Aya

What's new in Rust for eBPF?

Michal Rostecki (vad)

Anza

January 31, 2026

What is Aya?

Aya is an eBPF library allowing to write both user-space and eBPF components in Rust.

But why Rust?

Isn't eBPF memory-safe?
There are more benefits:

- Quick scaffolding of the project with cargo-generate.
- Ability to use dependencies from crates.io.
- Ability to publish on crates.io.
- `Option` and `Result`, pattern matching.

aya-template

Thanks to cargo-generate, you can start with Aya in a less than one minute.

```
cargo generate https://github.com/aya-rs/aya-template
```

crates.io

Any crate can be used as long as it's no_std.

network-types

A good example of a crate usable in Aya. It provides definitions of different L2/L3 network protocols.

```
use network_types::{
    eth::{EthHdr, EtherType},
    ip::{Ipv4Hdr, Ipv6Hdr, IpProto},
    tcp::TcpHdr,
    udp::UdpHdr,
};
```

What changed since last time?

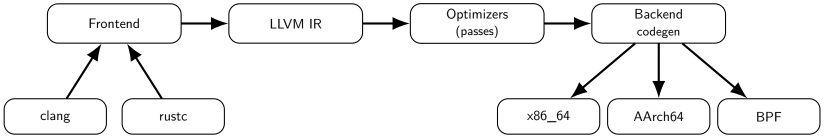
- Integration test framework. Testing on ARM.
- Logging ported from perf buffers to ring buffers.
- More map types, e.g. `sk_storage`.
- More program types, e.g. flow dissector, iterator.
- Support for entirety of Rust types in BTF.
- TCX links.

BTF

BPF Type Format (BTF) is a debug info format, way smaller than DWARF. It's used to perform relocations of kernel types.
It's produced either by:

- A compiler that builds a BPF program.
- pahole, that converts DWARF from the Linux kernel to BTF.

LLVM


Frontend
LLVM IR
Optimizers
(passes)
Backend
codegen
clang
rustc
x86_64
AArch64
BPF

Simple Rust code

```rust
pub struct Foo {
    a: i32,
    b: u32,
}
```

LLVM Debug Info

```
!87 = !DIBasicType(name: "u32", size: 32, encoding: DW_ATE_unsigned)
[...]
!152 = !DICompositeType(tag: DW_TAG_structure_type, name: "Foo", scope:
↪  !9, file: !14, size: 64, align: 32, flags: DIFlagPublic, elements:
↪  !153, templateParams: !74, identifier:
↪  "bce6452370c655d63a49cfb6bdce4bc7")
!153 = !{!154, !156}
!154 = !DIDerivedType(tag: DW_TAG_member, name: "a", scope: !152, file:
↪  !14, baseType: !155, size: 32, align: 32, flags: DIFlagPrivate)
!155 = !DIBasicType(name: "i32", size: 32, encoding: DW_ATE_signed)
!156 = !DIDerivedType(tag: DW_TAG_member, name: "b", scope: !152, file:
↪  !14, baseType: !87, size: 32, align: 32, offset: 32, flags:
↪  DIFlagPrivate)
```

BTF

```
#2: <STRUCT> 'Foo' sz:8 n:2
        #00 'a' off:0 --> [3]
        #01 'b' off:32 --> [4]
#3: <INT> 'i32' bits:32 off:0 enc:signed
#4: <INT> 'u32' bits:32 off:0
```

BTF relocations

BTF relocations live inside the `.BTF.ext` ELF header. They are produced by compilers.
LLVM produces them using the following intrinsics:

- `llvm.preserve.struct.access.index.*`
- `llvm.preserve.union.access.index.*`

Language frontends should wrap the `GetElementPtr` instructions, where developer requests the relocation.

BTF relocation marker in clang

```
struct task_struct {
    [...]
    pid_t pid;
    [...]
} __attribute__((preserve_access_index));

struct task_struct *task = bpf_get_current_task_btf();
return task->pid;
```

Corresponding BTF

```
#351: <STRUCT> 'task_struct'
[...]
        #96 'pid' off:21632 --> [1553]
```

Corresponding bytecode

```
r0 = *(u32 *)(r2 + 2704)
```

Relocation

```
RELO: FIELD_OFFSET
  type_id = task_struct
  access = pid
  insn_offset = 5
```

How BTF relocations are consumed?

eBPF loaders, like Aya or libbpf, have to perform the relocations themselves by reading the kernel's BTF and replacing the offsets mentioned by relocations in the bytecode.

Different BTFs

Program:

```
#351: <STRUCT> 'task_struct'
[...]
       #96 'pid' off:21632 --> [1553]
```

Kernel:

```
#351: <STRUCT> 'task_struct'
[...]
       #97 'pid' off:21664 --> [1553]
```

Patching offsets

Before:

```
r0 = *(u32 *)(r2 + 2704)
```

After:

```
r0 = *(u32 *)(r2 + 2708)
```

How we might do it in Rust?

```
#[repr(C)]
#[preserve_access_index]
struct task_struct {
    pid: i32,
}
```

BTF support in Aya

- **Done:** Rust is able to produce correct BTF for all types.
- **To do:** Rust is not (yet) able to produce BTF relocations.
- **Done:** Aya is able to load BTF from the program and perform the relocations against kernel's BTF.
- **Soon:** Support for BTF-compatible map definitions is coming in the next release.

Producing correct BTF for all types

Making production of BTF possible required two changes in BPF backend:

- Support of `DW_TAG_variant_part`.
- Stripping names of pointer types in Debug Info.

DW_TAG_variant_part / tagged union

A structure that has:

- A discriminant (`DW_AT_discr`).
- Possible variants of data (`DW_TAG_variant`).

Similar to a union, but with a discriminant that ensures the safety.

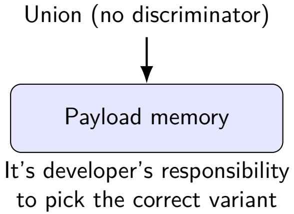
C union (not tagged)

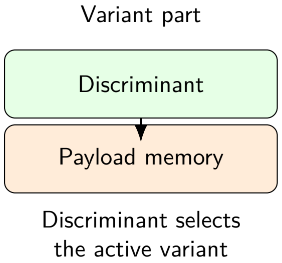
```
union {
    struct { uint32_t a; int32_t b; } ab;
    uint32_t u;
};
```

Rust enum

```rust
pub enum MyEnum {
    First { a: u32, b: i32 },
    Second(u32),
}
```

Union vs. variant part





Resulting BTF

```
[1] STRUCT 'MyEnum' size=12 vlen=1
        '(anon)' type_id=3 bits_offset=0
[2] INT 'u32' size=4 bits_offset=0 nr_bits=32 encoding=(none)
[3] UNION '(anon)' size=12 vlen=3
        '(anon)' type_id=2 bits_offset=0
        'First' type_id=4 bits_offset=0
        'Second' type_id=6 bits_offset=0
[4] STRUCT 'First' size=12 vlen=2
        'a' type_id=2 bits_offset=32
        'b' type_id=5 bits_offset=64
[5] INT 'i32' size=4 bits_offset=0 nr_bits=32 encoding=SIGNED
[6] STRUCT 'Second' size=12 vlen=1
        '__0' type_id=2 bits_offset=32
```

Future plans and ideas

- BTF relocations in Rust compiler.
- Allow linking with binutils.
- Struct ops and `sched_ext`.

Adoption

anza

EXEIN

kubernetes

Red Hat

Anza

Anza develops `agave-xdp`, a Rust crate for using XDP based on Aya.
We use XDP for fast packet retransmission in Agave, a Solana validator implementation.
`https://crates.io/crates/agave-xdp`

mitmproxy

Uses Aya in their WireGuard and Local Redirect modes.

`https://github.com/mitmproxy/mitmproxy_rs`

Thank you

Questions?