RUST IN THE KERNEL (VIA EBPF)

Writing eBPF programs in Rust with Aya

ABOUT ME

- Michal Rostecki
- **vadorovsky** @ Github, Discord, Mastodon etc.
- Software Engineer @ Deepfence Inc
- Rustacean with Go, C and Python background

AYA

- eBPF programs in pure Rust.
- Rust toolchain is the only dependency.
- Async support for the user-space part.
- Focus on developer experience.

WHAT'S MISSING?

- Emitting BTF debug info.
  - Currently being worked on, experimental PoC ready.
  - Needed for BTF maps and verifier debug logs.
- BTF relocations are supported by the loader, but not the compiler - long-term future plan.
  - Needed for CO-RE.

LEGACY BPF MAPS (IN C)

```
struct bpf_map_def SEC("maps") flow_map = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(unsigned int),
    .value_size = sizeof(unsigned int),
    .max_entries = 1,
};
```

Map parameters as values.

BTF MAPS

IN C

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __type(key, unsigned int);
    __type(value, unsigned int);
    __uint(max_entries, 1024);
} pid_map SEC(".maps");
```

Map parameters embedded in the struct layout.

WHICH GETS TRANSFORMED TO

```
struct {
    int (*type)[BPF_MAP_TYPE_HASH];
    typeof(unsigned int) *key;
    typeof(unsigned int) *value;
    int (*max_entries)[1024];
} pid_map SEC(".maps");
```

IN RUST

```
pub struct HashMap<K, V, const M: usize, const F: usize = 0> {
    r#type: *const [i32; BPF_MAP_TYPE_HASH as usize],
    key: *const K,
    value: *const V,
    max_entries: *const [i32; M],
    map_flags: *const [i32; F],
}
```

USAGE

```rust
use aya_btf_map::{macros::btf_map, HashMap};

#[btf_map]
static mut PID_MAP: HashMap<i32, i32, 1024> = HashMap::new();
```

```rust
fn try_fork(ctx: TracePointContext) -> Result<u32, c_long> {
    const PARENT_PID_OFFSET: usize = 24;
    const CHILD_PID_OFFSET: usize = 44;
    let parent_pid: i32 = unsafe {
        ctx.read_at(PARENT_PID_OFFSET)? };
    let child_pid: i32 = unsafe {
        ctx.read_at(CHILD_PID_OFFSET)? };

    unsafe { PID_MAP.insert(&parent_pid, &child_pid, 0)? };

    Ok(0)
}
```

USER-SPACE CODE DOESN'T CHANGE!

```
use aya::maps::HashMap;

let pid_map: HashMap<_, u32, u32> =
    HashMap::try_from(bpf.map_mut("PID_MAP").unwrap())?;
[...]
for r in pid_map.iter() {
    let (k, v) = r?;
    info!("parent: {k}, child: {v}");
}
```

PROBLEMS

- Kernel expects:
  - BTF maps as anonymous structs.
  - Anonymous pointer types.
  - Only C types, Rust supports more complex types (e.g. data carrying enums).

DEBUG INFO (FROM C)

```
!19 = !DIDerivedType(tag: DW_TAG_pointer_type,
    baseType: !20, size: 64)
```

```
!38 = distinct !DICompositeType(tag: DW_TAG_structure_type,
    file: !3, line: 6, size: 256, elements: !39)
```

DEBUG INFO (FROM RUST)

```
!8 = !DIDerivedType(tag: DW_TAG_pointer_type,
    name: "*const [i32; 1]", baseType: !9, size: 64,
    align: 64, dwarfAddressSpace: 0)
```

```
!4 = !DICompositeType(tag: DW_TAG_structure_type,
    name: "HashMap", scope: !2, file: !5, size: 320,
    align: 64, elements: !6, templateParams: !27,
    identifier: "9622a717fe95ebf2fc2a9847a6ff0bef")
```

BTF (FROM C)

```
[9] STRUCT '(anon)' size=32 vlen=4
        'type' type_id=1 bits_offset=0
        'key' type_id=5 bits_offset=64
        'value' type_id=5 bits_offset=128
        'max_entries' type_id=7 bits_offset=192
```

```
[5] PTR '(anon)' type_id=6
```

BTF (FROM RUST)

```
[11] STRUCT 'HashMap' size=40 vlen=5
        'type' type_id=1 bits_offset=0
        'key' type_id=5 bits_offset=64
        'value' type_id=5 bits_offset=128
        'max_entries' type_id=7 bits_offset=192
        'map_flags' type_id=9 bits_offset=256
```

```
[5] PTR '*const u32' type_id=6
```

SOLUTIONS

- Temporary: modify DI in bpf-linker.
- Long-term: `#[btf_export]` macro in Rust.

BPF-LINKER

Currently working PoC. Transforms DI to meet kernel expectations:

- Removes names from pointer types and BTF map structs.
- Tweaks the DI of Rust-specific types to be C-compatible.

BTF (FROM RUST) AFTER MODIFICATIONS

```
[10] STRUCT '(anon)' size=40 vlen=5
        'type' type_id=1 bits_offset=0
        'key' type_id=5 bits_offset=64
        'value' type_id=5 bits_offset=128
        'max_entries' type_id=6 bits_offset=192
        'map_flags' type_id=8 bits_offset=256
```

```
[6] PTR '(anon)' type_id=7
```

DEBUG INFO INCLUDED

```
; let parent_pid: i32 = unsafe {
    ctx.read_at(PARENT_PID_OFFSET)? };
9: 63 1a f4 ff 00 00 00 00 *(u32 *)(r10 - 0xc) = r1
[...]
; let child_pid: i32 = unsafe {
    ctx.read_at(CHILD_PID_OFFSET)? };
19: 63 1a f8 ff 00 00 00 00 *(u32 *)(r10 - 0x8) = r1
20: bf a2 00 00 00 00 00 00 r2 = r10
```

LLVM WORKING ITEMS

- BTF backend segfaults with some DI produced by Rust (e.g. data carrying enums).
- `LLVMGetDINodeTag` function to get the tag of DI Node.
- `LLVMReplaceMDNodeOperandWith` function to modify DI.

#[BTF_EXPORT]

- Decoupled from `-C debuginfo`.
- Generates DI, which produces correct BTF for annotated types.
- Raises a compiler error when used on BTF-incompatible type.

IF YOU WANT TO TRY IT OUT

- github.com/vadorovsky/aya-btf-map - structs and macros for BTF maps.
- github.com/vadorovsky/aya-btf-maps-experiments - example project using it.
- Requires LLVM and bpf-linker patches.

THANK YOU

- aya-rs.dev
- github.com/aya-rs/aya
- Discord