AYA

fedora

Building your eBPF Program with Rust and Aya

Fedora eBPF SIG Group

Daniel Mellado

Principal Software Engineer@Red Hat

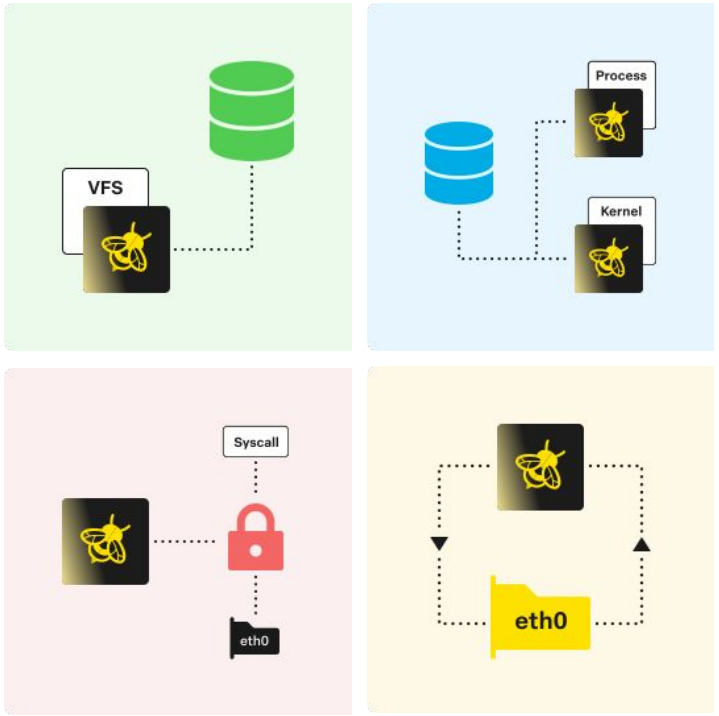
dmellado@fedoraproject.org

What is eBPF and how does it work?

What is eBPF?

- eBPF (Extended Berkeley Packet Filter) allows execution of sandboxed programs in the Linux kernel.
- eBPF is a technology that allows you to dynamically program the kernel for efficient networking, observability, tracing, and security.

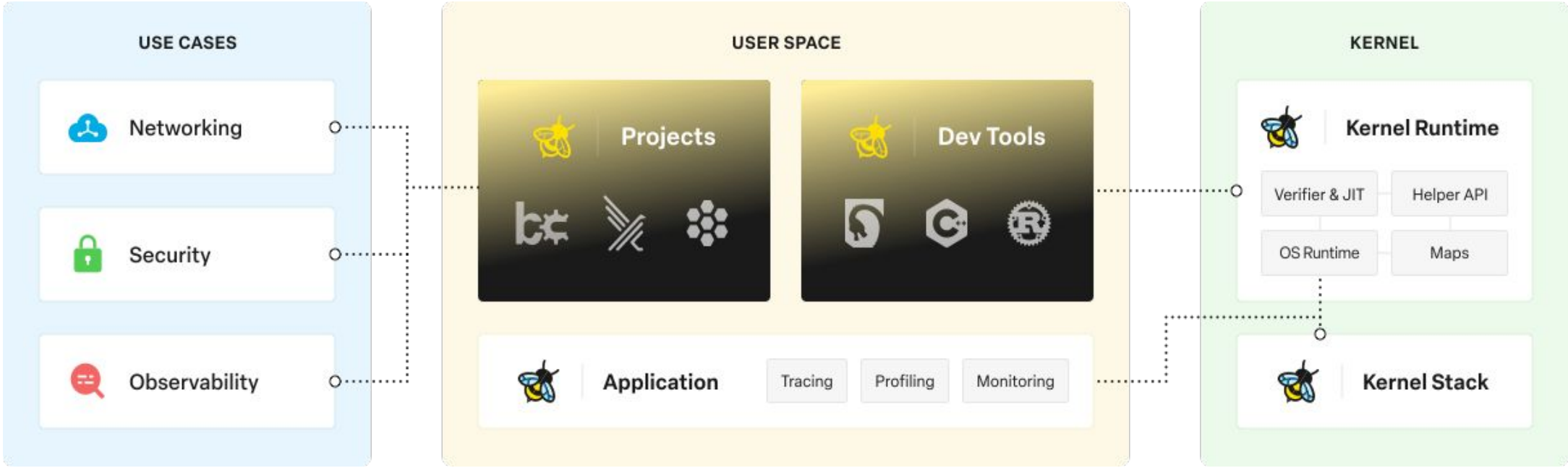
Head over to ebpf.io to learn more



dmellado@fedoraproject.org

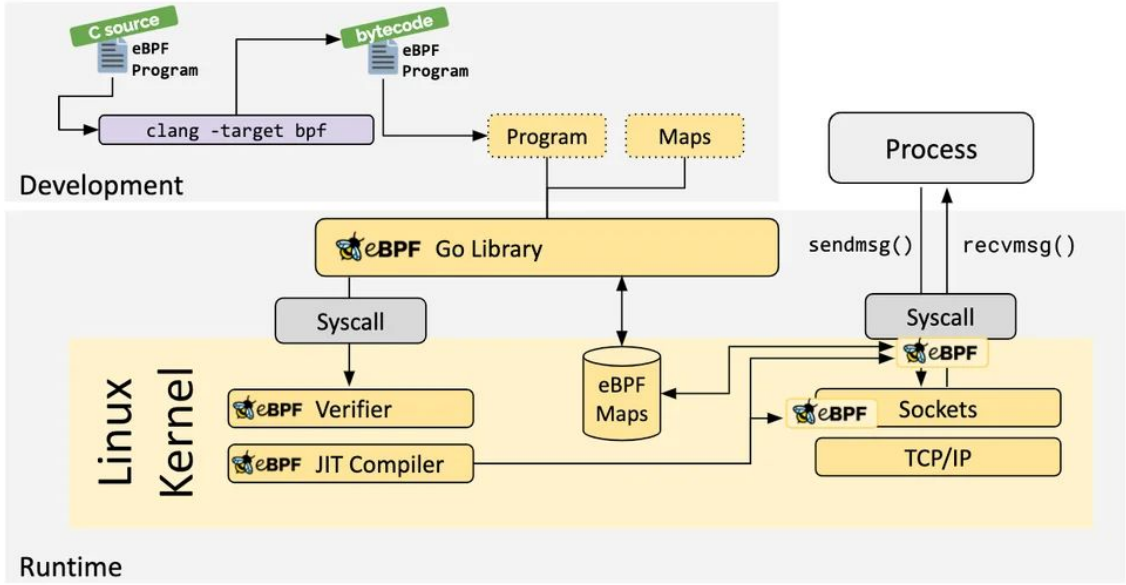
fedora

How does it work?



How does it work?

- Kernel Space
  - Attaches to eBPF hooks (e.g., network events, syscalls)
  - Performs specific tasks (filtering, tracing, security enforcement)
  - May write data into eBPF Maps
- User Space
  - Deploys eBPF programs to the kernel
  - May read data from eBPF Maps
- BPF Maps
  - Shared storage between user space and kernel space
  - Different map types (hash, array, etc.) for efficient data handling
  - Storage space is limited



Why Rust?

Other options

- Python
  - BPF Compiler Collection (BCC)
- C
  - Libbpf
- Go
  - gobpf (probably first one, deprecated)
  - Cillium ebpf
  - libbpf-go
- Rust
  - libbpf-rs
  - Redbpf (not maintained)
  - Aya

Enter Aya

Aya

- Built from scratch in Rust – No dependency on libbpf or bcc
- BPF Type Format Support – Automatically enabled for target kernels with BTF support.
- Function Call Relocation & Global Data Maps – Use function calls, global variables, and initializers
- Async Support – Works with tokio and async-std
- Fast Build & Easy Deployment – No kernel build, headers, or C toolchain required
- Compile Once, Run Everywhere – Musl-linked for portability across distros and kernel versions

Creating a program: aya-template

rustup install stable

rustup toolchain install nightly --component rust-src

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

⚠️ Favorite `https://github.com/aya-rs/aya-template` not found in config, using it as a git repository: https://github.com/aya-rs/aya-template

🔧 Destination: /home/dmellado/Devel/fosdem25/fosdem ...

🔧 project-name: fosdem ...

🔧 Generating template ...

✔ 🤷 Which type of eBPF program? · xdp

🔧 Moving generated files into: `/home/dmellado/Devel/fosdem25/fosdem`...

🔧 Initializing a fresh Git repository

✨ Done! New project created /home/dmellado/Devel/fosdem25/fosdem

Creating a program: aya-template

```
[dmellado@fedora  fosdem(main #%)]$ ll
total 24
-rw-r--r--. 1 dmellado dmellado 1068 ene 30 14:52 Cargo.toml
drwxr-xr-x. 3 dmellado dmellado 4096 ene 30 14:52 fosdem
drwxr-xr-x. 3 dmellado dmellado 4096 ene 30 14:52 fosdem-common
drwxr-xr-x. 3 dmellado dmellado 4096 ene 30 14:52 fosdem-ebpf
-rw-r--r--. 1 dmellado dmellado 1266 ene 30 14:52 README.md
-rw-r--r--. 1 dmellado dmellado  113 ene 30 14:52 rustfmt.toml
```

Structure

```
[dmellado@fedora fosdem(main %)]$ tree -L 3
.
├── Cargo.lock            # Automatically generated lock file that records the exact versions of dependencies
├── Cargo.toml            # Main configuration file for the project that specifies dependencies and metadata
├── fosdem                 # User-space application
│   ├── build.rs          # Custom build script to handle any non-Rust build tasks
│   ├── Cargo.toml        # Configuration file for the user-space app
│   └── src
│       └── main.rs       # Entry point for the user-space application
├── fosdem-common          # Shared code library reused by eBPF programs and user-space programs
│   ├── Cargo.toml        # Configuration file for the shared library
│   └── src
│       └── lib.rs        # The main code library used across both eBPF and user-space programs
├── fosdem-ebpf            # eBPF program
│   ├── build.rs          # Custom build script for eBPF-specific tasks
│   ├── Cargo.toml        # Configuration file for the eBPF program
│   └── src
│       ├── lib.rs        # Shared library or helper functions used in the eBPF program
│       └── main.rs       # Entry point for the eBPF program
├── README.md             # Project documentation and instructions for usage
├── rustfmt.toml          # Configuration file for Rust code formatting (via rustfmt)
└── target                # Output directory for compiled artifacts
    ├── CACHEDIR.TAG      # A file indicating that this is a directory for cached data
    └── debug             # Compiled artifacts during the debugging build process
        ├── build         # Build output directory (contains build artifacts)
        ├── deps          # Dependencies directory
        ├── examples      # Compiled example programs
        ├── fosdem        # Compiled user-space program
        ├── fosdem.d      # Debug information for the user-space program
        ├── incremental   # Incremental build information
        ├── libfosdem_common.d  # Debug information for the common library
        └── libfosdem_common.rlib # Compiled library for the shared code
```

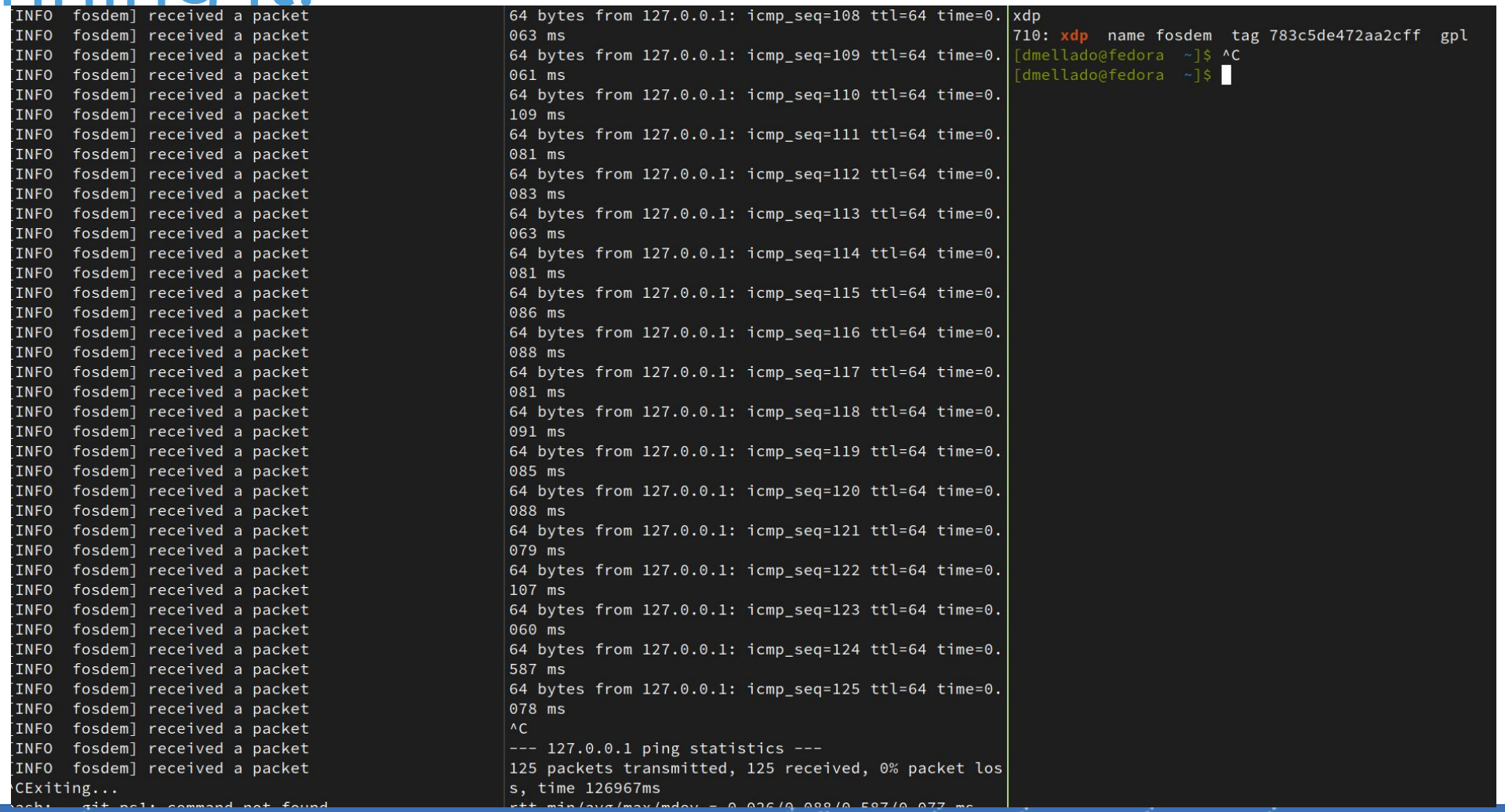
Compile and test

- Totally straightforward!
- Just cargo build and cargo check would do it

Structure

```rust
#![no_std]
#![no_main]

use aya_ebpf::{bindings::xdp_action, macros::xdp, programs::XdpContext};
use aya_log_ebpf::info;

#[xdp]
pub fn fosdem(ctx: XdpContext) -> u32 {
    match try_fosdem(ctx) {
        Ok(ret) => ret,
        Err(_) => xdp_action::XDP_ABORTED,
    }
}

fn try_fosdem(ctx: XdpContext) -> Result<u32, u32> {
    info!(&ctx, "received a packet");
    Ok(xdp_action::XDP_PASS)
}

#[cfg(not(test))]
#[panic_handler]
fn panic(_info: &core::panic::PanicInfo) -> ! {
    loop {}
}
```

Running it!


[INFO fosdem] received a packet
64 bytes from 127.0.0.1: icmp_seq=108 ttl=64 time=0.
063 ms
64 bytes from 127.0.0.1: icmp_seq=109 ttl=64 time=0.
061 ms
64 bytes from 127.0.0.1: icmp_seq=110 ttl=64 time=0.
109 ms
64 bytes from 127.0.0.1: icmp_seq=111 ttl=64 time=0.
081 ms
64 bytes from 127.0.0.1: icmp_seq=112 ttl=64 time=0.
083 ms
64 bytes from 127.0.0.1: icmp_seq=113 ttl=64 time=0.
063 ms
64 bytes from 127.0.0.1: icmp_seq=114 ttl=64 time=0.
081 ms
64 bytes from 127.0.0.1: icmp_seq=115 ttl=64 time=0.
086 ms
64 bytes from 127.0.0.1: icmp_seq=116 ttl=64 time=0.
088 ms
64 bytes from 127.0.0.1: icmp_seq=117 ttl=64 time=0.
081 ms
64 bytes from 127.0.0.1: icmp_seq=118 ttl=64 time=0.
091 ms
64 bytes from 127.0.0.1: icmp_seq=119 ttl=64 time=0.
085 ms
64 bytes from 127.0.0.1: icmp_seq=120 ttl=64 time=0.
088 ms
64 bytes from 127.0.0.1: icmp_seq=121 ttl=64 time=0.
079 ms
64 bytes from 127.0.0.1: icmp_seq=122 ttl=64 time=0.
107 ms
64 bytes from 127.0.0.1: icmp_seq=123 ttl=64 time=0.
060 ms
64 bytes from 127.0.0.1: icmp_seq=124 ttl=64 time=0.
587 ms
64 bytes from 127.0.0.1: icmp_seq=125 ttl=64 time=0.
078 ms
^C
--- 127.0.0.1 ping statistics ---
125 packets transmitted, 125 received, 0% packet los
s, time 126967ms
^CExiting...
xdp
710: xdp name fosdem tag 783c5de472aa2cff gpl
[dmellado@fedora ~]$ ^C
[dmellado@fedora ~]$

Real usage: bpfman

Fedora ebpf-sig group

Fedora eBPF SIG Group

- A new sig group was created in late 2023 to gather interest around eBPF in Fedora. Fedora eBPF Special Interest Group
- Identified bpfman as a useful tool to use as a bpf manager and decided to push for it to be included in Fedora.

dmellado@fedoraproject.org

fedora

Questions?

Thank you for attending!