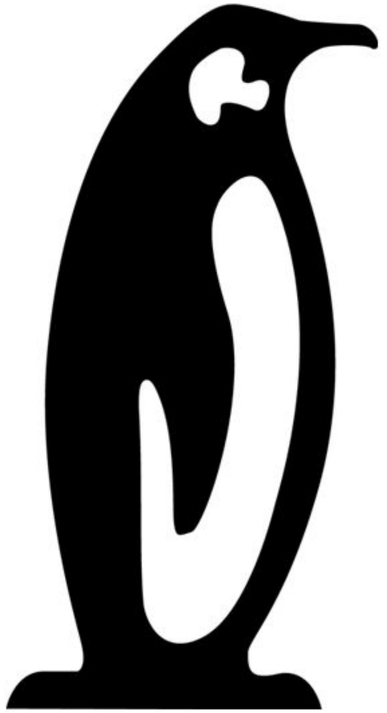

Linux Plumbers Conference

Vienna, Austria | September 18-20, 2024

Deepak Gupta: <debug@rivosinc.com>

ISA recap “Zicfilp” - protects forward control flow

Zicfilp: Enforces all indirect branches must land on `lpad` (`auipc rd=x0`)

- Except when `rs1 == (x1 | x5 | x7)`
- Label setup in `x7` must match label encoded in `lpad` instruction on target
- New exception (cause = 18) – software-check exception
  - *tval = 2, missing `lpad` or label didn’t match

lui x7,0x1 ← label setup in x7
jalr a5 ← expects landing pad at target `foo_lpad_loc`

auipc x7, <offset> ← addr of `func_body_bar`
jalr x7 ← No landing pad expected

```
foo_lpad_loc:
        lpad <label>
func_body_foo:
```

```
func_body_bar:    ← No label expected
```

auipc rd=x0 ***is HINT NOP***

ISA recap “Zicfiss” - protects return control flow

Zicfiss: Extends architecture with shadow stack (encoding `RWX = b010`)

- Regular stores not allowed. Regular loads allowed.
  - Access fault on regular stores.
- Shadow stack memory accesses strictly operate on shadow stack memory
  - SS access on RO memory ⟶ store page fault
  - SS access on RWX or XO memory or RW memory ⟶ access fault
  - `sspopchk` can raise software-check exception (*tval = 3)

```
func_main:
    lpad  <label>
    sspush x1            ← push return address on top of shadow stack
    …
    …

    ld x5, offset(sp)    ← get return address from stack
    add sp, sp, offset   ← adjust stack
    sspopchk x5          ← pop from top of shadow stack and compare with x5
    jr x5                ← sspopchk didn’t fault. Return back
```

ISA recap “zimop” and “zcmop”

zimop : “may be operation”

- If implemented, moves `0` → `rd`
- Allows future CPU extensions looking for instruction encodings
  - That don’t fault if feature is disabled or not implemented
  - Existing spec to maintain sanity that HINT NOPs never change architectural state
- RVA23 profile mandates zimop implementation
- But zimop compiled binaries will exhibit illegal instruction on existing / old hardware

zcmop: “compressed may be operation”

- Ditto as zimop except doesn’t do `0` →`rd`
- Instead allows future extensions to read encoded register

Shadow stack (“zicfiss”) extension is first consumer of zimop and zcmop

More info here on zimop and zcmop

Update since last time

CPU extensions (zicfilp and zicfiss) are ratified

Toolchain:

- Toolchain / gcc : Thanks to Kito and team
- Toolchain / llvm : Thanks to Ming-Li (only landing pad but this enables function signature based multi-label scheme)

Upstream status:

- Going with `VM_SHADOW_STACK` vma flag used by x86 (and ongoing arm64 series as well) : Shadow stack only on 64 bit
- User mode control enabling of feature for its entire address space (via new proposed generic `prctl`)
- Qemu patches for enabling zicfilp and zicfiss
- Revised kernel patches for user mode control flow integrity
- Kernel patches (RFC) for kernel control flow integrity
- Opensbi patches for zicfilp and zicfiss

Opens / Discussion

vDSO management

- Shadow stack instructions come from zimop encodings
- Kernel doesn’t know if target CPU has zimop

Landing pad label scheme

- Label setup in `t2/x7` must match label embedded in `lpad`
- All exec objects in address space *must* have same label scheme
- Compiled vDSO also must have same label scheme

vDSO management

Options

- Patch vDSO in runtime: Too cumbersome and flaky
- Follow RISC-V profiles and build vDSO based on that: No profile awareness as of today
- Carry two vDSO, one with zimop and one without: Small size, doable
  - Some future CPU extensions might also take some of the zimop encodings (e.g. memory tagging).
  - Only need to add one more compile command-line for vDSO generation
- Once kernel is compiled with shadow stack support (or any zimop instruction encoding)
  - Start carrying a single vDSO again

Landing pad label scheme

Zero label scheme: `lpad #0`

- Embedding of `0` in `lpad` means label check is bypassed
- Simple and can be easily adopted
- *But* any callsite can reach any target and thus much coarser grained cfi

Multi-label scheme: `lpad #imm_label_func_sign_trunc_20bit`

- Using 20 bit truncated hash of function signature as label value
- A callsite can only go to target locations with matching function prototype: finer grained cfi
- Func signature multi-label scheme is somewhat similar to FineIBT (in upstream) for x86 linux kernel

Yes there are some difficulties with multi-label scheme

- But nothing which can't be solved
- More discussion on mitigating these difficulties on this pull request on psABI for multi-label scheme: Thanks to Kito, Ming-Li & many others

Landing pad label scheme – real problem

Real problem actually is possibly of existence of more than *one* labeling scheme

- An executable object compiled with one label scheme can’t indirectly call into object with different label scheme
- More than one labeling scheme leads to fragmentation in package / library management (not to mention vDSO management)
- In practice, only one label scheme will be used
- Landing pad enabling in conjunction with shadow stack enabling → New binaries with zimop dependency
  - Mixing of legacy binaries only possible on machines with zimop enabled hardware (doesn’t exist today)
  - zimop enabled binaries can’t run on old hardware
- Allows a clean break from old CPUs → new CPUs

Use the opportunity and enable multi-label scheme

Discussion & Question