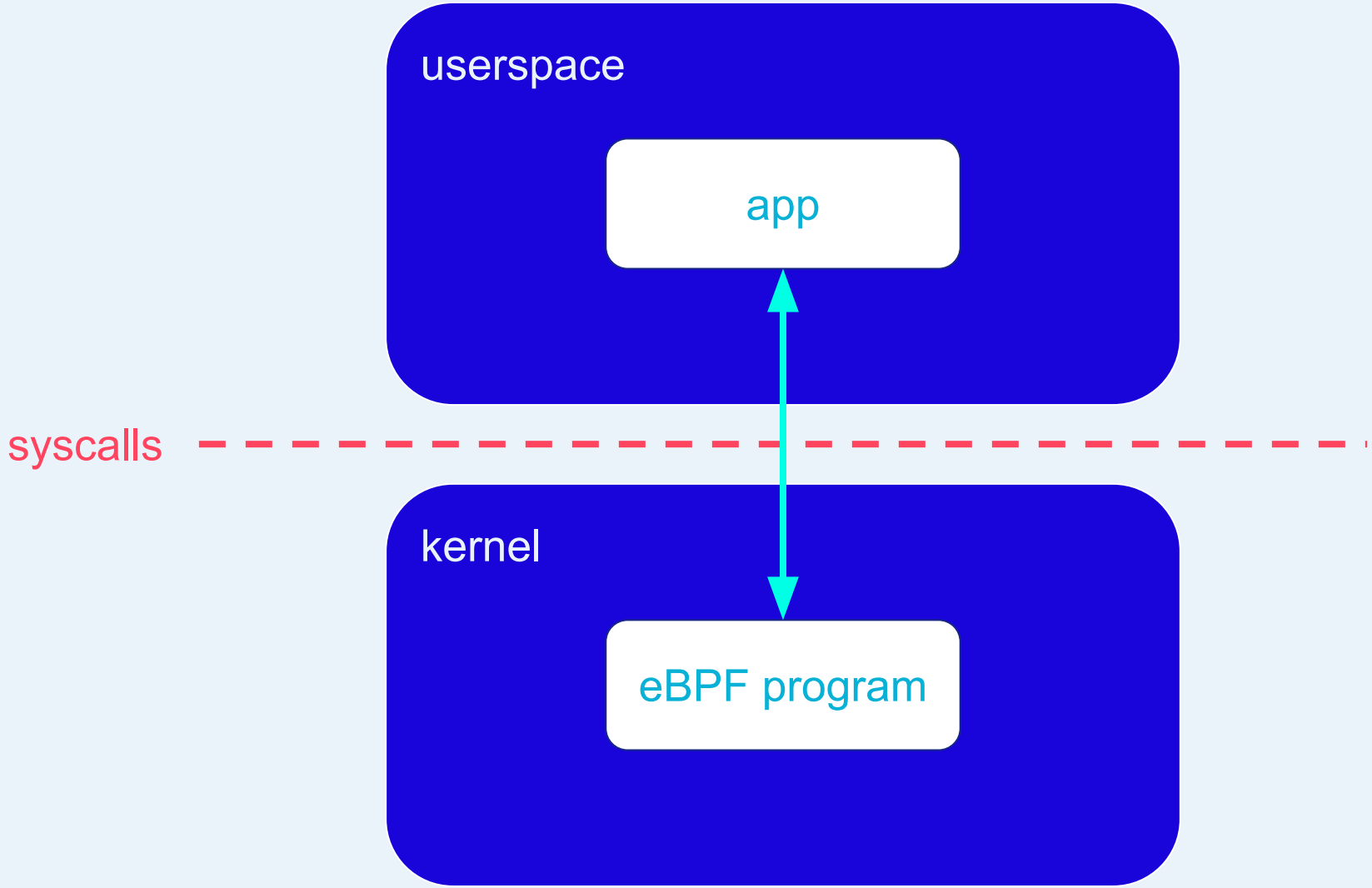
October 28, 2020

A Beginner's Guide to eBPF Programming

Liz Rice @lizrice

VP Open Source Engineering, Aqua Security

Run custom code in the kernel



aqua

man bpf

eBPF programs can be written in a restricted C that is compiled (using the clang compiler) into eBPF bytecode. Various features are omitted from this restricted C, such as loops, global variables, variadic functions, floating-point numbers, and passing structures as function arguments.

man bpf

eBPF programs can be written in a restricted C that is compiled (using the clang compiler) into eBPF bytecode. Various features are omitted from this restricted C, such as loops, global variables, variadic functions, floating-point numbers, and passing structures as function arguments.

[eBPF Helper functions] are used by eBPF programs to interact with the system, or with the context in which they work. For instance, they can be used to print debugging messages...

```
bpf_trace_printk()
bpf_get_current_uid_gid()
...
```

man bpf

Maps are a generic data structure for storage of different types of data. They allow sharing of data between eBPF kernel programs, and also between kernel and user-space applications.

man bpf

Maps are a generic data structure for storage of different types of data. They allow sharing of data between eBPF kernel programs, and also between kernel and user-space applications.

eBPF programs can be attached to different events.

bpftrace

bpftrace is a high-level tracing language for Linux enhanced Berkeley Packet Filter (eBPF) available in recent Linux kernels (4.x). bpftrace uses LLVM as a backend to compile scripts to BPF-bytecode and makes use of BCC for interacting with the Linux BPF system, as well as existing Linux tracing capabilities: kernel dynamic tracing (kprobes), user-level dynamic tracing (uprobes), and tracepoints. The bpftrace language is inspired by awk and C, and predecessor tracers such as DTrace and SystemTap. bpftrace was created by Alastair Robertson.

To learn more about bpftrace, see the Reference Guide and One-Liner Tutorial.

One-Liners

The following one-liners demonstrate different capabilities:

```
# Files opened by process
bpftrace -e 'tracepoint:syscalls:sys_enter_open { printf("%s %s\n", comm, str(args->filename)); }'

# Syscall count by program
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'

# Read bytes by process:
bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret/ { @[comm] = sum(args->ret); }'

# Read size distribution by process:
bpftrace -e 'tracepoint:syscalls:sys_exit_read { @[comm] = hist(args->ret); }'

# Show per-second syscall rates:
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @ = count(); } interval:s:1 { print(@); clear(@);
```

Explore bpf syscalls in bpftrace

eBPF programs & maps

```
bpf(BPF_PROG_LOAD, …)
bpf(BPF_MAP_CREATE, …)
```

Attach custom code to an event

```
bpf(BPF_PROG_LOAD, …) = x
perf_event_open(…) = y
ioctl(y, PERF_EVENT_IOC_SET_BPF, x)
```

eBPF hello world

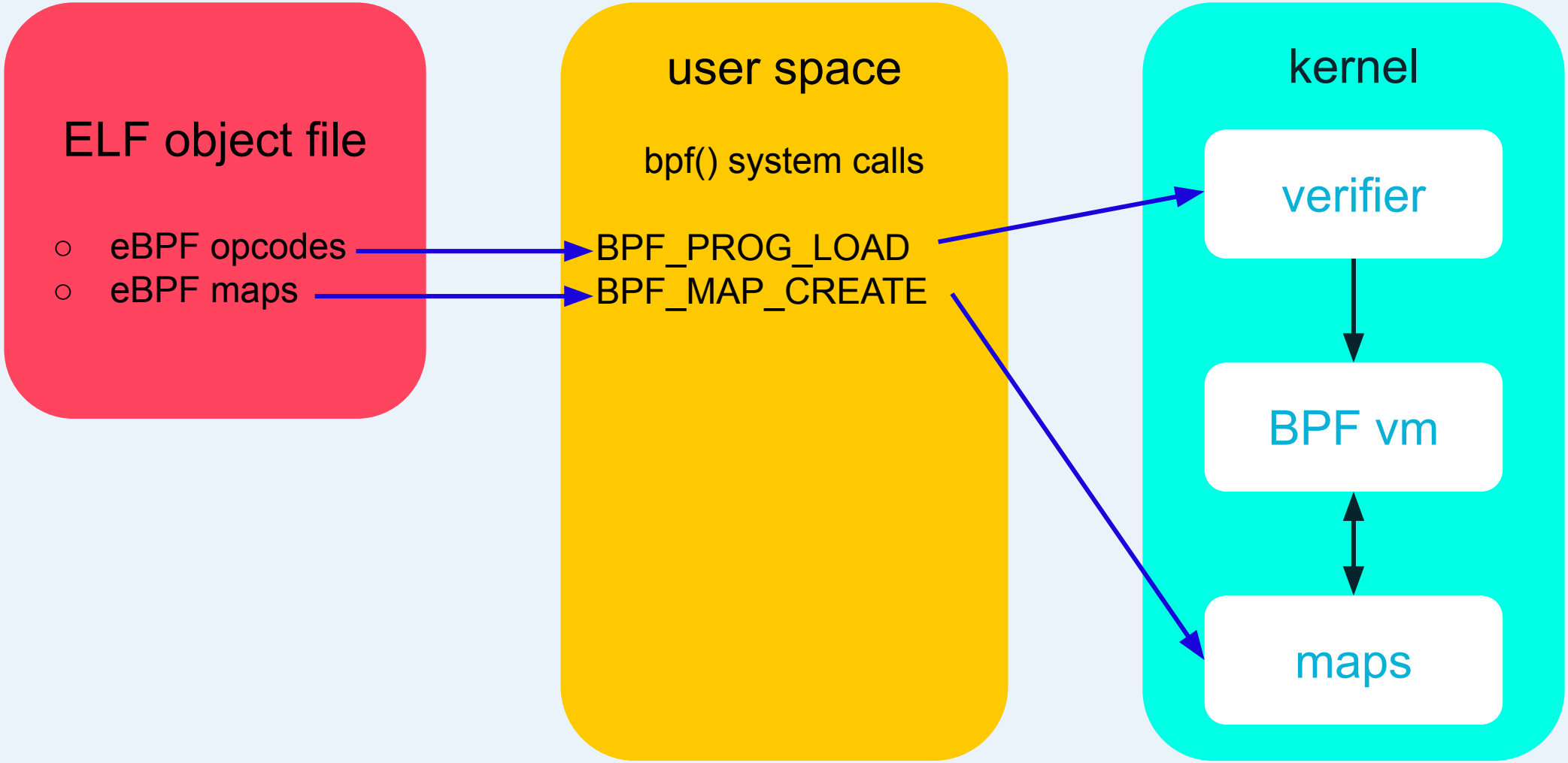
```
#!/usr/bin/python
from bcc import BPF

prog = """
int helloworld(void *ctx) {
   bpf_trace_printk("Hello world\\n");
   return 0;
}
"""

b = BPF(text=prog)
clone = b.get_syscall_fnname("clone")
b.attach_kprobe(event=clone, fn_name="helloworld")

b.trace_print()
```

```
#!/usr/bin/python
from bcc import BPF
from time import sleep

prog = """
BPF_HASH(clones);

int hello_world(void *ctx) {
  u64 uid;
  u64 counter = 0;
  u64 *p;
  uid = bpf_get_current_uid_gid() & 0xFFFFFFFF;
  p = clones.lookup(&uid);
  if (p != 0) {
     counter = *p;
  }
  counter++;
  clones.update(&uid, &counter);
  return 0;
}
"""
```

```
b = BPF(text=prog)
clone = b.get_syscall_fnname("clone")
b.attach_kprobe(event=clone, fn_name="helloworld")

while True:
  sleep(2)
  s = ""
  if len(b["clones"].items()):
     for k,v in b["clones"].items():
        s += "ID {}: {}\t".format(k.value, v.value)
     print(s)
  else:
     print("No entries yet")
```

@lizrice

github.com/lizrice/ebpf-beginners


ELF object file
eBPF opcodes
eBPF maps
user space
bpf() system calls
BPF_PROG_LOAD
BPF_MAP_CREATE
kernel
verifier
BPF vm
maps

ELF object file

- eBPF opcodes
- eBPF maps

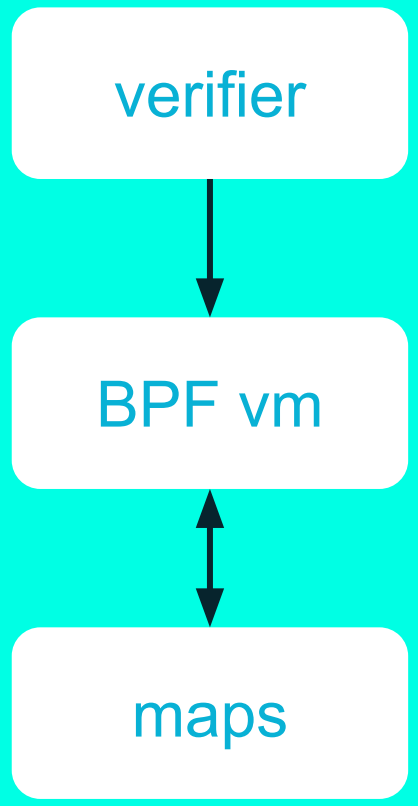
user space

bpf() system calls

BPF_PROG_LOAD
BPF_MAP_CREATE

Attach BPF program to event

kernel



aqua

ELF object file

- eBPF opcodes
- eBPF maps

user space

bpf() system calls

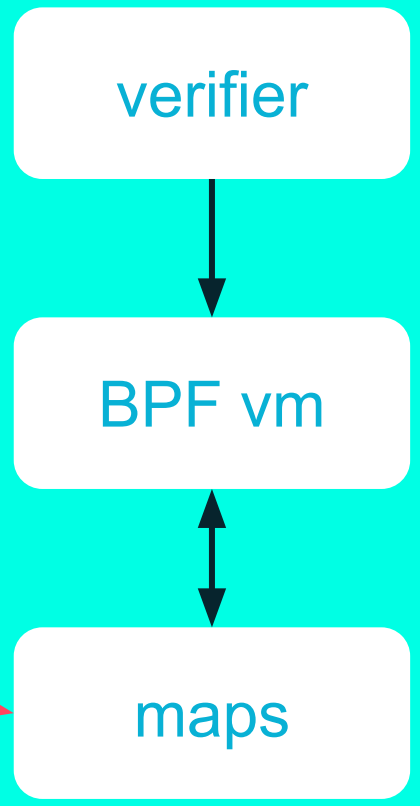
BPF_PROG_LOAD
BPF_MAP_CREATE

Attach BPF program to event

Read / write maps

BPF_MAP_GET_NEXT_KEY
BPF_MAP_LOOKUP_ELEM
BPF_MAP_UPDATE_ELEM
BPF_MAP_DELETE_ELEM

kernel



aqua

Thank you

github.com/lizrice/ebpf-beginners

@lizrice