Boot Linux -> kernel crash
3. Using Raven to locate & fix



Overhead

We use Lmbench to evaluate Raven's performance overhead.
Both experiments are tested with one dummy breakpoint (which does not halt the kernel).





Limitations

- May interfere regular usage of PMPs
  - TEE, Isolation, etc
- No instruction-level precision
  - Misalignment -> Hidden instruction
- There exists bypass to PMPs
  - DMAs, co-processor, etc

What else?

- Trace on multi-core
  - Each core has its own PMP
- Cooperation with GDB
  - Use GDB as debugging client for better usability
- Integration with PMU like Ninja did
  - More transparency

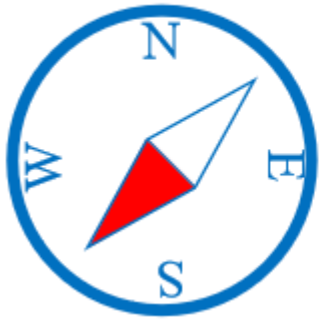
Conclusion

We summarize our work as follows

1. We propose a new approach to debug kernel on RISC-V with PMP

2. We implement its prototype and prove that it is largely equivalent to a hardware debugger

3. Raven is a non-invasive debugger without external hardware



COMPASS Lab

COMPuter And System Security Lab

计 算 机 系 统 安 全 实 验 室

COMPASS Research Interests:

- Hardware-assisted Security
- Transparent Malware Analysis
- Transportation Security
- TEE on Arm/x86/RISC-V
- Arm Debugging Security
- Plausible Deniability encryption

COMPASS Lab

COMPuter And System Security Lab

计 算 机 系 统 安 全 实 验 室

Thank You!

Contact:

zhangfw@sustech.edu.cn