# Day 1

## What should be the anthem of BPF?

- Universal assembly language, nothing more, nothing less

### Discussion

- AI: Guidelines for LLVM submissions to make them more visible to BPF list

## Compiled BPF

### LLVM-specific builtins

- builtins relying on LLVM internals that allow it to infer typess, but optimizations in GCC make this difficult. Not explicit.
- Change headers to pass the types in => Need to change LLVM to also recognize these.
- Difficulty: C doesn't thave a way to express types to be passed in.
- BTF? Happens too late.
- C11 _Generic as a future answer
- For now, macros in the headers is ok.

### BPF Assembly dialects

Assembler-like and pseudo-c dialects. Proposal to move away from assembler-like and support assembler syntax.

Pain not understood by the room. Already implemented, what's the problem?

### BTF generation

- DWARF is currently an intermediate step. Hard to extend. Standardized. etc.
- Generate BTF directly

- Could provide Linux compilation time speedup
- How is BTF deduplication going to work? CTF uses a parent/child relationship for dedup.
- Maybe pahole solves this already?

## Kernel bpf_insn representation

- Support future instruction classes in the definition without breaking stability?
- Already have 8B and 16B instructions. We have infinite codepoints.
- Can define this with new structure.

## Unifying kfunc and helper definitions

- Patterns for resource tracking, value poisoning etc.; compare helpers/kfun

## Unprivileged BPF

- Tokens associated to a BPFFS instance to grant BPF privileges to trusted, unpriv applications (including inside user namespaces)
- BPF + User namespaces doesn't work at all today due to CAP check
- Detailed discussion about the nature of the tokens - separate lifecycle + pinned to an FS (or passed via unix socket) or directly tied to FS
- Systemd folks had strong opinions about tying to BPFFS instance - Having a handle to the FS is the same as having the token
- Some disagreement from the room incl. KP

…

# Day 2

## Generic multi-attach API

- BPF_F_BEFORE - Immediately before
- BPF_F_FIRST/BPF_F_LAST - Only my program. Permanent.
- "A must be before B"
    - At load time, immediately before the target prog.
    - This is not a permanent constraint.

        - If "Before B" fails, userspace should just retry to figure out what else it should be before.
- Some discussion about revision numbers for attachments, so if anything changes the command may return a failure. Good tool for if you have strict ordering requirements.
- Optional args => last parameter as a list of options.
- Some discussion regarding first|last flags.
    - What's the use case?
        - e.g. ensure your DoS prevention runs first
        - Hard requirement for security tooling
    - Concern that one program would lock everyone else out
    - Prepend/Append: This is "BEFORE" or "AFTER" with prog_id 0
    - Request for Admin override for prog-specified flags
        - Is this just a force-detach semantic?
    - Better naming for clearer semantics
- Why prog_ids vs. fds. Most APIs use fd today.
    - When you dump progs, you get ids. Extra step to convert to fd which need CAP_SYS_ADMIN permissions. With ids this can be avoided.
- Should we be moving towards ids by default in new APIs instead of fds?
    - Maybe yeah
- Add support for Cgroups, LSM

## Meta devices

- Latest updates, plan
- Some discussion about semantics

## XDP metadata

- Metadata for more efficient CPU usage, NIC cooperation for offloads
- RX is mostly implemented, TX there is no infra or implementations.
- Discussion on two hooks - pre completion + post
    - pre: Ask NIC for timestamps
    - post: NIC provides timestamps, invoke callback
- Naming discussion - Seems symmetric with XDP. Why not "XDP Egress"?
    - More tracing-style
- Completion hook seems most interesting to begin with

## Better hash functions for maps

## Sockets iterator

- Discussion on the semantics of the "bool global" for iteration of all sockets.
    - Concern it's a bit limited
    - Use an enum?
    - 0 => current namespace
    - 1 => global iterator
    - 2 => namespace by fd
        - Pass fd as
- fd=0 is technically legit fd, but BPF API mostly assumes it is not. Loader libraries do dup to ensure fd is nonzero

## Cgroup ID API improvements

- Negative ancestor level input seems favourable.
- No error detection since the result is u64
- AI(David Vernett): Apply these semantics also to the kfunc he recently added

## BPFilter

- Difficulty with developing in Linux tree
- Userspace daemon
- Some use cases not solvable with module
- Stateful daemon, no support for restarting/restore.
    - Why stateful?
        - Simpler
        - Concerns about how to find statistics for instance
- UDS format?
    - Binary iptables format
- New "flavour" - BPF prog inputs, outputs.
- 5tuple support exists. Not supporting the long tail of iptables features today.
- Longer term use case, add new "match" statements specific to their use case, to optimize e.g. "pool of hosts" lookup.

## sched_ext

- Leaky, complicated abstraction in core sched class
- Simple abstraction of callbacks with default behaviours that you implement
- Dispatch queues are understood by both kernel and BPF land
- Can direct dispatch locally, or enqueue elsewhere

- …
- Interesting questions/discussions about API/ABI stability
- Distros concerned about 3rd party schedulers
    - Just do bpftool dump and list all progs. If one is attached and you don't recognize it, don't support it.
    - BPF guarantee: No kernel crashes
    - Do upstreamed BPF progs for sched expose internals as UAPI?
    - General intent is no, minimal we can do is to document this.