Playing with the eBPF in-kernel virtual machine

Because you know you need to and also it's cool

CloudConf 2018
Lorenzo Fontana (@fntlnz)
SRE at InfluxData


eBPF or DIE

eBPF stands for

Extended Berkeley Packet Filter

BPF is a Tracing Framework*

Used to access kernel trace backend instrumentation tools

*Actually, it can also do packet mangling, forwarding and encapsulation.And there's also XDP.

sched:

signal:

irq:

tcp:

```
# ls /sys/kernel/debug/tracing/events/irq/
enable  filter  irq_handler_entry
irq_handler_exit  softirq_entry  softirq_exit
softirq_raise
```

kvm:

Static tracepoints

Look at:

```
# cat /sys/kernel/debug/tracing/available_events
```

workqueue:

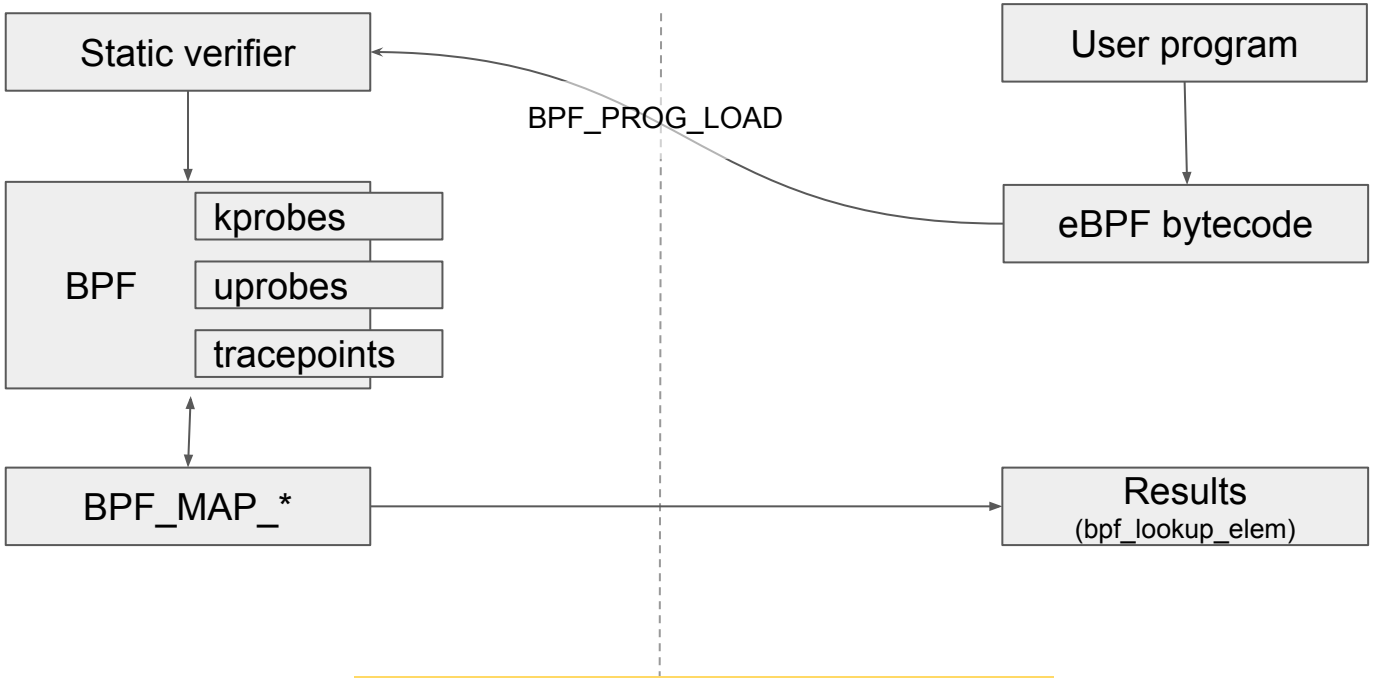
timer:

task:

Dynamic trace functionalities

uprobes, kprobes

Interactions using an eBPF



Kernel space

User pace

The mustache parrot warns!
eBPF programs can't be turing complete!

In today’s world

In today's world: tcpdump

`-d` stands for: Dump the compiled packet-matching code in a human readable form to standard output and stop.

```
# 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
```

Is it an ethernet IP IPv4 packet?

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

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

Documentation about the instruction set: https://www.kernel.org/doc/Documentation/networking/filter.txt

In today's world: seccomp

```c
#include <linux/seccomp.h>
#include <stdio.h>
#include <sys/prctl.h>
#include <unistd.h>

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

  prctl(PR_SET_SECCOMP, SECCOMP_MODE_STRICT);
  printf("something's gonna happen!!\n");

  dup2(1, 2);

  printf("sorry, can't get here\n");
  return 0;
}
```

```
gcc -lseccomp seccomp-test.c
```

```
./a.out
hey there!
something's gonna happen!!
[1]  19463 killed  ./a.out
```

In today's world: More practical examples?

- Trace file opens by filename
- Trace queries done against a database, like InfluxDB or MySQL
- Trace TCP retransmissions
- Trace all commands done in a bash shell
- Trace block device I/O latency over time
- Duplicate data flows
- JVM events
- Go Runtime Events
- Debuggers!
- Firewalls, packet rewriting, dropping etc..

Interesting projects

- Iovisor BCC https://www.iovisor.org/
- Cilium: HTTP, gRPC, and Kafka Aware Security and Networking for Containers with BPF and XDP https://github.com/cilium/cilium
- iovisor/gobpf https://github.com/iovisor/gobpf (do eBPF in Go)
- Landlock LSM https://landlock.io/ (like seccomp but for kernel objects instead of syscalls)

iovisor/BCC - trace tool

```
-t timestamp
-I additional header to include
<probe>
```

uprobe on tcp_set_state | Format | Arguments

```
# /usr/share/bcc/tools/trace -t -I net/sock.h 'p::tcp_set_state(struct sock *sk) "%llx: %d -> %d", sk, sk->sk_state, arg2'
TIME   PID    TID    COMM         FUNC                -
2.931834 8424    8432    Socket Thread tcp_set_state    ffff9cf258c77800: 7 -> 2
3.098617 477       477    irq/155-iwlwifi tcp_set_state ffff9cf258c77800: 2 -> 1
3.099123 8424    8432    Socket Thread tcp_set_state    ffff9cf258e8a000: 7 -> 2
3.183138 8424    8432    Socket Thread tcp_set_state    ffff9cf258e8d000: 7 -> 2
3.206508 477       477    irq/155-iwlwifi tcp_set_state ffff9cf258e8a000: 2 -> 1
3.299355 477       477    irq/155-iwlwifi tcp_set_state ffff9cf258e8d000: 2 -> 1
3.410099 8424    8432    Socket Thread tcp_set_state    ffff9cf258e8d000: 1 -> 4
3.416010 477       477    irq/155-iwlwifi tcp_set_state ffff9cf258e8d000: 4 -> 7
3.531687 8424    8432    Socket Thread tcp_set_state    ffff9cf27c7c4000: 7 -> 2
3.531753 8424    8432    Socket Thread tcp_set_state    ffff9cf27c7c6800: 7 -> 2
```

https://github.com/torvalds/linux/blob/e8fce23946b7e7eadf25ad78d8207c22903dfe27/include/trace/events/tcp.h#L180

See you