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

<table>
<thead>
<tr>
<th>Board</th>
<th># PMP</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEMU Virtboard</td>
<td>16</td>
<td>4 byte</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>8</td>
<td>4 byte</td>
</tr>
<tr>
<td>HiFive Unmatched</td>
<td>8</td>
<td>4 kilobyte</td>
</tr>
<tr>
<td>HiFive Rev B</td>
<td>8</td>
<td>4 byte</td>
</tr>
<tr>
<td>Allwinner Nezha D1</td>
<td>8</td>
<td>4 kilobyte</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Command Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b &lt;address&gt;</td>
<td>Set a breakpoint at &lt;address&gt;</td>
</tr>
<tr>
<td>w &lt;address&gt;</td>
<td>Set a watch point at &lt;address&gt;</td>
</tr>
<tr>
<td>pr (pw) &lt;address&gt;</td>
<td>Read(Write) memory content at &lt;address&gt;</td>
</tr>
<tr>
<td>rr (rw) &lt;reg&gt;</td>
<td>Read(Write) register content of &lt;reg&gt;</td>
</tr>
<tr>
<td>map &lt;address&gt;</td>
<td>View the memory mapping of &lt;address&gt;</td>
</tr>
<tr>
<td>csrr (csrw) &lt;csr&gt;</td>
<td>Read(Write) control status register of &lt;csr&gt;</td>
</tr>
<tr>
<td>s</td>
<td>Single-step execution</td>
</tr>
<tr>
<td>c</td>
<td>Continue execution after a breakpoint</td>
</tr>
<tr>
<td>&lt;GPIO Switch&gt;</td>
<td>Send an external interrupt to halt the kernel</td>
</tr>
</tbody>
</table>
Case Study: Buggy Device Tree

Steps

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

At 0x80202000 0x80202000
[Raven] Input command: b 0xfffffffffe0002011e8
[Raven] Input command: c
At 0xfffffffffe0002011e8 0x804011e8
[Raven] Input command: csrr $sepc
$sepc: 0xfffffffffe00017d96
[Raven] Input command: csrr $scause
$scause: 5
[Raven] Input command: csrr $stval
$stval: 0xfffffffffe00002080
[Raven] Input command: map 0xfffffffffe00002080
[Raven] Map of virtual address 0xfffffffffe00002080 is 0xa002080
[Raven] Input command: pr 0xfffffffffe00017d96
[Raven] *0xfffffffffe00017d96=0x420c
Current Instruction: id a0 0(a2)
(driver/irqchip/irq-sfive-plic.c)

Exception Handler

Exception Cause

Exception Address

Buggy Address

(should be 0xc002080)

(driver/irqchip/irq-sfive-plic.c)
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 Research Interests:

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

Contact:

zhangfw@sustech.edu.cn