@fntlnz

Sysdig

eBPF-powered distributed Kubernetes performance analysis

Lorenzo Fontana. Open Source Software Engineer, Sysdig.

eBPF

Berkley Packet Filter

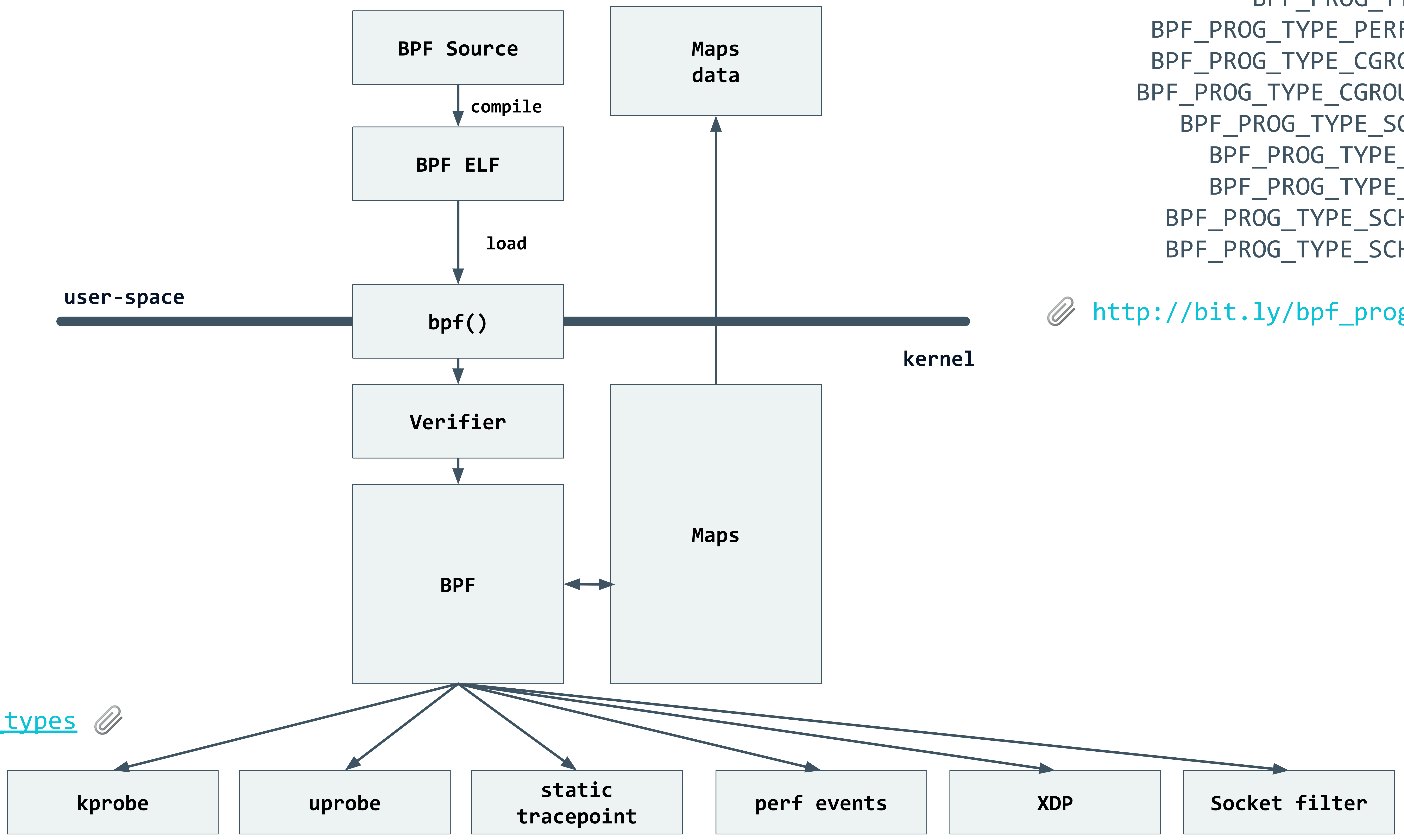
extended BPF

extended because it’s **not just** packets anymore

How does eBFP work?

BPF_PROG_TYPE_SOCKET_FILTER
BPF_PROG_TYPE_KPROBE
BPF_PROG_TYPE_TRACEPOINT
BPF_PROG_TYPE_RAW_TRACEPOINT
BPF_PROG_TYPE_XDP
BPF_PROG_TYPE_PERF_EVENT
BPF_PROG_TYPE_CGROUP_SKB
BPF_PROG_TYPE_CGROUP_SOCK
BPF_PROG_TYPE_SOCK_OPS
BPF_PROG_TYPE_SK_SKB
BPF_PROG_TYPE_SK_MSG
BPF_PROG_TYPE_SCHED_CLS
BPF_PROG_TYPE_SCHED_ACT

http://bit.ly/bpf_prog_types



BPF_MAP_CREATE
BPF_MAP_LOOKUP_ELEM
BPF_MAP_UPDATE_ELEM
BPF_MAP_DELETE_ELEM
BPF_MAP_GET_NEXT_KEY

http://bit.ly/bpf_map_types

Aggregate events at kernel level

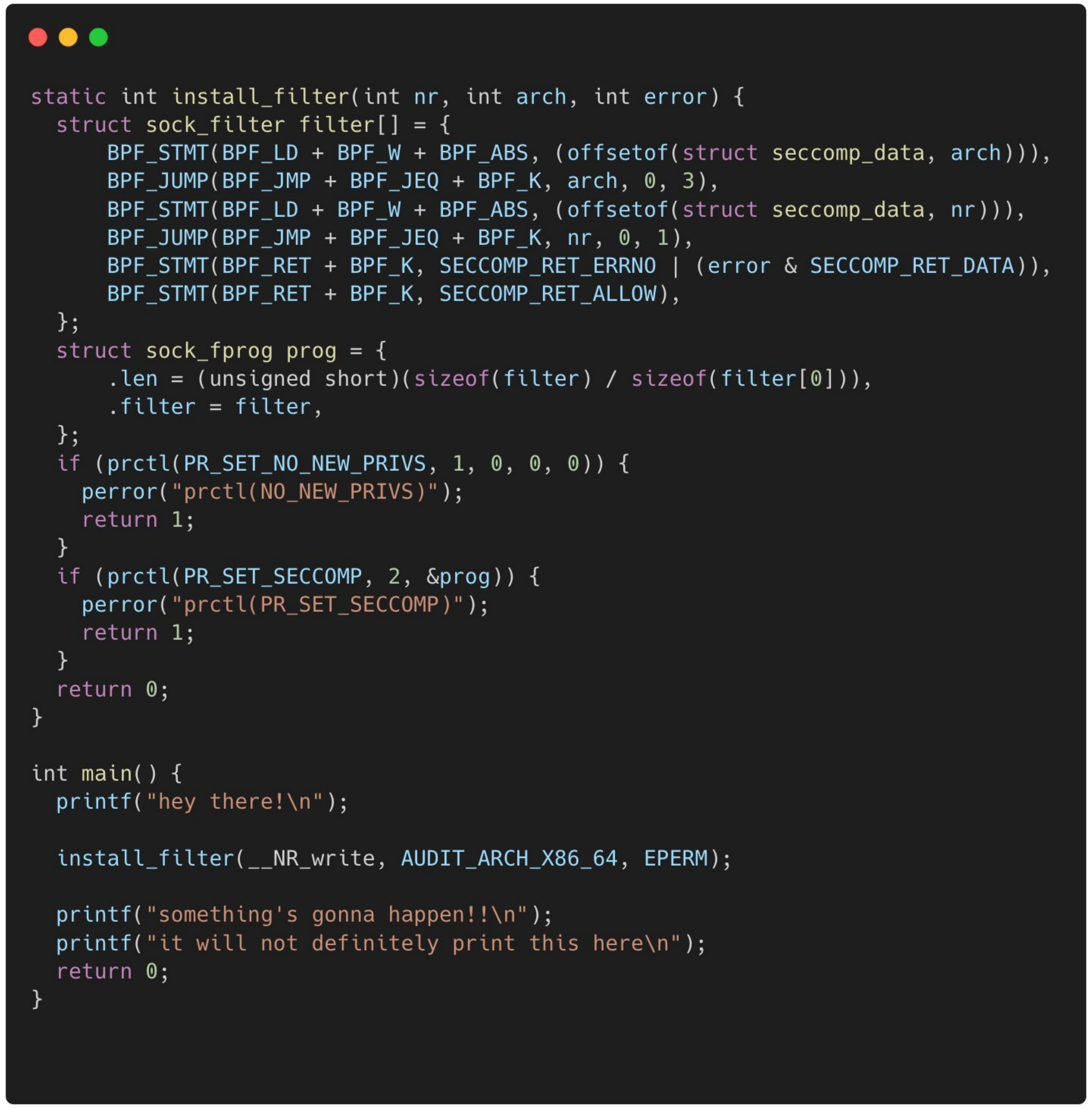
and deal with just a few

instead of thousands of them

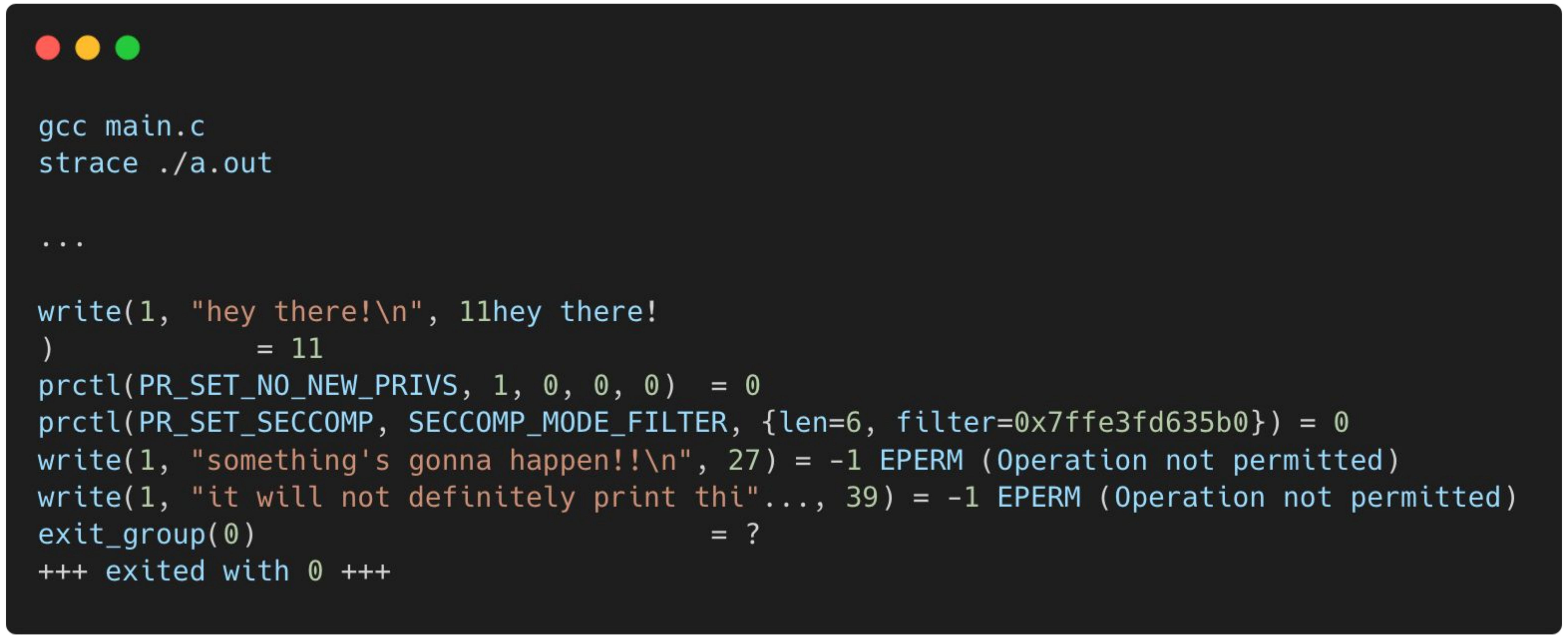
BPF and eBPF In today’s world

@fntlnz

Today's world BPF: seccomp-bpf



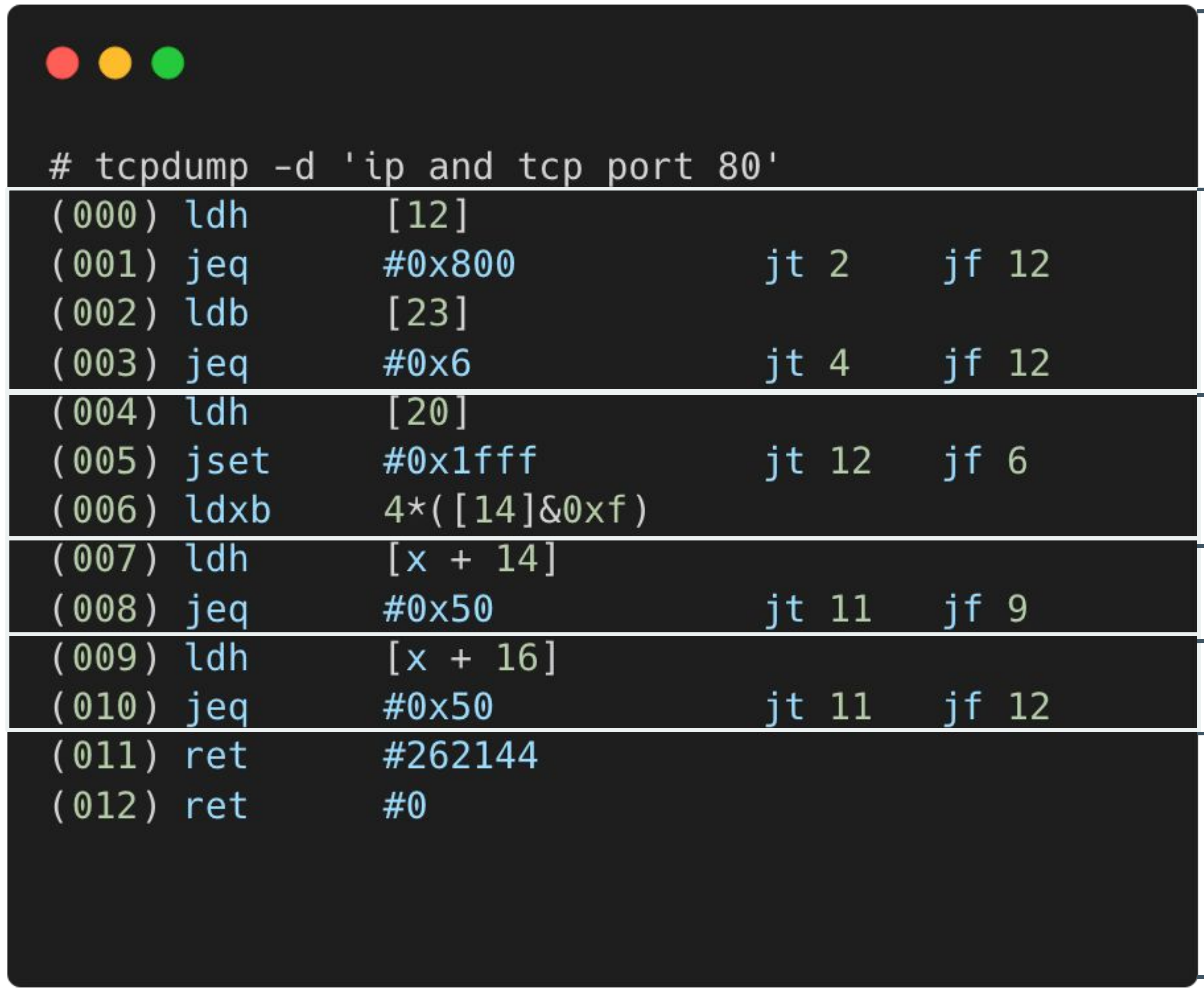
```c
static int install_filter(int nr, int arch, int error) {
  struct sock_filter filter[] = {
      BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, arch))),
      BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, arch, 0, 3),
      BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
      BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, nr, 0, 1),
      BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ERRNO | (error & SECCOMP_RET_DATA)),
      BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
  };
  struct sock_fprog prog = {
      .len = (unsigned short)(sizeof(filter) / sizeof(filter[0])),
      .filter = filter,
  };
  if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
    perror("prctl(NO_NEW_PRIVS)");
    return 1;
  }
  if (prctl(PR_SET_SECCOMP, 2, &prog)) {
    perror("prctl(PR_SET_SECCOMP)");
    return 1;
  }
  return 0;
}

int main() {
  printf("hey there!\n");

  install_filter(__NR_write, AUDIT_ARCH_X86_64, EPERM);

  printf("something's gonna happen!!\n");
  printf("it will not definitely print this here\n");
  return 0;
}
```



```
gcc main.c
strace ./a.out

...

write(1, "hey there!\n", 11hey there!
)             = 11
prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)  = 0
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, {len=6, filter=0x7ffe3fd635b0}) = 0
write(1, "something's gonna happen!!\n", 27) = -1 EPERM (Operation not permitted)
write(1, "it will not definitely print thi"..., 39) = -1 EPERM (Operation not permitted)
exit_group(0)                           = ?
+++ exited with 0 +++
```

@fntlnz

Today’s world BPF: tcpdump



```
# tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 12
(002) ldb      [23]
(003) jeq      #0x6             jt 4    jf 12
(004) ldh      [20]
(005) jset     #0x1fff          jt 12   jf 6
(006) ldxb     4*([14]&0xf)
(007) ldh      [x + 14]
(008) jeq      #0x50            jt 11   jf 9
(009) ldh      [x + 16]
(010) jeq      #0x50            jt 11   jf 12
(011) ret      #262144
(012) ret      #0
```

-d means: Dump the compiled packet-matching code in a human readable form

Does this ethernet frame contain an IPv4 Packet (ethertype 0x800? And the protocol is TCP (0x6) ?

Initialize packet and frame offset to “x”

Is src (x+14) on port 80 (0x50)?

Is dst (x+16) on port 80 (0x50)?

When a match Is found return the snap len, 262144,
It can be set with the -s parameter

@fntlnz

Open Source tools using eBPF

Tool	Description	GitHub
Falco	Container Runtime Security	https://github.com/falcosecurity/falco
BCC	Makes eBPF programs easier to write	https://github.com/iovisor/bcc
bpftrace	High-level tracing language for eBPF	https://github.com/iovisor/bpftrace
kubectl trace	bpftrace for Kubernetes!	https://github.com/iovisor/kubectl-trace
cilium	API Aware Networking and Security using BPF and XDP	https://github.com/cilium/cilium

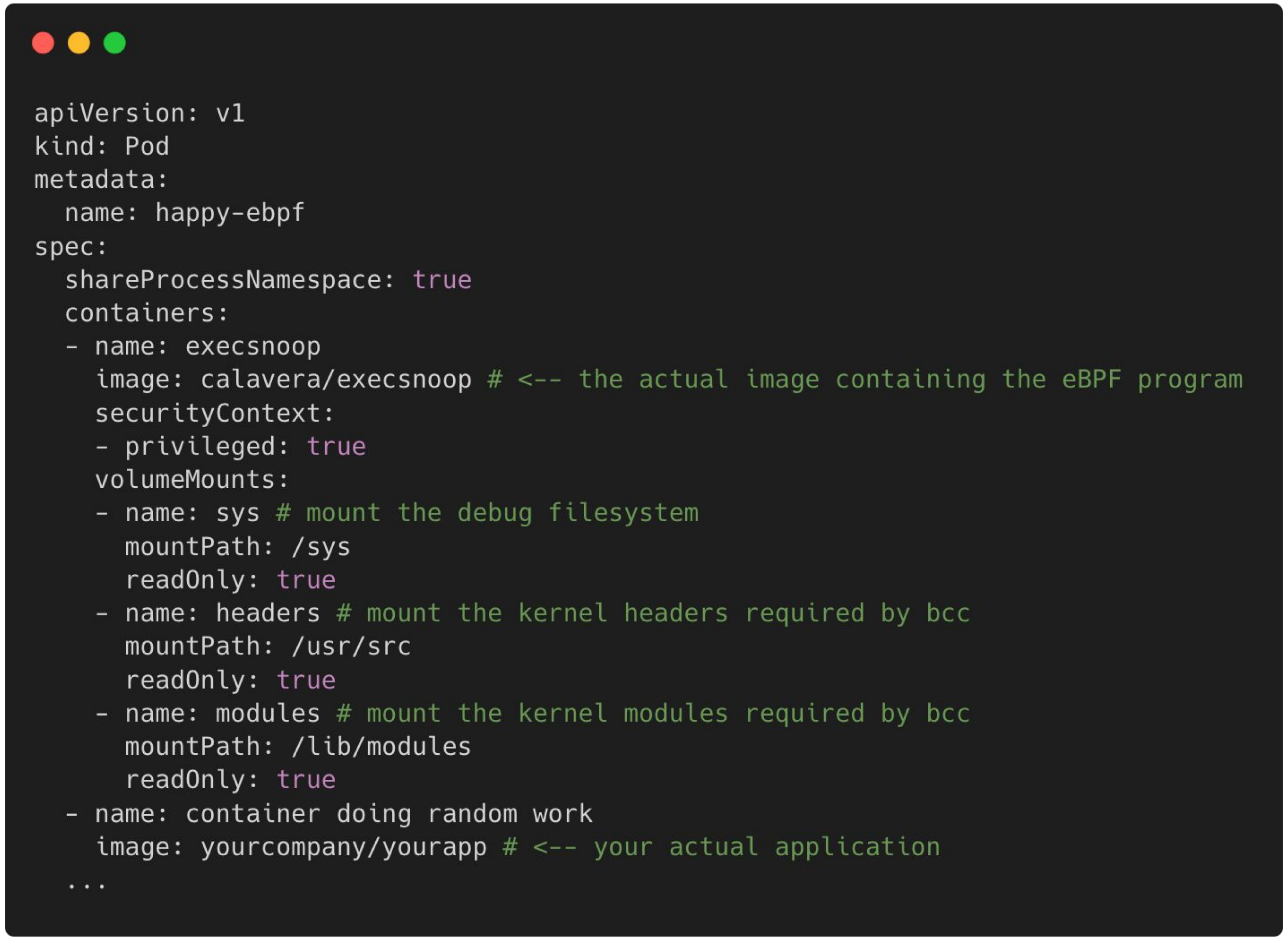
What is performance analysis about?

Performance analysis is a **quantitative** and **systematic** approach to **identify** performance issues in a software by doing:

- Measurement of time
- Measurement of space
- Measurement of complexity
- Profiling
- Code Instrumentation

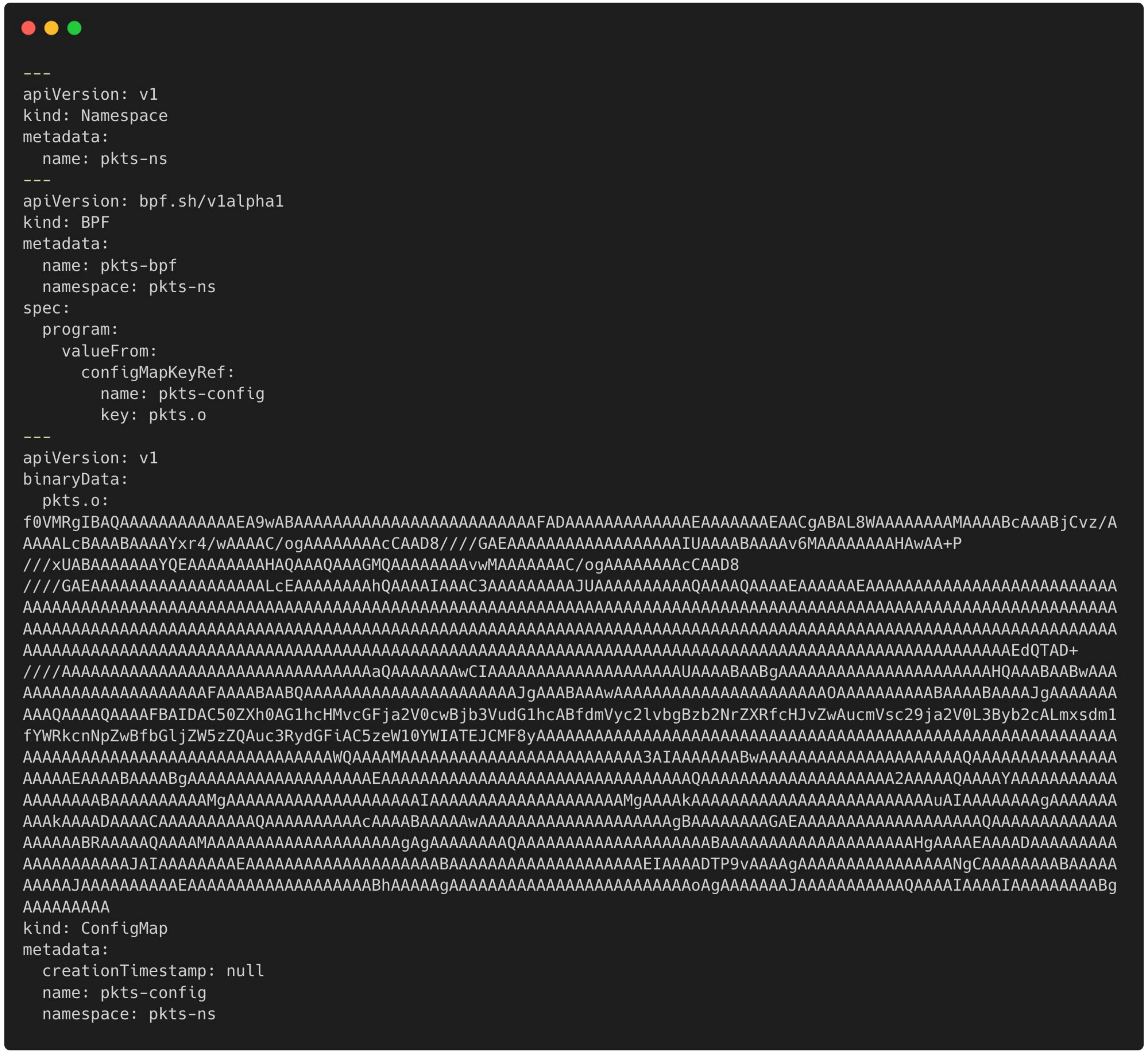
What about Kubernetes ?

Just use a container



```
apiVersion: v1
kind: Pod
metadata:
  name: happy-ebpf
spec:
  shareProcessNamespace: true
  containers:
  - name: execsnoop
    image: calavera/execsnoop # <-- the actual image containing the eBPF program
    securityContext:
    - privileged: true
    volumeMounts:
    - name: sys # mount the debug filesystem
      mountPath: /sys
      readOnly: true
    - name: headers # mount the kernel headers required by bcc
      mountPath: /usr/src
      readOnly: true
    - name: modules # mount the kernel modules required by bcc
      mountPath: /lib/modules
      readOnly: true
  - name: container doing random work
    image: yourcompany/yourapp # <-- your actual application
  ...
```

- 👏 A sidecar container sharing the process namespace
- 👏 You just provide an image with an eBPF loader and program in it
- 👏 Not extremely generic but does the job!
- 👏 A very flexible approach!



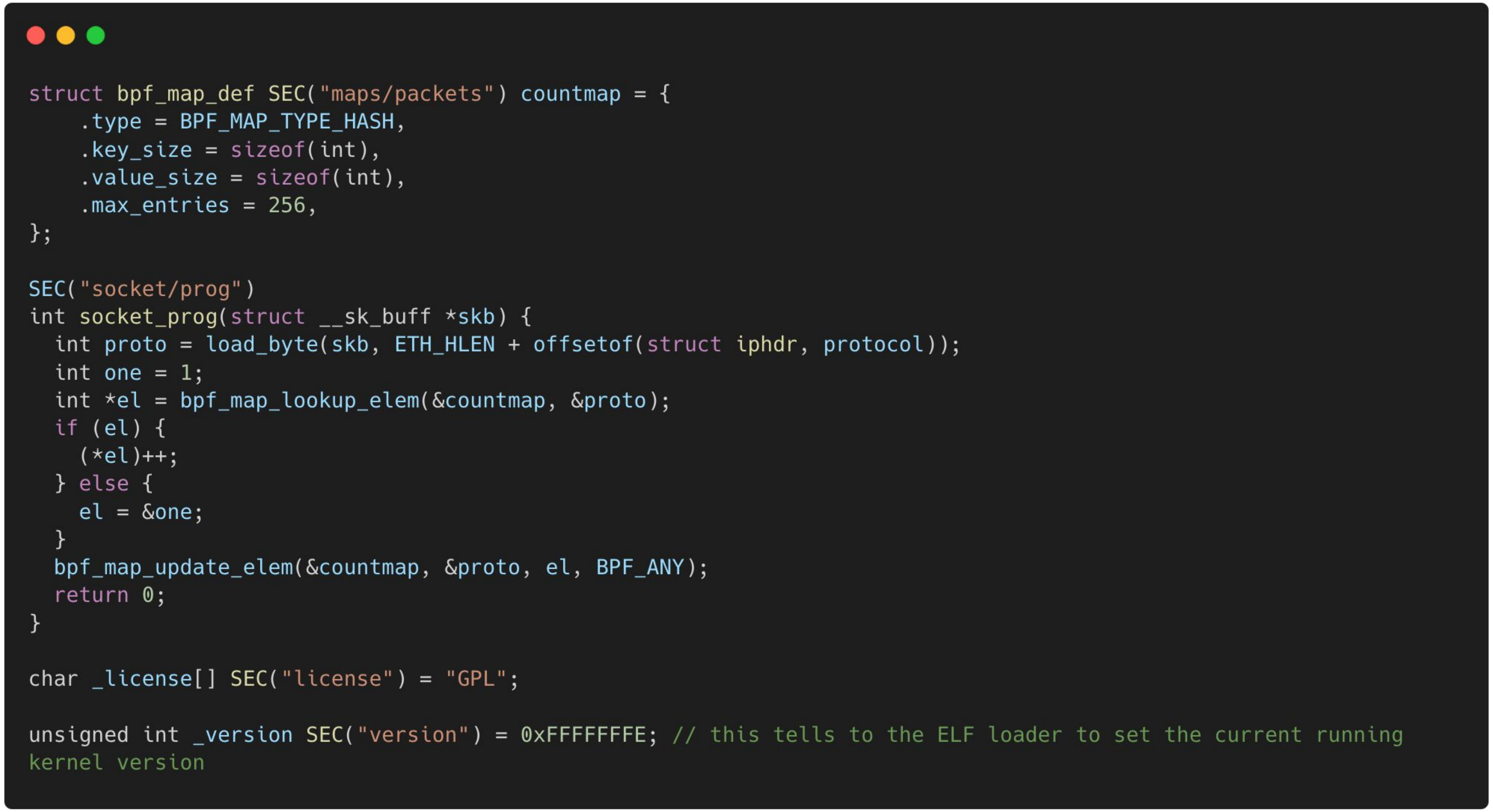
```
---
apiVersion: v1
kind: Namespace
metadata:
  name: pkts-ns
---
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
  namespace: pkts-ns
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
---
apiVersion: v1
binaryData:
  pkts.o:
```
f0VMRgIBAQAAAAAAAAAAAAAEA9wABAAAAAAAAAAAAAAAAAAAAAAAAAAFADAAAAAAAAAAAAAEAAAAAAAAEAACgABAL8WAAAAAAAAAMAAAAABcAAABjCvz/A
AAAALcBAAABAAAAYxr4/wAAAAC/ogAAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAAAIUAAAABAAAAv6MAAAAAAAAAHAwAA+P
///xUABAAAAAAAYQEAAAAAAAAAHAQAAAQAAAGMQAAAAAAAAAvwMAAAAAAAAC/ogAAAAAAAAAcCAAD8
////GAEAAAAAAAAAAAAAAAAAALcEAAAAAAAAAhQAAAAIAAAC3AAAAAAAAAJUAAAAAAAAAAAQAAAAQAAAAEAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEdQTAD+
////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAaQAAAAAAAAwCIAAAAAAAAAAAAAAAAAAAAAUAAAABAABgAAAAAAAAAAAAAAAAAAAAAAAAAHQAAABAABwAAA
AAAAAAAAAAAAAAAAAAFAAAABAABQAAAAAAAAAAAAAAAAAAAAAAJgAAABAAAwAAAAAAAAAAAAAAAAAAAAAAAAOAAAAAAAAAAABAAAABAAAAJgAAAAAAA
AAAQAAAAQAAAAFBAIDAC50ZXh0AG1hcHMvcGFja2V0cwBjb3VudG1hcABfdmVyc2lvbgBzb2NrZXRfcHJvZwAucmVsc29ja2V0L3Byb2cALmxsdm1
fYWRkcnNpZwBfbGljZW5zZQAuc3RydGFiAC5zeW10YWIATEJCMF8yAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAWQAAAAMAAAAAAAAAAAAAAAAAAAAAAAAA3AIAAAAAAAABwAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAA
AAAAAEAAAABAAAABgAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAA2AAAAAQAAAAYAAAAAAAAAAAA
AAAAAAAABAAAAAAAAAAMgAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAMgAAAAkAAAAAAAAAAAAAAAAAAAAAAAAAAAuAIAAAAAAAAgAAAAAAA
AAAkAAAADAAAACAAAAAAAAAAAQAAAAAAAAAAAcAAAABAAAAAwAAAAAAAAAAAAAAAAAAAAAgBAAAAAAAAAGAEAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAA
AAAAAABRAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAgAgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAHgAAAAEAAAADAAAAAAAAAAA
AAAAAAAAAAAAJAIAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAEIAAAADTP9vAAAAgAAAAAAAAAAAAAAAANgCAAAAAAAAABAAAAA
AAAAAJAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAABhAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAAoAgAAAAAAAJAAAAAAAAAAAAAQAAAAIAAAAIAAAAAAAAAAABg
AAAAAAAAA
```
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: pkts-config
  namespace: pkts-ns
```

@fntlnz

Want something more generic?

- 👏 Here's an experiment I've been working with @leodido
- 👏 It loads eBPF ELF objects using a CRD
- 👏 Same as the container example but you don't have to write the loader
- 👏 Exposes a Prometheus endpoint

YAML ENGINEERING
https://yaml.engineering



```c
struct bpf_map_def SEC("maps/packets") countmap = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(int),
    .max_entries = 256,
};

SEC("socket/prog")
int socket_prog(struct __sk_buff *skb) {
  int proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
  int one = 1;
  int *el = bpf_map_lookup_elem(&countmap, &proto);
  if (el) {
    (*el)++;
  } else {
    el = &one;
  }
  bpf_map_update_elem(&countmap, &proto, el, BPF_ANY);
  return 0;
}

char _license[] SEC("license") = "GPL";

unsigned int _version SEC("version") = 0xFFFFFFFE; // this tells to the ELF loader to set the current running
kernel version
```

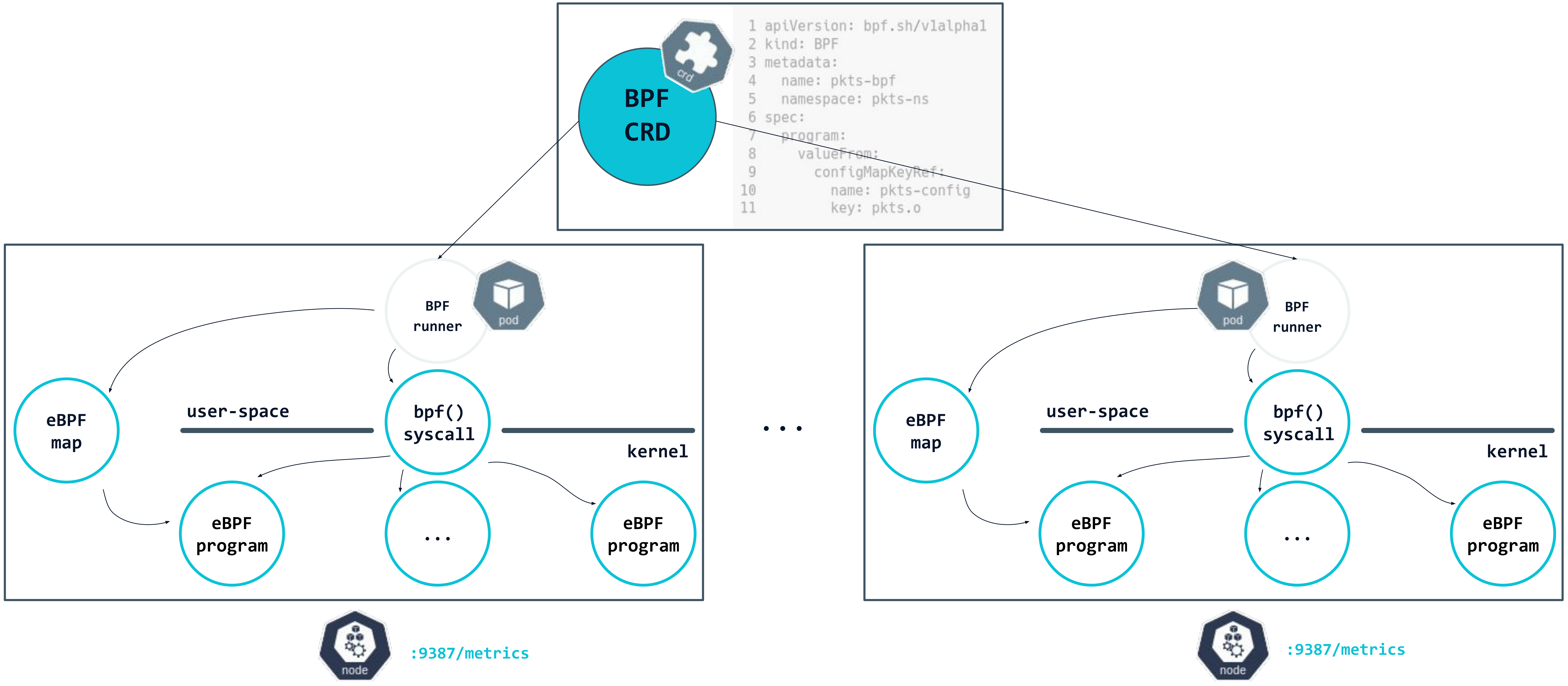
pkts.c

- 👏 Counts all the packets
- 👏 Uses a map to keep a counter
- 👏 It’s an HASH map so that it can assign the counter to a packet type

ip-10-12-0-136.ec2.internal:9387/metrics

```
# HELP test_packets No. of packets per protocol (key), node
# TYPE test_packets counter
test_packets{key="00001",node="127.0.0.1"} 8                  # <- ICMP
test_packets{key="00002",node="127.0.0.1"} 1                  # <- IGMP
test_packets{key="00006",node="127.0.0.1"} 551                # <- TCP
test_packets{key="00008",node="127.0.0.1"} 1                  # <- EGP
test_packets{key="00017",node="127.0.0.1"} 15930              # <- UDP
test_packets{key="00089",node="127.0.0.1"} 9                  # <- OSPF
test_packets{key="00233",node="127.0.0.1"} 1                  # <- ?
# EOF
```

Here's the evil plan



```yaml
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
  namespace: pkts-ns
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
```

Get the code!

github.com/bpftools/kube-bpf

eBPF tracing in the kubectl!

https://github.com/iovisor/kubectl-trace

kubectl-trace

The kubectl trace plugin

Your bpftrace program

```
kubectl trace run -e 'kprobe:do_sys_open { printf("%s,%s\n", comm, str(arg1))
}'  ip-180-12-0-220.ec2.internal -a
```

Attach the terminal to the program’s TTY

The node where to run it in your cluster

eBPF tracing in the kubectl!

https://github.com/iovisor/kubectl-trace

kubectl-trace

Run bpftrace program (from file)

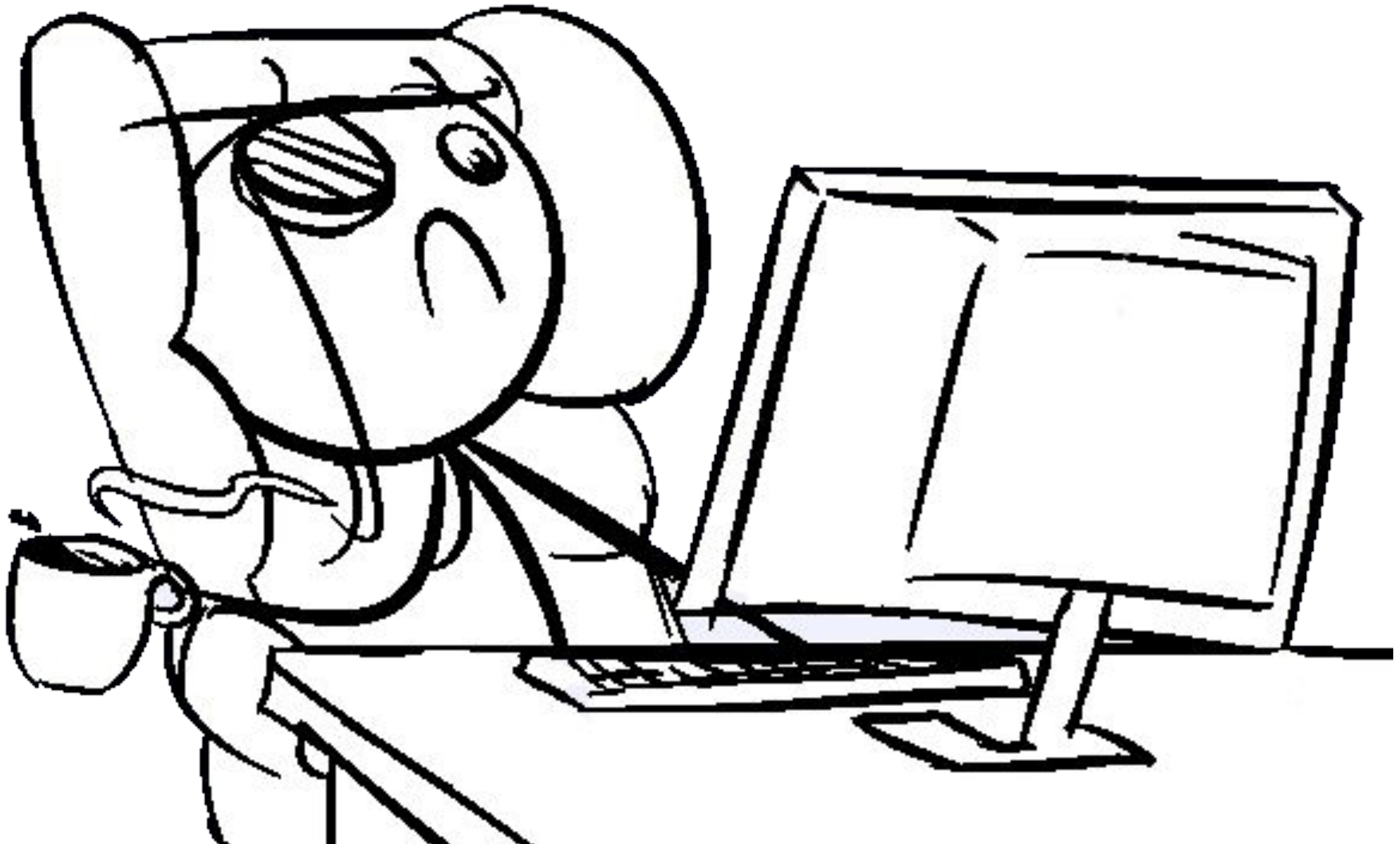
Ctrl-C tells the program to plot the results using hist()

Maps

The output histogram

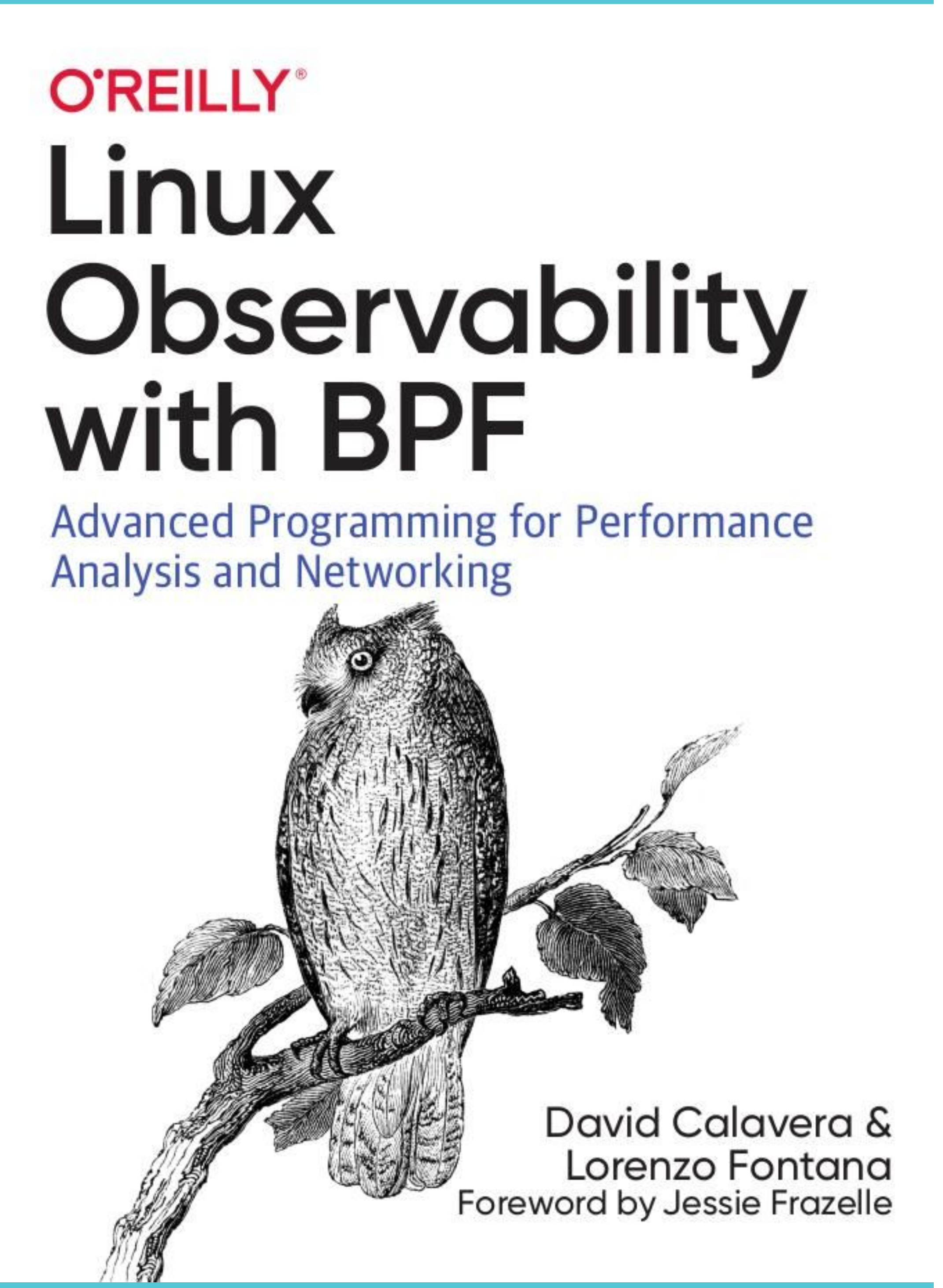
```
kubectl trace run 127.0.0.1 -f read.bt -a
trace 9df7388a-f0b4-11e8-ae05-8c164500a77e created
^C

@start[12509]: 49914871556264
@start[12856]: 49914833559762
@start[12865]: 49914847759523
@start[12866]: 49914848563942
@start[12867]: 49914872764939


@times:
[512, 1K)             85 |@@@@                                                |
[1K, 2K)             767 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@         |
[2K, 4K)             700 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             |
[4K, 8K)             920 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)            751 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          |
[16K, 32K)           393 |@@@@@@@@@@@@@@@@@@@@@@                              |
[32K, 64K)            90 |@@@@@                                               |
[64K, 128K)           14 |                                                    |
[128K, 256K)           3 |                                                    |
[256K, 512K)           4 |                                                    |
[512K, 1M)             2 |                                                    |
[1M, 2M)               2 |                                                    |
[2M, 4M)               2 |                                                    |
```

demo



Wait wait wait wait!

There’s a book!


O'REILLY
Linux Observability with BPF
Advanced Programming for Performance Analysis and Networking
David Calavera & Lorenzo Fontana
Foreword by Jessie Frazelle

- 👏 From me and David Calavera
- 👏 Almost published
- 👏 Preorder on Amazon.com, DO IT!
- 👏 Early Release on O’Reilly Safari
- 👏 Foreword by Jessie Frazelle

All the acronyms

Computer people loves acronyms

BPF:	Berkley Packet Filter
eBPF:	Extended Berkley Packet Filter
CRD:	Custom Resource Definition (Kubernetes)

Thanks. 👋

Reach me out @fntlnz on twitter & github!