

EXTENDING BTF TO SUPPORT MORE LANGUAGES

Alan Maguire, Arnaldo Carvalho de
Melo

Linux Plumbers Conference 2026
eBPF Track



WHY NOW?

- Kernel modules can be built in **Rust**
- **BTF** was designed around C
- Time to assess non-C constructs
- Rough suggestions to trigger discussion

TWO KINDS OF BTF

- Program BTF: from compiler/linker, for the verifier
- Module BTF: from pahole via DWARF, for tracing
- Rust BPF → C module: works today (Aya)
- Rust BPF → Rust module: no module BTF yet
- C BPF → Rust module: no module BTF yet
- kfuncs written in Rust: need module BTF too

THE PAHOLE ENCODING GAP

- Currently pahole skips generating BTF for Rust modules
- Want them to be first class citizens for tracing etc
- Find common ground with other projects wrt BTF encoding

ALSO: BPF PROGRAMS IN RUST

```
Option<alloc::collections::btree::map::BTreeMap<u32, u32>>
```

- [Alexei Starovoitov: rust-bpf series](#), bpf-next
- BTF: Rust names, unnamed args, small DATASEC vars
- scx_simple, scx_cosmos in Rust: BTF loads
- Modules too: SEC("fentry/rust_print/fmt<str>")

AYA

- Aya developed bpf-linker which helps rustc generates BTF from LLVM DWARF
- Uses existing BTF kinds
- BTF CO-RE support in progress
- BTF experimental: names made C, data enums dropped
- repr(C) where data leaves a function: maps, globals
- How far can we get using the same sort of approach for Rust-written kernel modules?

BTF REPRESENTATION PRIORITIES

- C: BTF → header → the same C, nearly lossless
- BPF needs the internal layout, for the verifier
- Rust via DWARF: we see the layout, not the intent
- You can't tell which layout is a guaranteed contract:
- `repr(C)` vs `repr(Rust)`
- Full circle back to Rust: limited

REPR(RUST), REPR(C) AND CO-RE

- `#[repr(Rust)]`: rustc picks the layout and order
- Two compilers may lay it out differently
- DWARF has the layout, never the `#[repr]`
- So BTF cannot recover it: CO-RE is vital
- Dump as `#[repr(C)] + preserve_access_index`

C: SIMPLE

```
struct point { int x; int y; };  
enum color { RED, GREEN, BLUE };
```

- Struct → BTF_KIND_STRUCT
- Enum → BTF_KIND_ENUM (plain integers)

RUST: RICH

```
enum Option<T> { Some(T), None }  
enum Result<T, E> { Ok(T), Err(E) }
```

- Enums with data: tagged unions
- Discriminants, generics, zero-sized types
- 128-bit discriminant values

DWARF: C ENUM

```
DW_TAG_enumeration_type "color"  
  DW_TAG_enumerator "RED" (0)  
  DW_TAG_enumerator "GREEN" (1)  
  DW_TAG_enumerator "BLUE" (2)
```

DWARF: RUST ENUM

```
DW_TAG_structure_type "Option<i32>"  
  DW_TAG_variant_part (DW_AT_discr -> discriminant)  
    DW_TAG_variant "Some" (DW_AT_discr_value: 1)  
      DW_TAG_member "0" (i32)  
    DW_TAG_variant "None" (DW_AT_discr_value: 0)
```

- New tags for pahole: `variant_part`, `variant`, `discr_value`
- Unused in C: no support until Rust came along

LLVM ALREADY ENCODES IT

```
[1] STRUCT 'MyEnum' size=12 vlen=1
    '(anon)' type_id=3 bits_offset=0
[3] UNION '(anon)' size=12 vlen=3
    '(anon)' type_id=2 bits_offset=0 <- discriminant
    'First' type_id=4 bits_offset=0
    'Second' type_id=6 bits_offset=0
```

- STRUCT wrapper + single anonymous UNION
- Discriminant as anonymous member + variants
- pahole follows this shape

DON'T INFER: DW_AT_DISCR

- rustc layout explicitly unstable
- DWARF 5, 5.7.10: DW_AT_discr points at the member
- Real type, real location, real size
- Synthesize name "__discriminant" (BTF needs names)

PRETTY TARGET

```
struct Option<u32> {  
    u32 __discriminant;      /* 0 4 */  
    union {  
        struct None {};  
        struct Some { u32 __0; /* 4 4 */ };  
    };  
};
```

ZERO-SIZED, DYNAMICALLY-SIZED

- **ZSTs**: byte_size 0, member type () rewrites to 4 bytes
- STRUCT size=0 with a 4-byte member: BRAIN FART ALERT!
- **DSTs**: str/[u8] tails, member bytes > struct size
- Fix: skip zero-size members, clamp padding math

NON-C PRECEDENT: GO, 2023

- Trace Summit Bilbao: uprobes + BPF on Go userspace
- Go DWARF, calling convention, [perf probe](#) for Go
- We've done non-C DWARF before

COMMON NEED ACROSS LANGUAGES - NAMESPACES?

- Namespacing is used extensively in Rust
- C hierarchy is flat, most languages have deep scope/nesting relations
- Can simulate with BTF tags, but it is wasteful; need one tag per tagged entity
- A more structured BTF kind could represent parent-child relationships
- Namespace as a whole could be tagged if needed
- First step: `rust:kind=<kind> decl tags`

RUST AND THE VERIFIER

```
__bpf_kfunc void clear(void *bytes, u32 bytes__sz);  
fn clear(bytes: &mut [u8]);
```

- C kfuncs: pointer + size by the __sz convention
- Rust: the slice carries both
- Enums as kfunc arguments: discriminant + payload
- BTF lets unsafe Rust in too: verifier support?

ASK TO THE ROOM

- Named `__discriminant` already usable in CO-RE relocations
- `discr_values` per variant not representable in BTF today
- New kind? DECL_TAG-like annotations on union members?
- Verifier tracking the active variant: match at load time

FUNCTION TRACING IN RUST MODULES

- Problem: ELF symbols, kallsyms for Rust module functions uses mangled Rust names
- Example kallsyms entry for rust_print module:

```
fffffffffc0fec430 t _RNvXs5_NtCs1DErgtJhccC_6kernel3fmtReNtB5_7Display3fmtCs  
7y3Drb9gYub_15rust_print_main [rust_print]
```

SOLUTION: DECL TAGS FOR MANGLED NAMES

- Have pahole use the DW_at_name, record the DW_at_linkage_name as a BTF decl tag:

```
[145526] FUNC 'fmt<str>' type_id=145525 linkage=static  
[145527] DECL_TAG 'rust:linkage:_RNvXs5_NtCs1DErgtJhccC_6kernel3fmtReNtB5_7  
Display3fmtCs7y3Drb9gYub_15rust_print_main' type_id=145526 component_idx=-1
```

USING DECL TAGS TO TRACE

- Allows for tracing a function using a stable name

```
SEC("fentry/rust_print/fmt<str>")
```

- Still supports runtime matching:
 - Find function via name
 - Use decl tag to identify attach point in kallsyms
- See [here](#) for a PoC

EXPERIMENTING WITH MODULE BTF GENERATION IN RUST

- remove lang-exclude for Rust CUs
- make LLVM=1
- With latest pahole v1.32, it works!
- We see many duplicate types in vmlinux BTF due to bindgen differences (bpf_link, sock)

C TYPE

```
struct leaf_0x2_table {
    union {
        enum _cache_table_type c_type;
        enum _tlb_table_type t_type;
    };
    /* offset 0, size 1 */
    /* 1 byte of padding */
    union {
        short c_size;
        short entries;
    };
    /* offset 2, size 2 */
};
```

RUST TYPE

```
struct leaf_0x2_table_bindgen {  
    union {  
        uint8_t c_type;  
        uint8_t t_type;  
    } __bindgen_anon_1;           /* offset 0, size 1 */  
    /* 1 byte of padding */  
    union {  
        int16_t c_size;  
        int16_t entries;  
    } __bindgen_anon_2;          /* offset 2, size 2 */  
};
```

AND ON REBOOT?

- vmlinux BTF load fails due to long names
- If we increase name limit to 2048 bytes, all works!
- Can load rust samples with BTF

THE END

- <https://github.com/acmel/dwarves>
- Test your Rust programs with pahole, file issues!



[fedorapeople.org/~acme/prez/
lpc-2026-btf-languages/](https://fedorapeople.org/~acme/prez/lpc-2026-btf-languages/)

OPEN: THE ABSTRACT'S LIST

- Slices / "fat pointers" with type and size
- Generics (monomorphization explosion)
- Interfaces (trait objects, vtables)
- Lifetime tags
- See: [btf-for-rust-proposal.md](#)

OPEN QUESTIONS

- `VARIANT_PART`: implicit fields or explicit `vlen=N`?
- Discriminant: mandatory? Counted in the size?
- Vtables: is method order consistent enough?
- Rollout: emit old and new? Ignore unknown kinds?
- 128-bit discriminants: straight to `btf_enum128`?

EXPLICIT VS NICHE

- Option<u32>: tag at 0, payload at 4 → emit it
- Option<NonZero<u32>>: tag IS the payload → emit nothing
- Rule: emit only when it doesn't overlap the payload
- Niche is the common kernel case (Result<(), Error>)

UNIONS SIT AT ZERO

- Non-zero discriminants can't live in the UNION
- `libbpf` refuses, kernel `verifier` rejects at load
- Real storage → outer STRUCT (overlap legal)
- Payload itself → dropped
- LLVM emits what the verifier rejects: upstream bug

COMPILERS EMIT BTF TOO

- GCC `-gbtf` (12+, default on BPF) + `-mco-re`
- LLVM `-gbtf` + `lld --btf-merge`: pahole step optional, someday
- `gccrs`: still out of tree