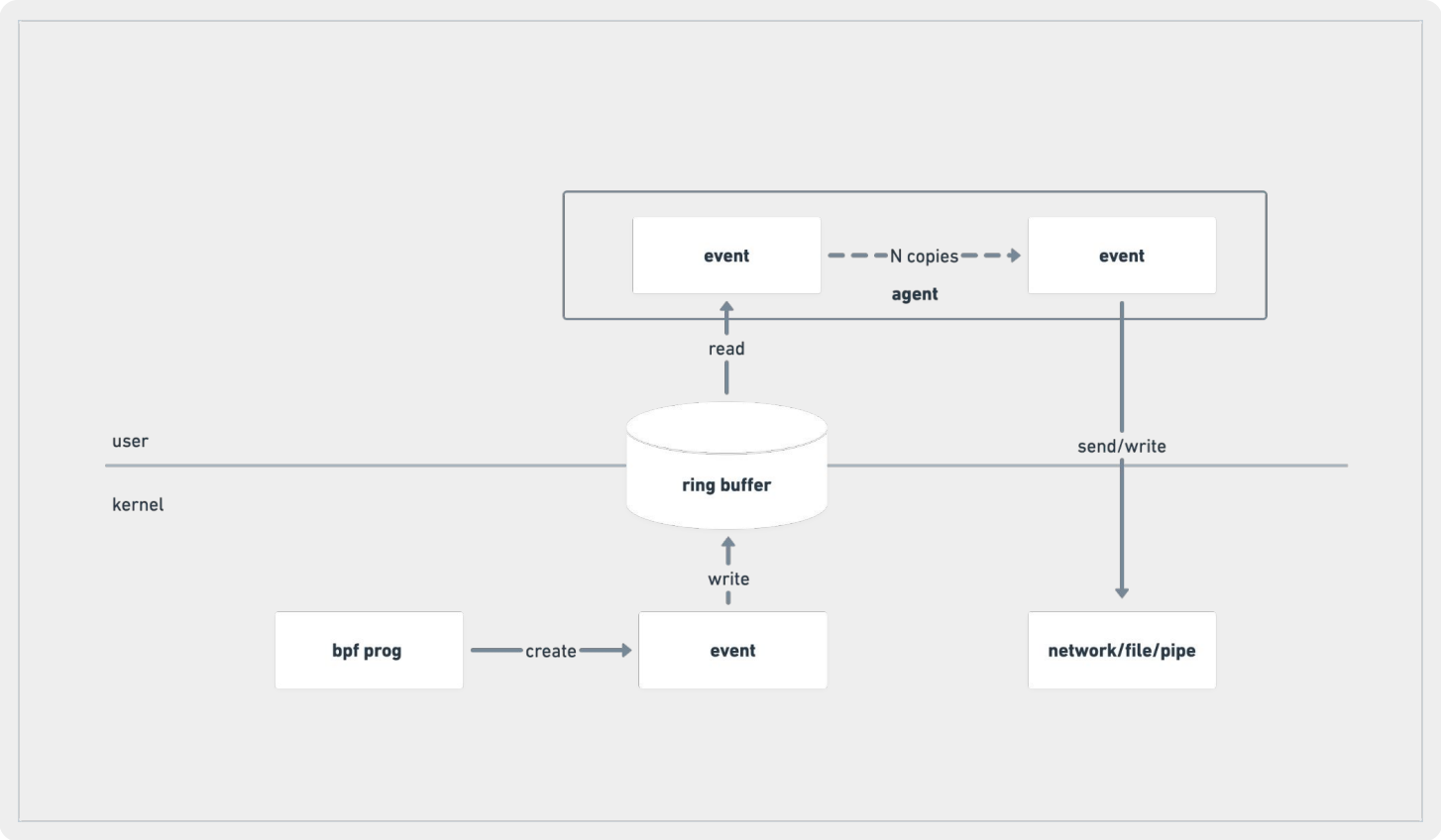
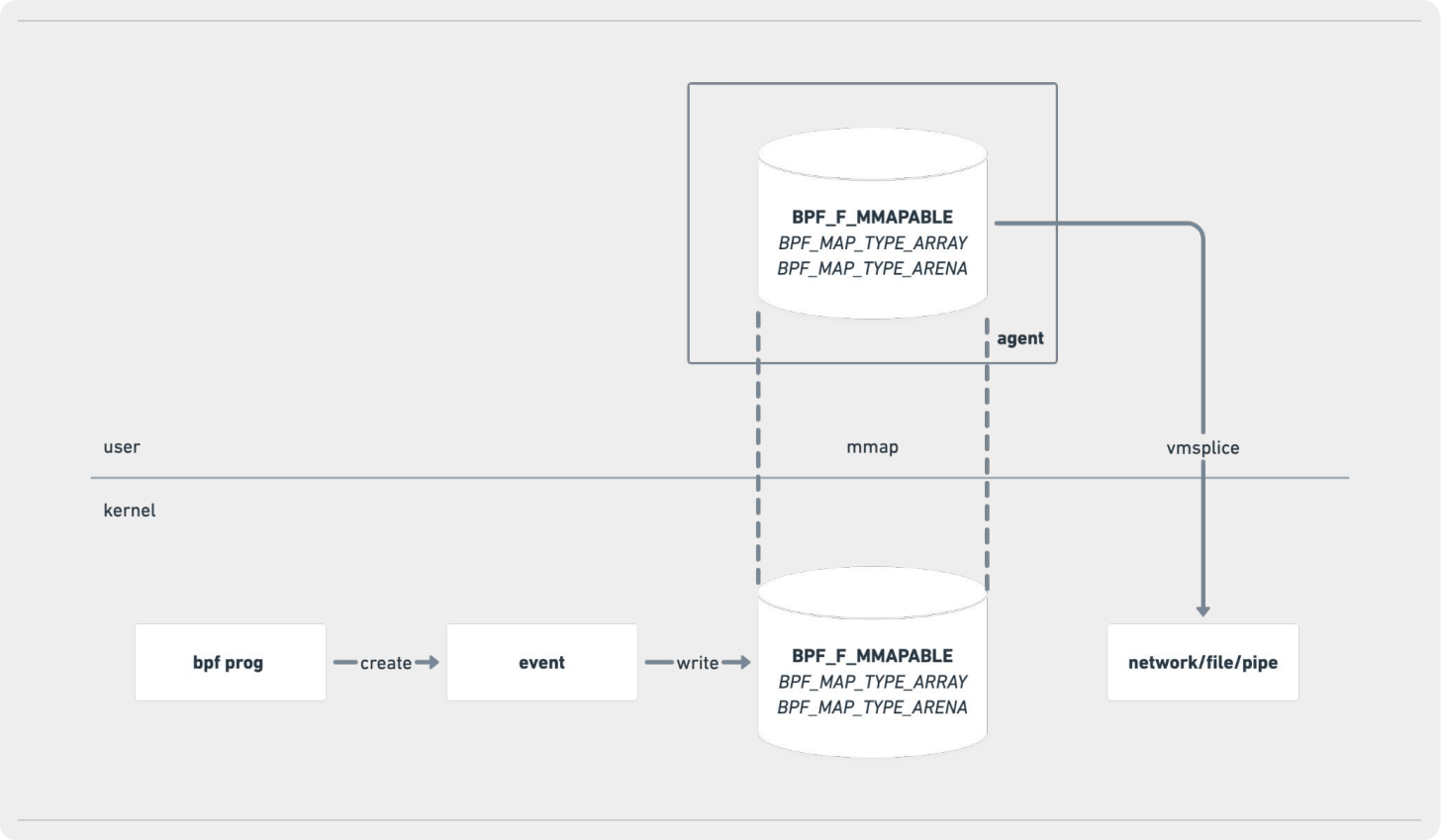
Splicing BPF maps to the NIC

John Fastabend, Kornilios Kourtis, Mahé Tardy

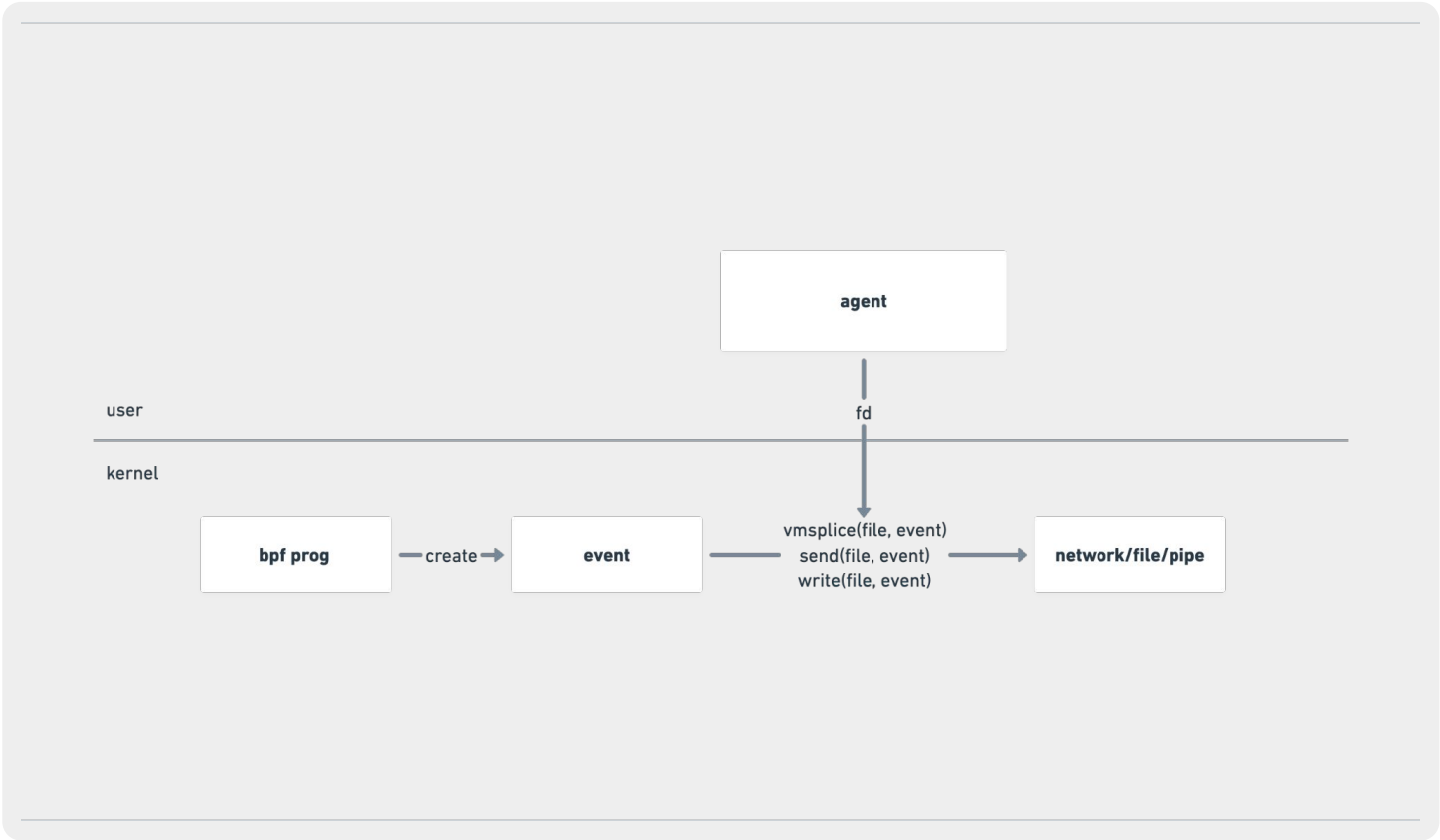
Case A: events journey in Tetragon



Case B: less copying



Case C: low latency BPF events



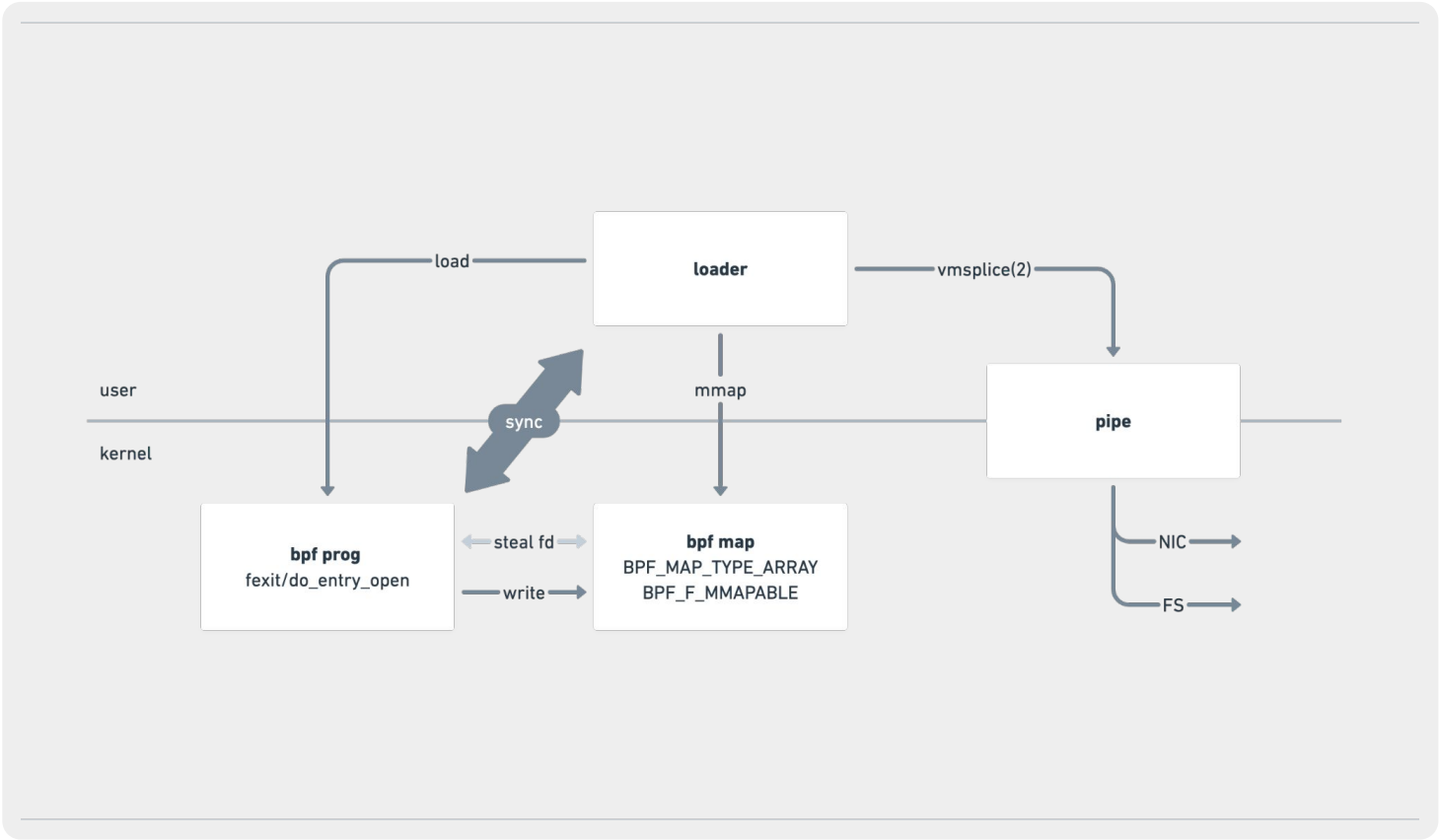
Main motivation

- Running without a userspace agent
- No perf ring issues to transmit event/alert between BPF and the agent
- Instant alerting mechanism for low latency
- Less context switch and copies of events

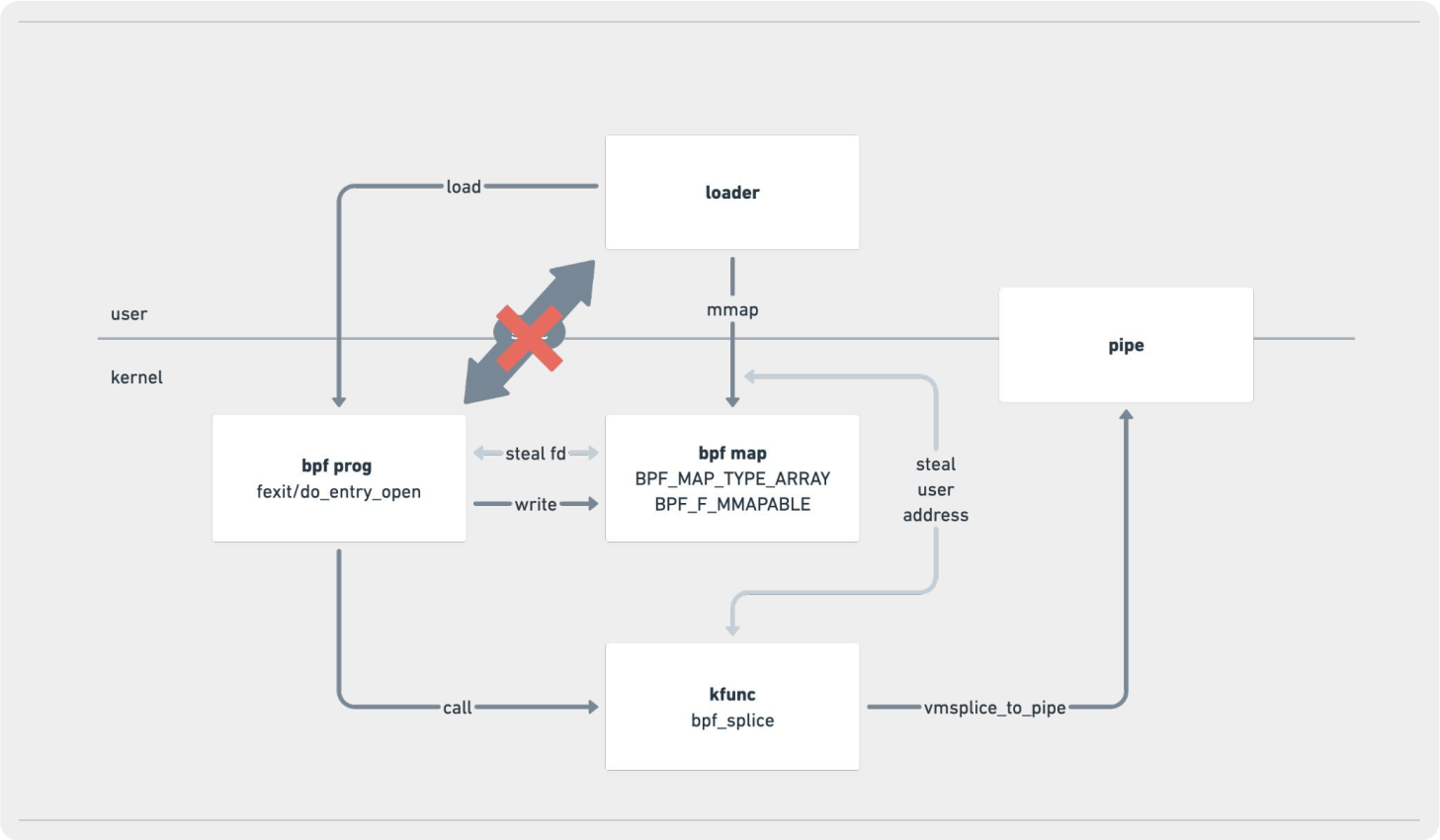
Why splice

- We initially thought of splice because it allows for zero copy between kernel and userspace
- Also generic and could be used to wire files/network.
- Splice seems to not be in a very appealing position nowadays (see Rethinking splice())
- Maybe use io_uring for async IO from bpf? (see Add bpf for io_uring, not really related but allowing to run bpf prog from io_uring events)
- Maybe just start with network send.

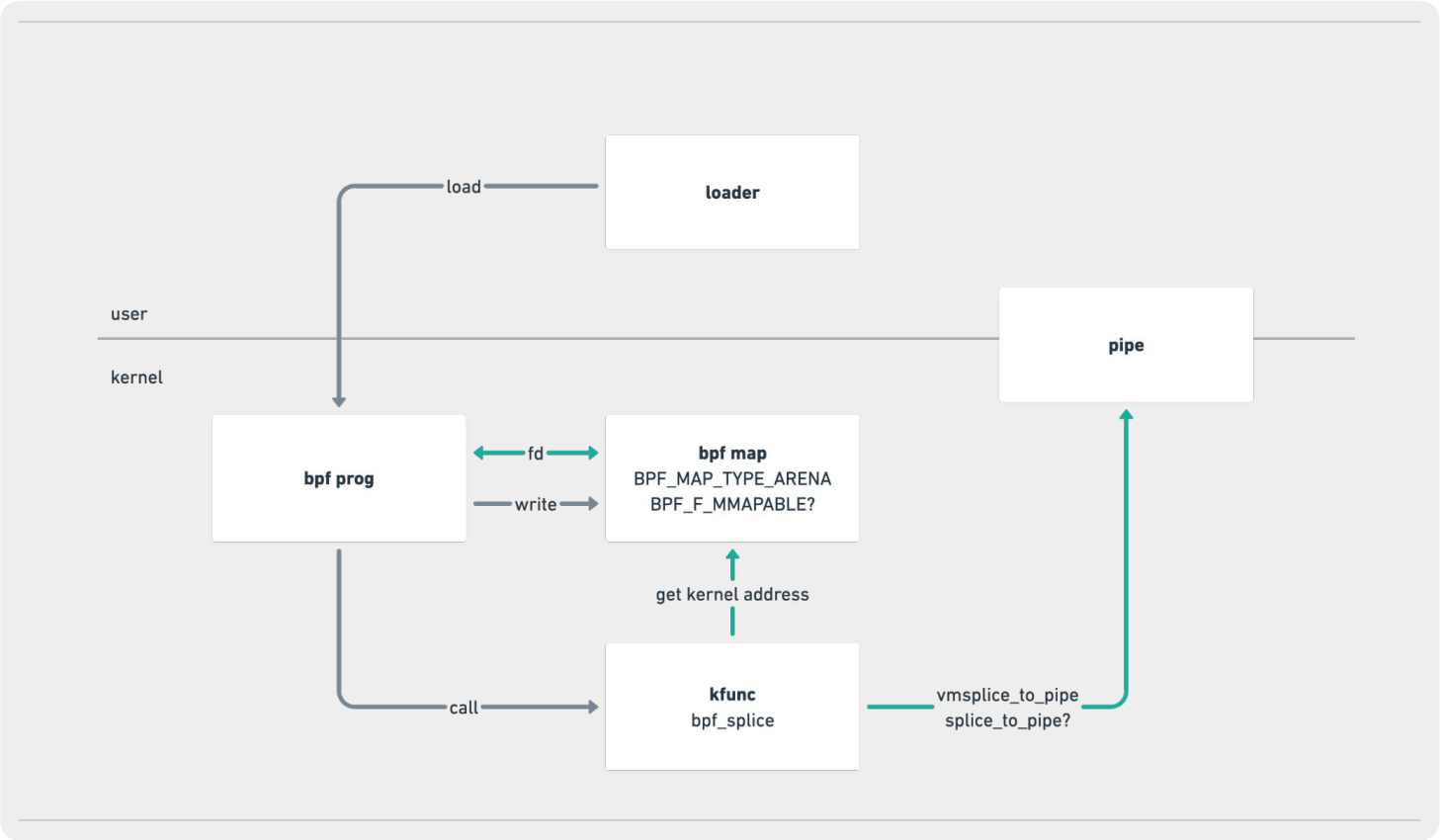
PoC B


load
loader
vmsplice(2)
user
kernel
sync
mmap
pipe
bpf prog
fexit/do_entry_open
steal fd
write
bpf map
BPF_MAP_TYPE_ARRAY
BPF_F_MMAPABLE
NIC
FS

PoC C


load
loader
user
mmap
pipe
kernel
bpf prog
fexit/do_entry_open
steal fd
write
bpf map
BPF_MAP_TYPE_ARRAY
BPF_F_MMAPABLE
steal
user
address
call
kfunc
bpf_splice
vmsplice_to_pipe

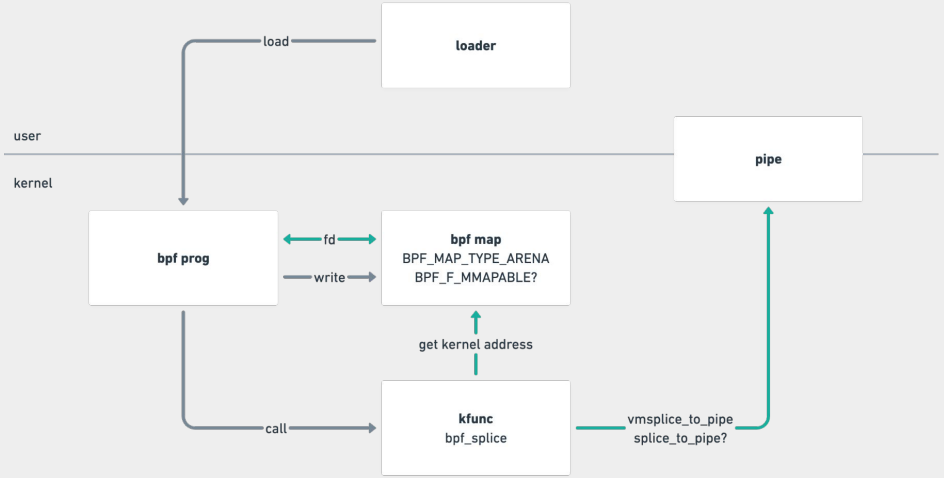
PoC C goal


load
loader
user
kernel
pipe
bpf prog
fd
write
bpf map
BPF_MAP_TYPE_ARENA
BPF_F_MMAPABLE?
get kernel address
call
kfunc
bpf_splice
vmsplice_to_pipe
splice_to_pipe?

Discussions

- Was this idea already discussed?
- Start with network instead of splice (see Rethinking splice())?
- What kind of interface for passing the fd?

- Benchmark different solutions? What's the benefit of approach C over B?
- IO approach?
  - sleepable, in kfunc
  - or using io_uring queues
- Latency (alerting) vs bandwidth (send to a data lake) use-cases:
  - Approach C seems better than B for the latency use-case
- Output format? Could use apache arrow.



Kfunc Timer kit [agentless send]

```
int sent;
struct bpf_timer *timer; //sleepable timers
int __arena *ptr

main() {
  bpf_timer_set_callback(timer, send_callback);
  bpf_timer_start(timer, nsec_future, 0);
}

void send_callback() { // sleepable callback
   ptr = &my_bpf_map; // arena
   tcp_sk = bpf_map_lookup(&tcp_endpoints, 0);
   sent += bpf_send(tcp_sk, kptr, sizeof(my_bpf_map));
   bpt_timer_start(timer, nsec_future, 0)
}
```