

TOKYO, JAPAN / DECEMBER 11-13, 2025

# Kernel control-flow integrity using RISC-V CFI extensions

Deepak Gupta (Rivos Inc.) debug@rivosinc.com





# Why?

Memory safety bugs allow attackers to manipulate pointers responsible for control-flow

- Return address on stack "backward control-flow integrity"
- Function pointers in read-write memory "forward control-flow integrity"

Linux kernel development cycles are intense and dense

- Bugs aren't dying
- Kees Cook has held bullhorn □ on the topic for over a decade (<a href="https://outflux.net/slides/2025/lss/kspp-decade.pdf">https://outflux.net/slides/2025/lss/kspp-decade.pdf</a>)



# forward control-flow integrity (fcfi)

An indirect control transfer must always land on a landing pad instruction

- lpad on RISC-V, endbr on x86 and bti on arm64
- reduces # of locations an indirect callsite can reach
- indirect call sites can reach location with landing pad (i.e. all the functions) ← coarse-grained

A fine-grained approach is better - tying a callsite to specific target address

#### Solutions

- kCFI (based on clang/Ilvm) → software instrumentation
  - GCC kCFI ABI work in progress
- FineIBT approach on x86 → software/hardware co-design



# backward control-flow integrity (bcfi)

Return addresses saved on non-tamperable shadow stack and on return must match return address saved on regular stack

- By nature fine-grained approach for backward edge

## Solution(s)

- Shadow call stack (SCS) → software based shadow stack (not protected)
- Upstream kernel is lacking support for hardware assisted kernel shadow stack



## RISC-V kernel fcfi

RISC-V ISA provides hardware support for fine-grained fcfi using 20-bit label value

- callsite can setup a static value (20b truncated hash over function type) in x7 register.
- lpad instruction at target can be encoded /w 20b immediate value.
- mismatch between x7 and immediate encoded in 1pad raises an exception

riscv-gnu-toolchain configured with --with-label-scheme=func-sig

 enables md5 hash over function signature (follows Itanium C++ mangling ABI) and sets up call sites and targets appropriately

Huge shoutout and thanks to Kito Cheng and Monk Chiang from SiFive



## RISC-V kernel bcfi

Piggybacks on software approach of shadow call stack (CONFIG\_SCS) with following difference

- Shadow stack is protected using hardware assistance (PTE encoding)
- Grows high to low memory (unlike software shadow callstack)
- Instead of gp register, uses CSR\_SSP register to track shadow stack pointer

#### Introduces

CONFIG\_ARCH\_HAS\_KERNEL\_SHADOW\_STACK in mm/KConfig



# RISC-V kernel cfi – putting it together

### Preparatory patches to

- Place 1pad at hand-coded assembly routines
- Convert assembly callsites in software guarded branches (if possible)

Introduces CONFIG\_RISCV\_KERNEL\_CFI in arch/riscv/KConfig

- Depends on CONFIG\_RISCV\_USER\_CFI
- Lights up -fcf-protection=full for kernel compile
- Selects CONFIG\_ARCH\_HAS\_KERNEL\_SHADOW\_STACK

https://lore.kernel.org/all/alLD8LeUypdAKc8a@debug.ba.rivosinc.com/



## Opens for discussions

#### **Toolchain**

- Need assembler support to get hash over function signature
  - Something like lui t2, %lpad\_hash(func\_type)%
- Probably makes sense to converge with GCC KCFI typeinfo ABI effort

## text patching

- breakpoints/kprobe/kretprobe
- ftrace

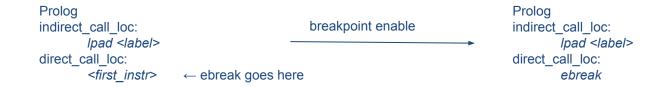
#### Control transfers between runtimes

- Kernel module loading without cfi instructions
- eBPF jitted code
- UEFI runtime services

Optimized kernel shadow stack allocation and deallocation



# Breakpoints and text patching in kernel /w CFI



## Proposal

- Setting breakpoint must not patch lpad
- Subsequent instruction is patched
- Normal breakpoint handling is followed



10

## kprobes and kretprobes

#### kprobes

Same as breakpoint handling

### kretprobe: probes on function returns

- Installs a kprobe on function entry
- kprobe handler does pt\_regs->ra = arch\_rethook\_trampoline
  - Saves away original ra
- arch\_rethook\_trampoline gets called on return and calls retprobes
  - Eventually does jr to original ra
- None of this violates Zicfilp or Zicfiss



# CFI and tracing support

## Proposal

- Currently tracing enable uses jalr x5, x5 ← should work as is
  - landing pad not expected on target trampoline
  - Return saved in x5
  - Target trampoline uses x5 on return path (rs1 == x5 doesn't require landing pad)
  - lpad can't be patched



TOKYO, JAPAN / DEC. 11-13, 2025

12

## Control transfer between different runtimes

## Load (insmod) kernel module not compiled with cfi?

- simply not load such kernel module(s)
- Proposal: to prevent disruption and easy adoption
  - Create kernel command line parameter riscv\_kernel\_cfi
  - default is riscv\_kernel\_cfi=off (even if CONFIG\_RISCV\_KERNEL\_CFI=y)
  - User must explicitly opt into riscv kernel cfi via riscv\_kernel\_cfi=on

#### **UEFI** runtime services

disable landing pad and shadow stack before switch\_mm and jumping to UEFI

## eBPF jitting

- Insert sspush/sspopchk in prolog/epilogue
- No idea on how to do labeling of landing pad
  - How does clang based kCFI work with it?



## Kernel shadow stack allocation and deallocation

Current patches allocate kernel shadow stack from vmalloc. Two problems

- Alternate map (direct map) exists and is vulnerable as per threat model
- Permissions change is required on vmalloc/free (thus required TLB shootdowns)
- Permissions changes and TLB shootdowns will impact fork (in multi-core setup)
  - This will show up as problem for other arches too

#### Possible solutions

- Create a kernel shadow stack allocator which uses vmalloc
  - Kernel shadow stack allocator ensures that allocated range direct map is unmapped
- YOLO: go with current solution and improve later on

4K shadow stack page interleaved /w guard page around it should be good enough

- N shadow stack require N+1 guard pages. Total virtual memory needed = (2N+1)\*4K
- 10K kernel shadow stacks = 84MB range in kernel address space / vmalloc range

