AYA

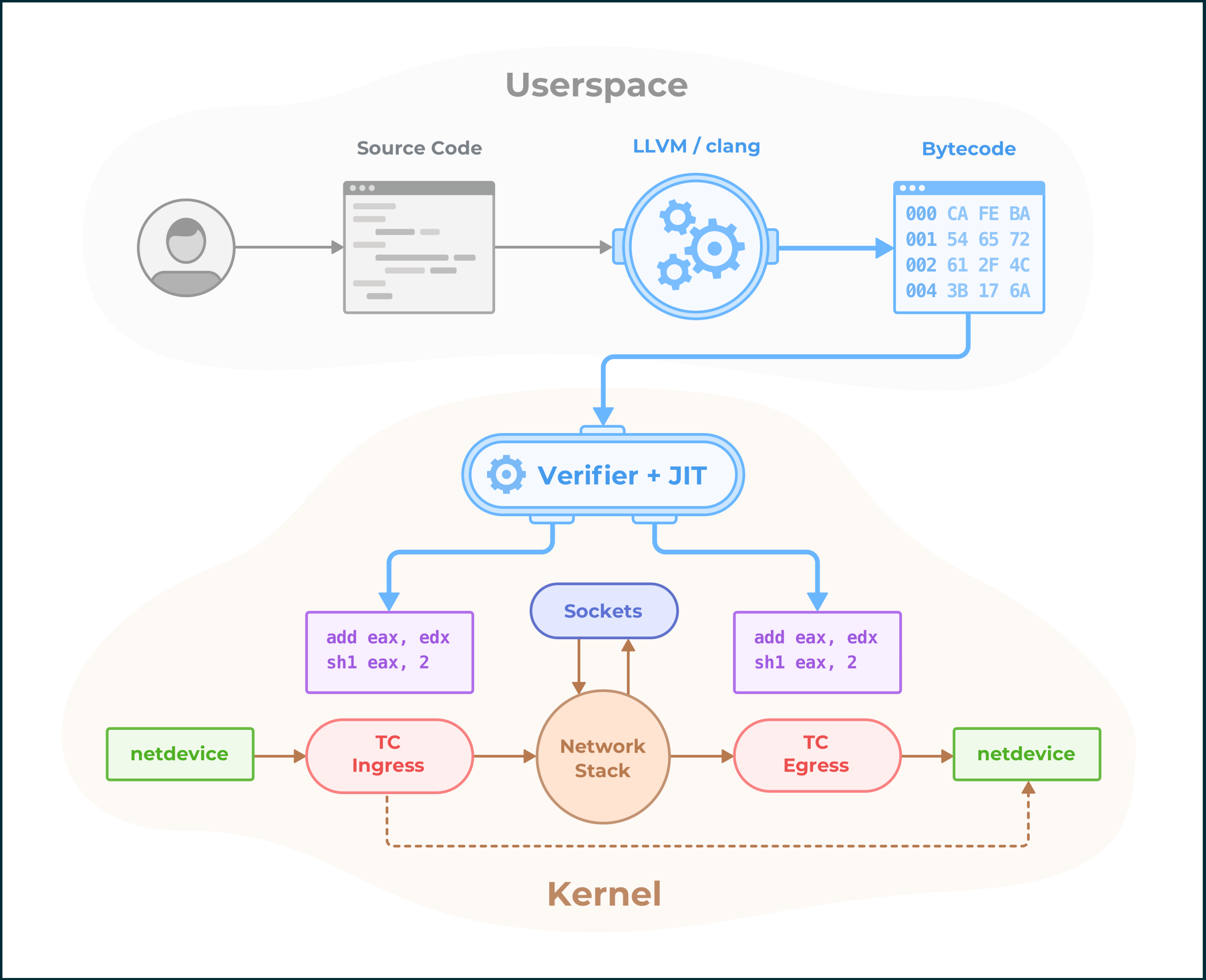
Easy development of eBPF programs in Rust.

ABOUT ME

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

WHAT IS EBPF?

- Technology which runs sandboxed programs in an operating system kernel.
- Event-driven, triggered by events in the kernel, receiving pointers to kernel or userspace memory.
- No need to modify kernel code or load kernel modules. eBPF programs can be just loaded and run.


Userspace
Source Code
LLVM / clang
Bytecode
000 CA FE BA
001 54 65 72
002 61 2F 4C
004 3B 17 6A
Verifier + JIT
Sockets
add eax, edx
sh1 eax, 2
add eax, edx
sh1 eax, 2
netdevice
TC Ingress
Network Stack
TC Egress
netdevice
Kernel

WHAT EVENTS?

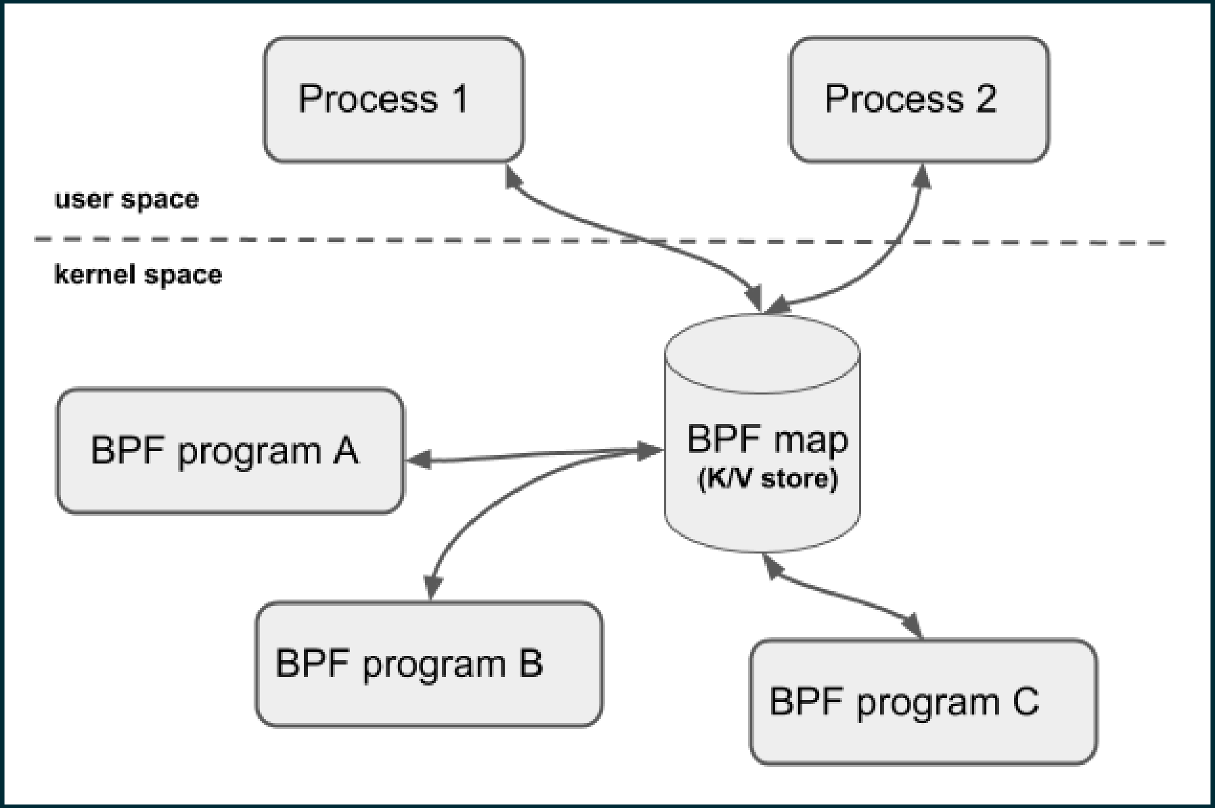
- kernel function calls
  - or tracepoints
- userspace function calls
- network packets
- messages on sockets
- actions which trigger LSM hooks
  - filesystem operations
  - allocating and freeing processes
  - changing the user identity

BPF PROGRAM TYPES

- For each type of events there is a separate type of program.
- Examples:
  - **KProbe** (or **FEntry**) - for kernel function tracing
  - **UProbe** - for userspace function tracing
  - **Classifier** - for network packet inspection (sk_buff)
  - **XDP** - for network packet inspection (raw packet on NIC)
  - **LSM** - for security policies
- Each type has its own requirement of kernel version.

BPF MAPS

- Storage for sharing data between eBPF programs (in kernel) and userspace.
- Different types:
  - hash maps
  - arrays
  - perf buffers (buffers for pushing events to userspace)
- Can be available in BPFFS (`/sys/fs/bpf`)


Process 1
Process 2
user space
kernel space
BPF map
(K/V store)
BPF program A
BPF program B
BPF program C

LIFETIME OF A BPF PROGRAM

- BPF programs have **links** - they are attached to events.
- By default, links are dropped when the `Bpf` object in the userspace project is dropped.
- You can detach the links earlier if you want.
- TODO: support link persistence by pinning in BPFFS (it's possible in libbpf)

EXAMPLES OF BPF PROJECTS

- Cilium - CNI plugin, Kubernetes networking, service mesh
- Katran - L4 load balancer
- Tracee - application tracing tool
- Falco - security monitoring for containers

EBPF PROJECT LAYOUT

- **userspace program**: loading eBPF programs to the kernel
- **eBPF programs**: running in the kernel and reacting to events

"OFFICIAL" WAYS TO USE EBPF

- **bpftrace**: very easy, own scripting language
- **libbpf**: umm... not easy at all (everything in C)
- **libbpf-rs**: easier (eBPF programs in C, userspace program in Rust)
- **bcc**: easier (eBPF programs in C,)

WHAT'S SO HARD ABOUT LIBBPF?

- Everything written in C.
- Official docs are limited. Fragmented information in multiple websites and blogs.
- Necessity to create custom Makefile, write boilerplate code.
- There is a bootstrap repo, but it requires a lot of manual tweaking.
- No logging. You can use `bpf_printk`, but you shouldn't. Logging is usually implemented from scratch in each project.

WHY AYA?

- Everything in Rust, also eBPF programs!
- rustup, rustc and cargo is all you need. No additional libs or tools.
- One central book/documentation.
- Template ready to use with `cargo generate`.

WHY RUST?

- Using language features like `Result`, `Option`.
- Static linking and portable binaries.
- You can keep both eBPF and userspace part in one language.
  - And share the structures.

MEMORY SAFETY?

- Actually... Rust memory model doesn't matter that much on eBPF part.
- eBPF has access to kernel and userspace memory via pointers, which are unsafe.
- Memory safety relies more on the eBPF verifier in the kernel rather than Rust compiler.

HOW TO START WITH AYA?

```
rustup toolchain install nightly --component rust-src
cargo install bpf-linker
cargo install cargo-generate
```

STARTING A NEW PROJECT

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

RUNNING THE PROJECT

```
cargo xtask run
```

Only building:

```
cargo xtask build-ebpf
cargo build
```

Talk is cheap. Show me the code.

— *Linus Torvalds* —

AZ QUOTES

THANK YOU

- Documentation
- Github
- Discord