



Beyond Coverage: Per-Task Function Boundary Extraction for Kernel Contract Verification

`-fsanitize-coverage=trace-args/ret`

LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

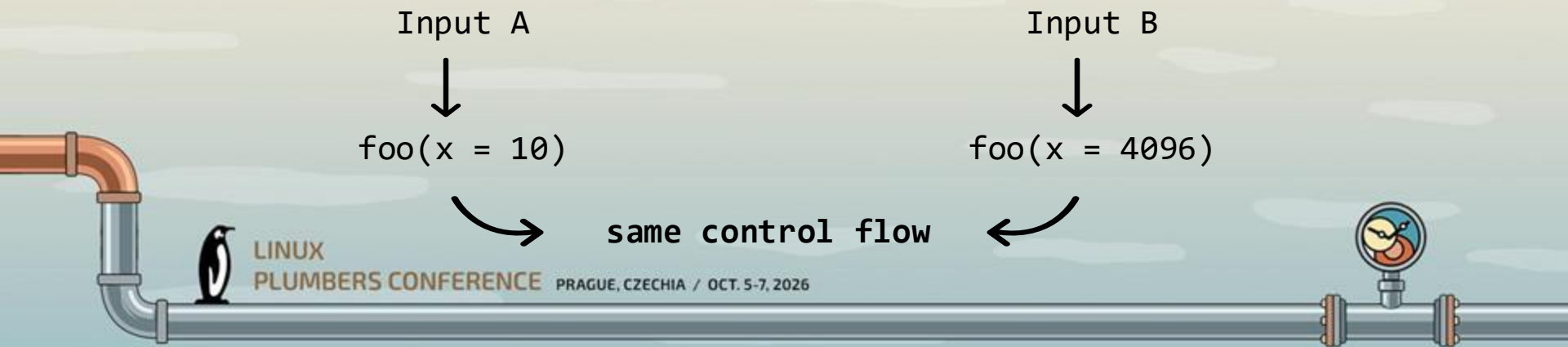
Yunseong Kim <yunseong.kim@est.tech>

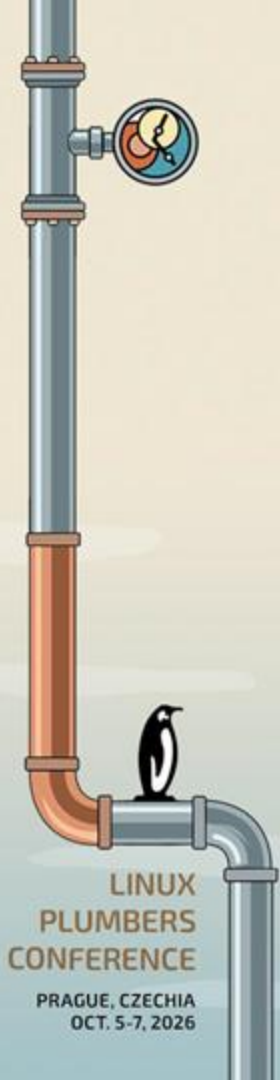
What Signal Are We Missing?

Existing KCOV -fsanitize-coverage=trace-pc can tell us:
"foo() was reached." also has trace-cmp, which observes comparison operands.

But sometimes we also need:

"foo() was reached with a value that has not been observed there before."





LINUX
PLUMBERS
CONFERENCE
PRAGUE, CZECHIA
OCT. 5-7, 2026

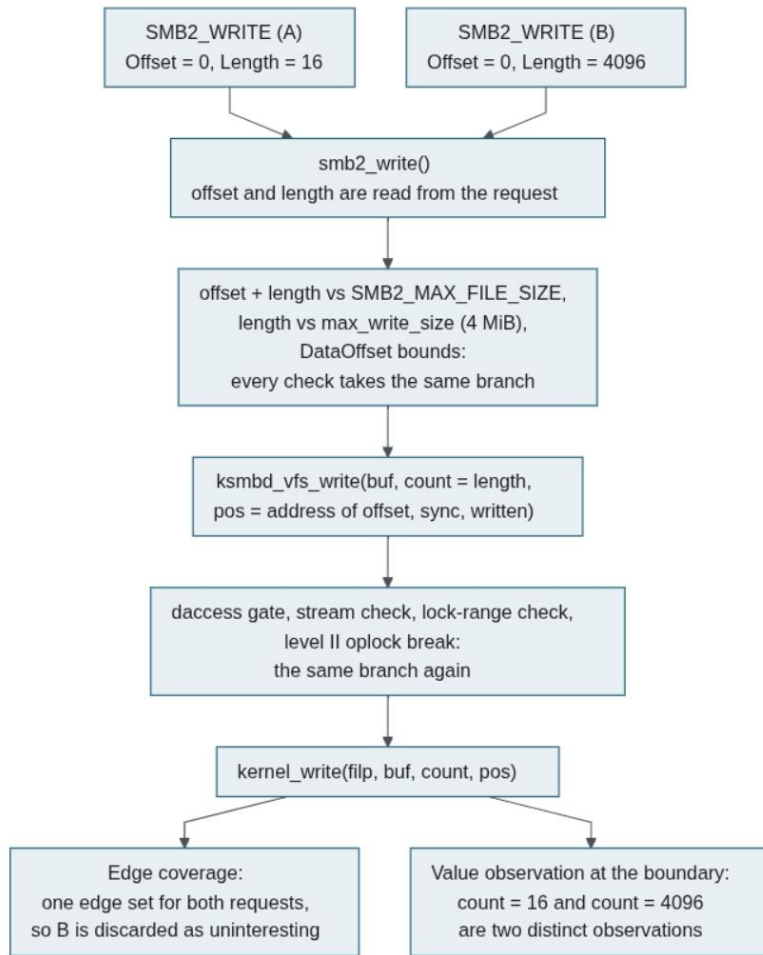


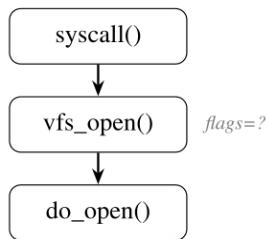
Diagram 1.1. The same control flow, twice. The length is never compared against anything that would separate 16 from 4096; it is bounds-checked once and then handed down as the `count` argument, and what it actually decides happens below `kernel_write()` — in the page cache, the allocator and the filesystem's own write path. `fs/smb/server/smb2pdu.c`, `fs/smb/server/vfs.c`

My RFC proposal

Rather than reconstructing ABI state at runtime, the compiler explicitly materializes argument values at compile time with additional signal:

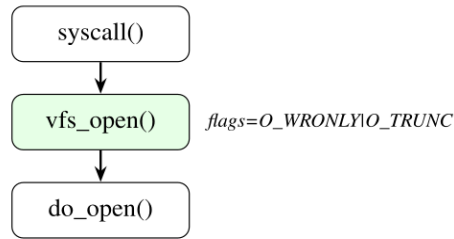
1. Function frame start → argument values
2. Function return → returned values

Edge Coverage (KCOV)



Sees: $A \rightarrow B \rightarrow C$

+ Data-Flow Extraction



Sees: $A \rightarrow B(flags=0x241) \rightarrow C$

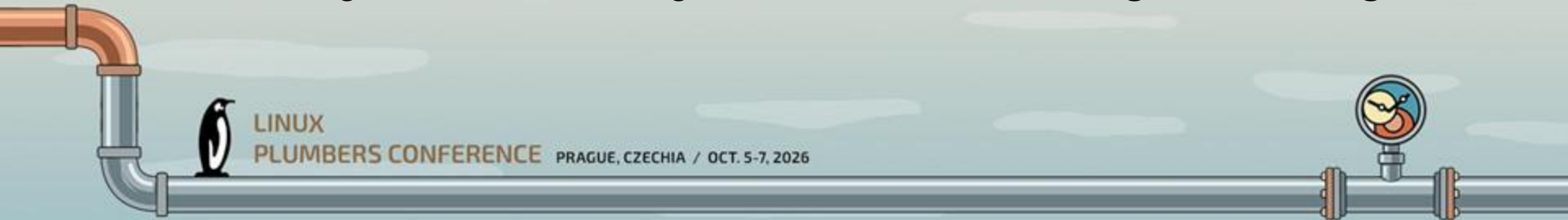
Figure 2: Edge coverage reports identical paths for different argument values. Data-flow extraction reveals the security-relevant difference.

Today's goal

Define the right upstream boundaries, not approve a finished this works.

I do not attempt to answer on this session, but the discussion is always welcome after this session:

1. How is each value produced?
e.g., tracking data provenance through a data-dependency graph
2. How can we feed vulnerability-hunting training to LLMs?
3. Coverage consumer tooling for `-fsanitize-coverage=trace-args/ret`



Worthwhile for New Discoveries

- [CVE-2026-74522](#) (CVSS3.1 score: 8.8): ksmbd: fix use-after-free in __close_file_table_ids() [e7188199eff4](#)
- [CVE-2026-90174](#) (CVSS3.1 score: 7.1): ksmbd: fix slab-out-of-bounds read in ksmbd_alloc_user() [7405d0ba2943](#)
- [CVE-2026-90173](#) (CVSS3.1 score: 9.8): smb: smbdirect: free completion queues with ib_free_cq() [fe2c0cacbcff](#)
- [CVE-2026-90172](#) (CVSS3.1 score: 7.5): smb: smbdirect: destroy QP before mem pools on accept failure [383a9480f5f4](#)
- [CVE-2026-90171](#): smb: smbdirect: release pending child sockets outside the handler lock [db82fbe4bb68](#)
- [CVE-2026-90170](#): ksmbd: validate ipc response length before dereferencing its fields [e9b33376bd07](#)
- [CVE-2026-90167](#): ksmbd: serialize oplock close with pending break ownership [b0148dc5625d](#)



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



End-to-End Prototype

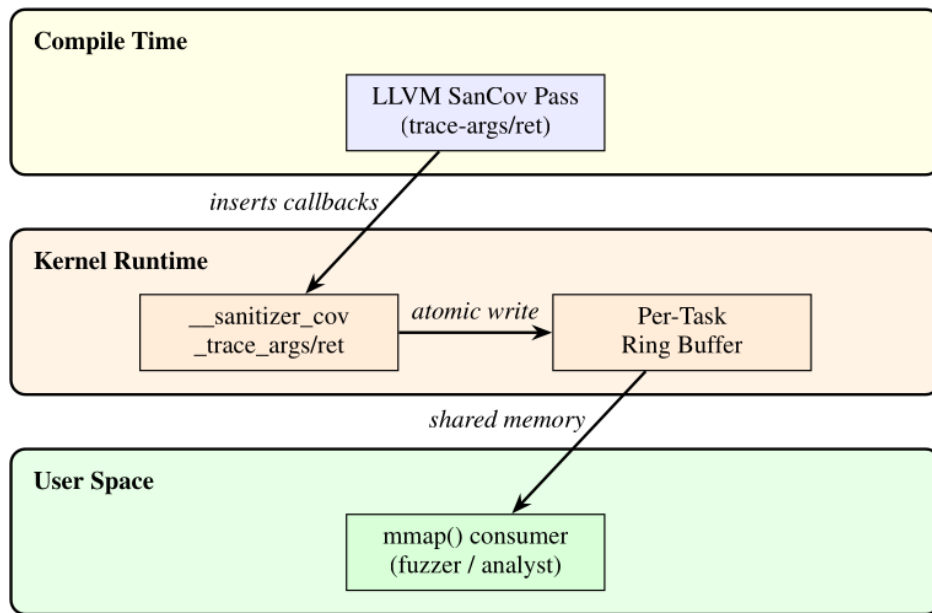


Figure 4: Three-layer architecture: compile-time instrumentation, kernel-resident lock-free buffer, and user-space consumption via `mmap()`.

End-to-End Prototype

The prototype connects compiler instrumentation to two runtime consumers.

```
do_sys_openat2(0xffffffff9c, "/etc/shadow", 0x241)
  build_open_flags( O_WRONLY|O_TRUNC )
  0x0 = build_open_flags()
  getname_flags( "/etc/shadow", 0x0 )
  0xffff8880... = getname_flags()
  vfs_open(path, {.flags=0x241, .mode=0x1b6} )
    do_dentry_open(file, inode, 0x0)
      security_file_open( file, 0x241 )
        0x0 = security_file_open()
        0x0 = do_dentry_open()
      0x4 = vfs_open()
    0x4 = do_sys_openat2()
```

*Captured
per-task
at runtime*

Figure 3: Actual KCOV-DATFLOW output for `open("/etc/shadow", O_WRONLY|O_TRUNC)`. Each line is a captured record showing function arguments (ENTRY) and return values (RET) with automatic struct field expansion. Indentation reflects call depth.

```
unsigned int ksmbd_open_durable_fd(struct ksmbd_file *fp)
{
    __open_id(&global_ft, fp, OPEN_ID_TYPE_PERSISTENT_ID);
    return fp->persistent_id;
}
```

```
000000000009fd0 <ksmbd_open_durable_fd>:
{
    9fd0: f3 0f 1e fa      endbr64
    9fd4: e8 00 00 00 00   call    9fd9 <ksmbd_open_durable_fd+0x9>
               9fd5: R_X86_64_PLT32 __fentry__ -0x4
    9fd9: 53              push    %rbx
    9fda: 48 89 fb        mov     %rdi,%rbx
    9fdd: 31 ff          xor     %edi,%edi
    9fdf: be e8 02 00 00   mov     $0x2e8,%esi
    9fe4: 48 89 da        mov     %rbx,%rdx
    9fe7: 48 c7 c1 00 00 00 00 mov     $0x0,%rcx
               9fea: R_X86_64_325 .rodata+0x14e0
    9fee: 41 b8 37 00 00 00 mov     $0x37,%r8d
    9ff4: e8 00 00 00 00   call    9ff9 <ksmbd_open_durable_fd+0x29>
               9ff5: R_X86_64_PLT32 sanitizer_cov_trace_args-0x4
    9ff9: e8 00 00 00 00   call    9ffe <ksmbd_open_durable_fd+0x2e>
               9ffa: R_X86_64_PLT32 sanitizer_cov_trace_pc-0x4
               fp, OPEN_ID_TYPE_PERSISTENT_ID);
    00 00 00 mov     $0x0,%rdi
    R_X86_64_325 .bss+0x40
    mov     %rbx,%rsi
    mov     $0x1,%edx
    call    a070 <__open_id>
    id;
    add     $0x8,%rbx
    mov     %rbx,%rax
    shr     $0x3,%rax
    movabs  $0xdffffc0000000000,%rcx
    a024: fc ff df      cmpb    $0x0,(%rax,%rcx,1)
    a027: 80 3c 08 00    je      a035 <ksmbd_open_durable_fd+0x65>
    a02b: 74 08          je
    a02d: 48 89 df        mov     %rbx,%rdi
    a030: e8 00 00 00 00   call    a035 <ksmbd_open_durable_fd+0x65>
               a031: R_X86_64_PLT32 asan_report_load8_noabort-0x4
    a035: 8b 1b          mov     (%rbx),%ebx
    a037: bf 04 00 00 00   mov     $0x4,%edi
    a03c: 48 89 de        mov     %rbx,%rsi
    a03f: 31 d2          xor     %edx,%edx
    a041: 31 c9          xor     %ecx,%ecx
    a043: e8 00 00 00 00   call    a048 <ksmbd_open_durable_fd+0x78>
               a044: R_X86_64_PLT32 sanitizer_cov_trace_ret-0x4
    a048: 89 d8          mov     %ebx,%eax
    a04a: 5b            pop     %rbx
    a04b: 2e e9 00 00 00 00 cs jmp   a051 <ksmbd_open_durable_fd+0x81>
               a04d: R_X86_64_PLT32 __x86_return_thunk-0x4
    a051: 66 66 66 66 66 66 66 66 data16 data16 data16 data16 data16 data16 cs nopw 0x0(%rax,%rax,1)
    a058: 0f 1f 84 00 00 00 00
    a05f: 00
```

\$ objdump -drS fs/smb/server/vfs_cache.o

End-to-End Prototype: Rust for Linux

```
eight_struct_args_rust.rs:253      0x20f = rstf_5+0x27({0x55, 0x11, 0x22, 0x33, 0x44}) [eight_struct_args_rust]
eight_struct_args_rust.rs:264      0x42f = rstf_6+0x2a({0x66, 0x11, 0x22, 0x33, 0x44, 0x55}) [eight_struct_args_rust]
eight_struct_args_rust.rs:276      0x86f = rstf_7+0x27({0x77, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66}) [eight_struct_args_rust]
eight_struct_args_rust.rs:288      0x10ef = rstf_8+0x2a({0x88, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77}) [eight_struct_args_rust]
eight_struct_args_rust.rs:467      0xaa = rsf_fwd_inner+0x33(0x11, {0x11, 0x22}, {0x11, 0x22, 0x33}, {0x11, 0x22, 0x33, 0x44}) [eight_struct_args_rust]
eight_struct_args_rust.rs:478      0xaa = rsf_fwd+0x33(0x11, {0x11, 0x22}, {0x11, 0x22, 0x33}, {0x11, 0x22, 0x33, 0x44}) [eight_struct_args_rust]
eight_struct_args_rust.rs:484      0x11, 0x11, 0x22, 0x33} @0xffffd0fbc12f3558 = rsf_ret_struct+0x31(0x11, {0x11, 0x22}) [eight_struct_args_rust]
```

[tools/testing/selftests/kcov_dataflow/eight_struct_args_rust](#)

```
rust_kworker_remote: 3466 words, 770 records, 6 from rust_kworker_remote (kaslr_offset=0x23a00000)
rust_kworker_remote.rs:58      <rust_kworker_remote[bcc8aecdd1c86427]::CompositeStore>::populate({0x0, 0x0, 0x0}) [rust_kworker_remote]
rust_kworker_remote.rs:73      <rust_kworker_remote[bcc8aecdd1c86427]::CompositeStore>::update({0xffff8f10d9938580, 0xffff8f10d98da4c0,
rust_kworker_remote.rs:92      <rust_kworker_remote[bcc8aecdd1c86427]::CompositeStore>::drain({0xffff8f10d9938580, 0xffff8f10d98da4c0, 0
```

[tools/testing/selftests/kcov_dataflow/rust_kworker_remote](#)



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



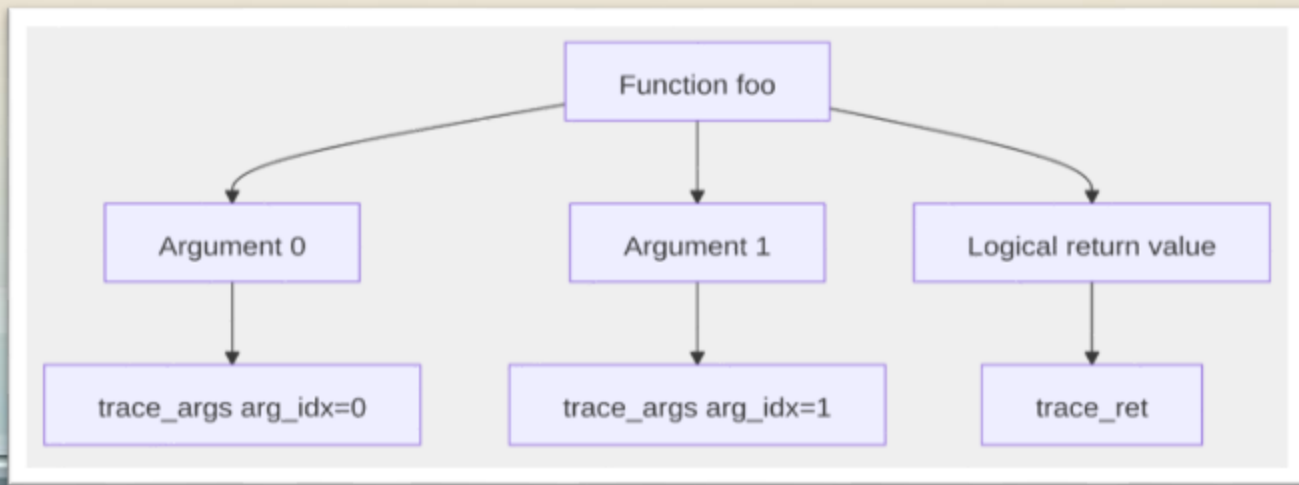
Current Callback ABI

Argument identity

A function can have multiple source parameters, so trace-args needs arg_idx.

Return identity

A source function has one logical return value, so trace-ret has no arg_idx.



Current Callback ABI: trace-args

1. `arg_idx`: Source parameter index
2. `size`: Observed value or object size
3. `val`: Direct value or object address
4. `*offsets`: Field offset
5. `num_fields`: Field count and val interpretation

```
void __sanitizer_cov_trace_args(  
    uint32_t arg_idx,  
    uint32_t size,  
    uint64_t val,  
    uint64_t *offsets,  
    uint32_t num_fields);
```

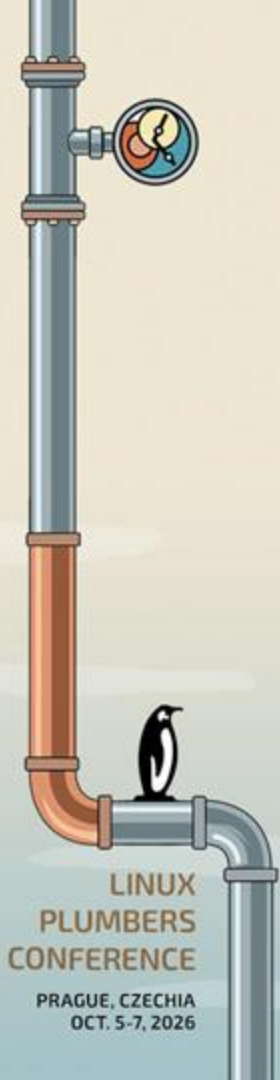


LINUX

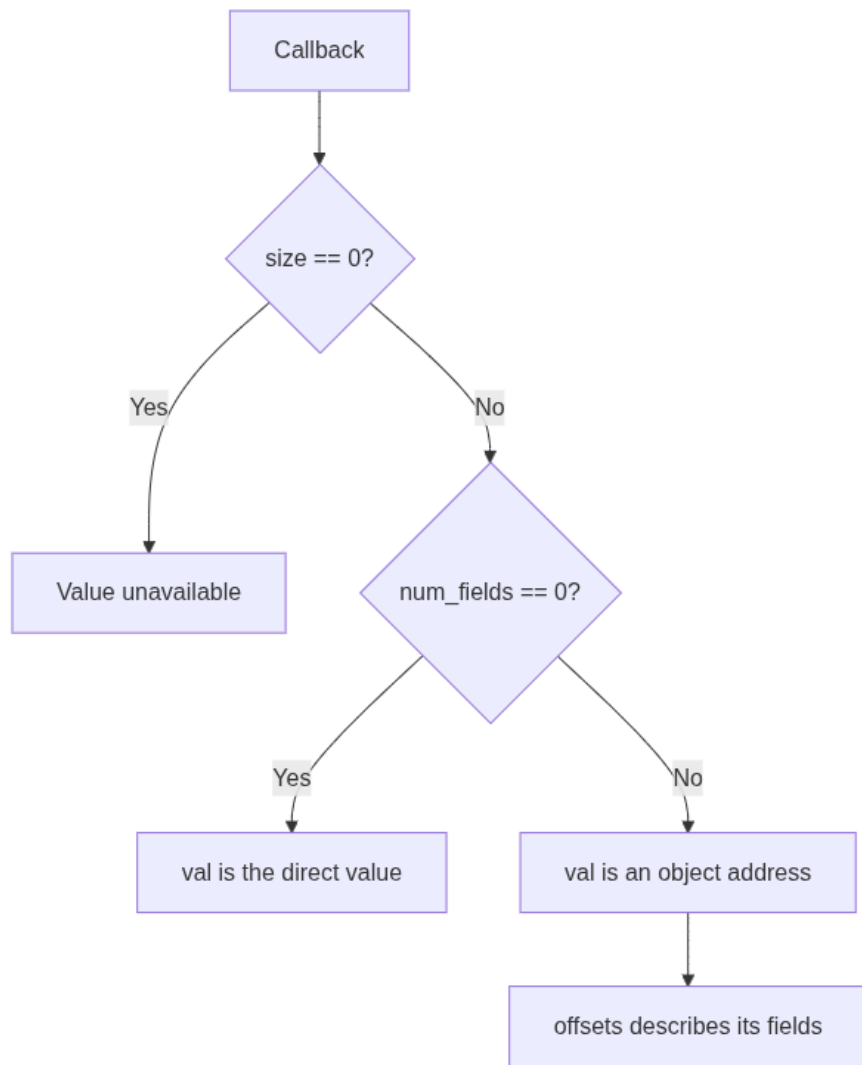
PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026





LINUX
PLUMBERS
CONFERENCE
PRAGUE, CZECHIA
OCT. 5-7, 2026



```
struct S { int a; long b; };          /* layout known: {0,4} {8,8} */  
  
void submit(struct S *s, int flags, void *token); /* void return */
```

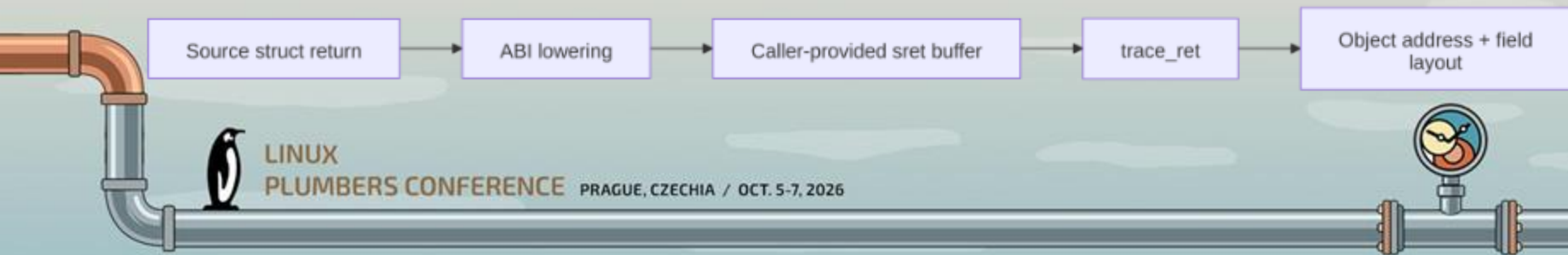
```
@__sancov_offsets_ = private unnamed_addr constant [4 x i64]  
                    [i64 0, i64 4, i64 8, i64 8]
```

```
define void @submit(ptr %0, i32 %1, ptr %2) {  
    %4 = ptrtoint ptr %0 to i64  
    call void @__sanitizer_cov_trace_args(i32 0, i32 16, i64 %4,  
                                           ptr @__sancov_offsets_, i32 2)  
  
    %5 = zext i32 %1 to i64  
    call void @__sanitizer_cov_trace_args(i32 1, i32 4, i64 %5, ptr null, i32 0)  
    %6 = ptrtoint ptr %2 to i64  
    call void @__sanitizer_cov_trace_args(i32 2, i32 8, i64 %6, ptr null, i32 0)  
    ...  
    call void @__sanitizer_cov_trace_ret(i32 0, i64 0, ptr null, i32 0)  
    ret void  
}
```

Current Callback ABI: trace-ret

1. size: Observed value or object size
2. val: Direct value or object address
3. *offsets: Field offset
4. num_fields: Field count and val interpretation

```
void __sanitizer_cov_trace_ret(  
    uint32_t size,  
    uint64_t val,  
    uint64_t *offsets,  
    uint32_t num_fields);
```



```
struct Big { long x; long y; long z; }; /* 24 bytes: too large for registers */
```

```
struct S *lookup(struct S *tab, int id); /* struct pointer */
```

```
@__sancov_offsets_.5 = private unnamed_addr constant [6 x i64]  
    [i64 0, i64 8, i64 8, i64 8, i64 16, i64 8]
```

```
define ptr @lookup(ptr %0, i32 %1) {  
    %3 = ptrtoint ptr %0 to i64  
    call void @__sanitizer_cov_trace_args(i32 0, i32 16, i64 %3,  
                                           ptr @__sancov_offsets_, i32 2)  
  
    %4 = zext i32 %1 to i64  
    call void @__sanitizer_cov_trace_args(i32 1, i32 4, i64 %4, ptr null, i32 0)  
    ...  
    %7 = ptrtoint ptr %6 to i64  
    call void @__sanitizer_cov_trace_ret(i32 16, i64 %7,  
                                          ptr @__sancov_offsets_, i32 2)  
  
    ret ptr %6  
}
```

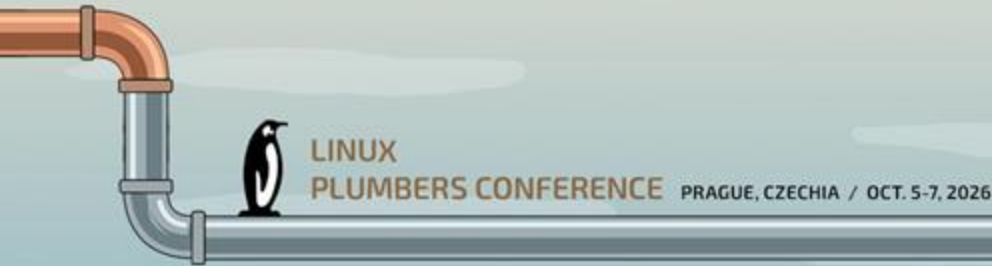
Debug Information Defines the Source View

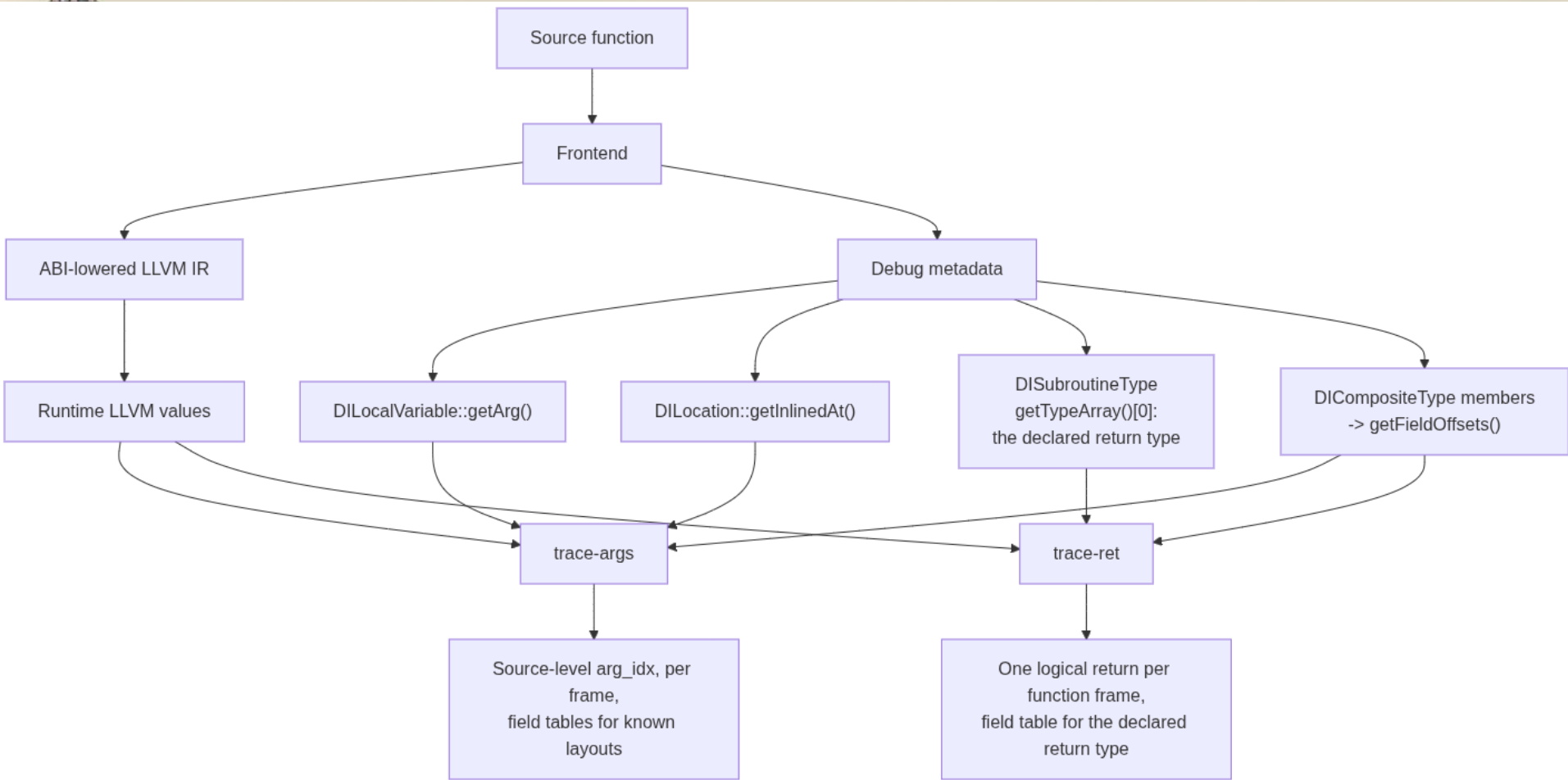
With `DEBUG_INFO`:

Source-level argument mapping currently uses debug records.

Without `DEBUG_INFO`:

Field tables generally unavailable. trace-args/trace-ret still observe concrete LLVM IR values without manufacturing memory.





What We Need to Agree On

New KCOV mode:

Add the struct `kcov_dataflow` infrastructure to struct `task_struct` and introduce `/sys/kernel/debug/kcov_dataflow` debugfs interface.

Callback representation:

Bytes size of callback ABI

Should `num_fields` serve as both field count and value/address Discriminator?



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



The background features a light blue and yellow gradient sky with soft, white clouds. Below the sky is a faint, stylized line-art illustration of a city skyline. On the left side, a large, vertical pipe with a blue body and orange-brown flanges extends from the top to the bottom. The pipe has a 90-degree elbow at the top. The city skyline includes several buildings of varying heights, some with multiple windows, and a prominent church with two tall, pointed spires. The foreground is a solid light blue color.

Thank you



LINUX
PLUMBERS
CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026

Appendix

1.

Repo: <https://github.com/yskzalloc/kcov-dataflow/>

2. CI: <https://github.com/yskzalloc/kcov-dataflow/actions>

3. RFC v3:

<https://lore.kernel.org/all/20260902-b4-kcov-dataflow-rfc-v3-v3-0-824609f279f5@est.tech/>

4. Paper: <https://arxiv.org/pdf/2606.00455>

5. LLVM RFC:

<https://discourse.llvm.org/t/rfc-sanitizercoverage-add-fsanitize-coverage-trace-args-trace-ret/91026>

6. Implementation Notes:

<https://vivliostyle.org/viewer/#src=https://trace-args-ret.github.io>



LINUX

PLUMBERS CONFERENCE

PRAGUE, CZECHIA / OCT. 5-7, 2026



Appendix: LLVM stacked PR

1. [sancov] Add -fsanitize-coverage=trace-args,trace-ret <https://github.com/llvm/llvm-project/pull/201410>
2. [Clang][SanitizerCoverage] Add trace-args and trace-ret instrumentation modes <https://github.com/llvm/llvm-project/pull/218254>
3. [compiler-rt][SanitizerCoverage] Add userspace runtime for trace-args/ret <https://github.com/llvm/llvm-project/pull/218265>
4. [compiler-rt][SanitizerCoverage] Add LibFuzzer support for trace-args/trace-ret <https://github.com/llvm/llvm-project/pull/227612>
5. [Clang][SanitizerCoverage] Document trace-args and trace-ret <https://github.com/llvm/llvm-project/pull/227614>

