UNIVERSITÀ
DI TORINO

Heuristic-free system call interception on RISC-V

Ottavio Monticelli, Marco Edoardo Santimaria, Marco Aldinucci, Iacopo Colonnelli

Computer Science Department, University of Turin

RISC-V Summit Europe 2026 - Bologna - 09/06/2026

Heuristic-free system call interception on RISC-V

- System call interception techniques
- Binary rewriting
- vpoline

System call interception techniques

System call interception enables user-defined code to be executed **before / instead of / after** a system call invocation. Needed in several use cases:

- Monitoring and debugging
- OS emulation
- Security (isolating applications from kernel)
- Performance optimization
- Many more purposes...

System call interception techniques

Two macro-categories:

- Kernel interfaces (strace, Syscall User Dispatch (SUD), seccomp-bpf, etc)

  ✅ **Common interface across different CPU architectures**
  ❌ **Significant overhead caused by inner working**

- Binary rewriting libraries (syscall_intercept, zpoline, lazypoline, etc)

  ✅ **Full interception is handled in user-space introducing minimal overhead**
  ❌ **Deeply dependant on target ISA characteristics, often heuristic-based**

Heuristic-free system call interception on RISC-V

- System call interception techniques
- Binary rewriting
- vpoline

Binary rewriting

- HPC applications impose strict performance requirements
  - ↦ interception overhead must be as negligible as possible

- **Binary rewriting** based tools are the go-to choice in HPC context. However:
  - Binary rewriting interception library availability doesn't imply multi-architectural support
  - Most of existing solutions rely on heuristic-based algorithms

Binary rewriting: idea

Binary rewriting:

1. Search for `ecall` occurrences in a target executable
2. Replace `ecall` with jumps to user-defined code **without altering the original program's context**.

This requires more than the 4 bytes occupied by `ecall` encoding

Binary rewriting: example with clone()

Let's analyze the **`clone()`** function from the GLIBC library:

```
00000000000b3b54 <__clone>:
b3b54:    c.andi a1,-16
b3b56:    c.beqz a0,b3b78
b3b58:    c.beqz a1,b3b78
b3b5a:    c.addi a1,-16
b3b5c:    c.sd   a0,0(a1)
b3b5e:    c.sd   a3,8(a1)
b3b60:    c.mv   a0,a2
b3b62:    c.mv   a2,a4
b3b64:    c.mv   a3,a5
b3b66:    c.mv   a4,a6
b3b68:    addi   a7,zero,220
b3b6c:    ecall
b3b70:    blt    a0,zero,b3b7a
...
```

PATCH

```
00000000000b3b54 <__clone>:
b3b54:    c.andi a1,-16
b3b56:    c.beqz a0,b3b78
b3b58:    c.beqz a1,b3b78
b3b5a:    c.addi a1,-16
b3b5c:    c.addi sp,-16
b3b5e:    c.sdsp ra,0(sp)
b3b60:    c.sdsp t0,8(sp)
b3b62:    auipc  t0,%pcrel_hi(dst)
b3b66:    jalr   ra,t0,%pcrel_lo(dst)
b3b6a:    c.ldsp ra,0(sp)
b3b6c:    c.ldsp t0,8(sp)
b3b6e:    c.addi sp,16
b3b70:    blt    a0,zero,b3b7a
...
```

Binary rewriting: heuristic approaches

`ecall` encoding just provides 4 bytes suitable for rewriting; jumping while preserving full context requires 20 bytes ↦ not always possible

Existing libraries adopt heuristic solutions: **unreliable** as they make assumptions on the binary layout

```
00000000000ad946 <__fcntl64_nocancel_adjusted>:
...
ad95c:    beq     a1,a4,ad97c
ad960:    ecall
ad964:    c.lui   a4,0xfffff
ad966:    bltu    a4,a0,ad9b2
...
```

Binary rewriting: **zpoline (Yasukata et al.)**

zpoline is the first **totally heuristic-free solution**, however:

- zpoline and and zpoline-based solutions **only** target **`amd64`**
- They **require root privileges** to map eXecutable-Only-Memory (XOM) memory at address 0x0

```
0000000001148f0 <__write>:
...
114900:     mov     $0x1,%eax
114905:     syscall
...
```

PATCH

```
0000000001148f0 <__write>:
...
114900:     mov     $0x1,%eax
114905:     callq   *%eax
...
```

Heuristic-free system call interception on RISC-V

- System call interception techniques
- Binary rewriting
- vpoline

vpoline: RISC-**V** tram**poline**

First **heuristic-free** RISC-V system call interception library

- **NO root requirement** ↦ suitable for HPC applications
- Leverages `LD_PRELOAD` linker feature
- Provides an internal `SIGSEGV` handler
- Provided interface lets user decide if intercepted system call must be still executed

vpoline: main steps

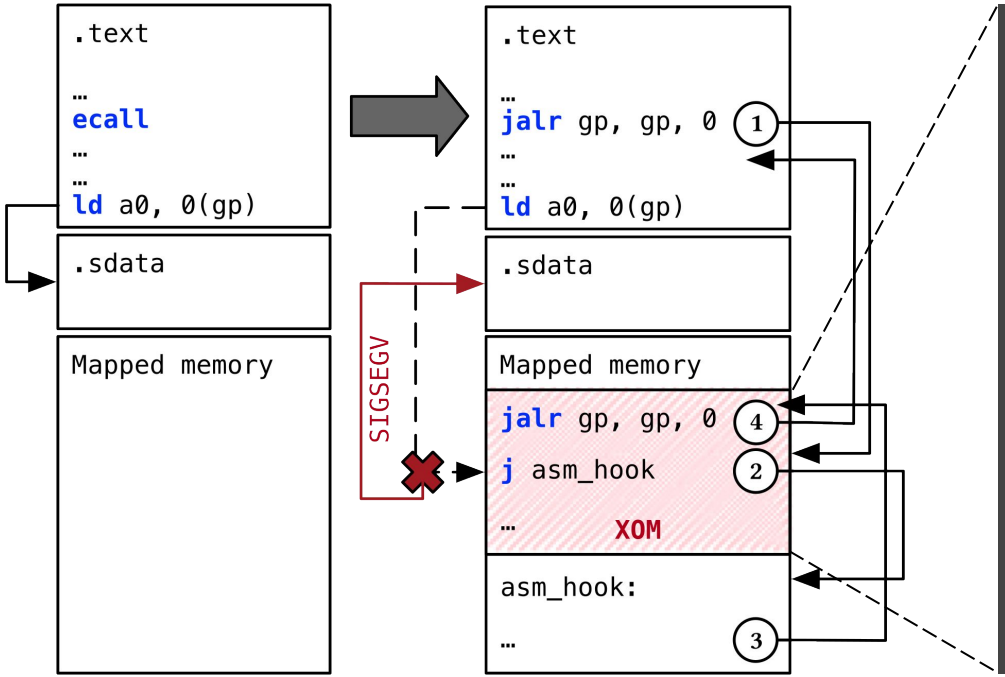
Leverages linker relaxation mechanism: `gp` is normally used to hold the fixed `__global_pointer$` symbol value (e.g. `.sdata+0x800`)

1. Store aside the `gp` value at preload time
2. Allocate a XOM memory region with trampoline towards user-defined code and a return sequence before the trampoline
3. Populate `gp` with the trampoline address (`surrogate_gp`)
4. Initialize a `SIGSEGV` handler to redirect `gp`-based read-write access which would now point to the XOM region

vpoline: implementation highlight

- Interception logic is programmed in assembly while the sequences encoded at runtime requires machine language programming
- **`ecall`** replaced by **`jalr, gp, gp 0`**; return address is saved in `gp` and 4 bytes before trampoline (pointed by `gp`), we place **`jalr gp, gp, 0`**, acting as a **`ret`** pseudo-instruction
- `gp`-based accesses are managed with the `SIGSEGV` handler redirecting read-write accesses from the XOM page to region pointed by the original `gp` value

vpoline




```
0:
...
surrogate_gp-56:
/*
 * return sequence is 56 bytes
 * long and finishes with a
 * jalr gp, gp, 0 acting as ret
 */
surrogate_gp-4: jalr gp, gp, 0
surrogate_gp: j asm_syscall_hook
/*
 * this trampoline is an 80
 * bytes instructions sequence
 */
...
```

UNIVERSITÀ
DI TORINO

Questions?

Ottavio Monticelli

ottavio.monticelli@unito.it

https://github.com/alpha-unito/vpoline