Improving the eBPF Developer Experience With Rust!

Dave Tucker
Alessandro Decina
About us
Dave Tucker

- Principal Software Engineer, Red Hat Office of the CTO
- A Go developer, learning Rust
- Networking & Containers (Docker)
Alessandro Decina

- Software Engineer, Deepfence
- Added eBPF support to Rust
- Started Aya
Developer experience
The eBPF Developer Path

1. Get hooked with perf and bpftrace one-liners

2. Identify > 1 line problem that could be solved with eBPF

3. Choose your own adventure:
   a. Use a DSL like bpftrace/systemtap
   b. Use C for the eBPF program and choose a userspace library
   c. Use a single language for both eBPF and userspace

"Choose Your Own Adventure" by Brett Kiger is licensed with CC BY-NC-ND 2.0. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/2.0/
Developer Experience

1. Documentation
   - Scattered. Evolving quickly
2. Project Bootstrap
   - Change an example from libbpf-bootstrap
3. Build and Test Loop
4. Debugging
   - Compile. Load. Manually Check Behaviour. Or, try BPF_PROG_TEST_RUN.
   - printk
Why RUST?
WHY RUST?

Rust is a highly expressive language, comes with a feature rich standard library and can still get as low level as C

Memory safety (userspace) is great. Powerful type system and macros make writing eBPF code easier.

Fantastic dev tools including rustup, cargo, rust-analyzer
ABOUT AYA

Aya is the first Rust native eBPF library. It provides:

- An userspace eBPF library (like libbpf), completely written in rust

- An high level rust API to write eBPF code - like bpftrace or the bcc DSL - but using plain rust
THE AYA EXPERIENCE
1. DOCUMENTATION

---

1. Introduction
2. eBPF Program Limitations
3. Getting Started
   3.1. Development Environment
   3.2. Hello XDP
   3.3. Logging Packets

---

**Introduction**

Welcome to Building eBPF Programs with Aya: An introductory book about Programming Language and Aya library to build extended Berkeley Packet!

**Who Aya Is For**

Rust is proving to be a popular systems programming language because of excellent C interoperability. The safety features are less important in the programs often need to read kernel memory, which is considered unsafe. Combined with Aya does offer is a fast and efficient development experience.

- Cargo for project scaffolding, build, test and debugging
- Generation of Rust bindings to Kernel Headers with Compile-Once, R support
- Easy code sharing between user-space and eBPF programs
- Fast compile times
- No runtime dependency on LLVM or BCC

**Scope**

The goals of this book are:

- Get developers up to speed with eBPF Rust development. i.e. How to environment.
- Share current best practices about using Rust for eBPF

---

https://aya-rs.github.io/book

---

https://docs.rs/aya/
2. Bootstrap

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

👨‍💻 Project Name: lpc2021
🔧 Generating template ...
❓ 🤔 Which type of eBPF program? 🔄

❯ kprobe
  - kretprobe
  - uprobe
  - uretprobe
  - sock_ops
  - sk_msg
  - xdp
  - classifier
  - cgroup_skb
  - probe
  - tracepoint
2. Bootstrap

This gives you a workspace with 3 packages:

- lpc2021 (userspace)
- lpc2021-common (code shared between eBPF and userspace)
- lpc2021-ebpf (eBPF code)
2. BOOTSTRAP

A task to generate bindings to kernel types can easily be added:

$ cargo xtask codegen

This uses aya-gen to create Rust bindings to using the BTF types in /sys/kernel/btf/vmlinux
3. Build and Test Loop

Build & Run:

$ cargo build

$ cargo xtask build-ebpf

$ sudo ./target/debug/myapp --path ./target/bpfel-unknown-none/debug/myapp

💡 The second step is required as we need nightly rust to compile eBPF and several unstable cargo features to support having a multi-target workspace. In time, this step will be removed
4. Debugging

Debugging eBPF programs can be hard. Common options include:

- `bpf_trace_printk()` - slow, hard to follow output with multiple programs

- ad hoc perf events to trace program flow and dump data - works but inconvenient
4. Debugging with aya-log

```rust
info!(ctx, "aya-log is a lightweight logging library for eBPF code");
warn!(ctx, "it sends logs to userspace as perf events");
debug!(ctx, "it supports string {}", "formatting");
trace!(ctx, "it integrates nicely with the standard rust log crate");
error!(ctx, "find it at https://github.com/aya-rs/aya-log");
```
4. Debugging with aya-log

07:17:40 [INFO] [src/main.rs:35] aya-log is a lightweight logging library for eBPF code

07:17:40 [WARN] [src/main.rs:36] it sends logs to userspace as perf events

07:17:40 [DEBUG] (4) [src/main.rs:37] it supports formatting

07:17:40 [TRACE] (4) [src/main.rs:38] it integrates nicely with the standard rust log crate

07:17:40 [ERROR] [src/main.rs:39] find it at https://github.com/aya-rs/aya-log
ROADMAP
UNIT TESTING

- We plan to add the ability for program contexts and maps to be mocked so code can be tested on the host architecture.

- This should speed up the build/test loop significantly.
LIBBBPF COMPATIBILITY

- Automated tests to ensure libbpf compatibility for implemented program types

- More program types! - LSM and more cgroup hooks are in progress
CRANELIFT

- A code-generator for WebAssembly, written in Rust

- We’re looking to add an eBPF backend, to allow Rust to eBPF compilation
QUESTIONS?
FIND US ON GITHUB

JOIN US ON DISCORD