Trace Aggregation and Collection with eBPF
Suchakrapani Sharma
Agenda

Introduction
- Quick eBPF Intro
- Internals of eBPF

Use cases
- Networking, Tracing, Security
- IOVisor BPF Compiler Collection
- Tracing Examples

Trace Collection
- eBPF to CTF

What’s Next
eBPF

Stateful, programmable, in-kernel decisions for networking, tracing and security

"One Ring" by Yukatan (CC)
Berkeley Packet Filter

Classical BPF (cBPF)
- Network packet filtering \cite{McCane1993}, Seccomp
- Filter Expressions $\rightarrow$ Bytecode $\rightarrow$ Interpret$^*$
- Small, in-kernel VM. Register based, switch dispatch interpreter, few instructions

Extended BPF (eBPF) \cite{Sharma1993} \cite{Clément1993}
- More registers, JIT compiler (flexible/faster), verifier
- Attach on Tracepoint/Kprobe/Uprobe/USDT
- In-kernel trace aggregation & filtering
- Control via \texttt{bpf()}, trace collection via BPF Maps
- Upstream in Linux Kernel (\texttt{bpf()} syscall, v3.18+)
- Bytecode compilation upstream in LLVM/Clang

$^*$JIT support eventually landed in kernel
Berkeley Packet Filter

Program Anatomy

- BPF Program
  - prog.bpf
  - LLVM/Clang
  - Bytecode

- Reader
  - bpf()
  - BPF Maps
  - Kernel Functions

- eBPF
  - bpf()
  - BPF Bytecode
  - Verifier + JIT
  - Native Code

Userspace

Suchakrapani Datt Sharma
eBPF for Networking

Traffic Control/XDP
- TC with cls_bpf \cite{Borkmann, 2016} act_bpf and XDP

Adapted from Thomas Graf's presentation “Cilium - BPF & XDP for containers”
eBPF for Security

LSM Hooks

BPF Program

- policy.bpf
- LLVM/Clang
- bpf()

Syscalls

Verifier + JIT

BPF Code

LSM Hook

Application

Application

Userspace

Kernel

EACCESS

Suchakrapani Datt Sharma
eBPF for Tracing

Kprobes/Kretprobes

BPF Program

trace.bpf

LLVM/Clang

bpf()

Verifier + JIT

BPF Code

Kprobe

Kernel Function

BPF Map

Trace Pipe

Perf Buffer

Monitor/Store

Read Events

Read/Update

Userspace

Kernel
eBPF Features & Support

Major BPF Milestones by Kernel Version*

- 3.18 : bpf() syscall
- 3.19 : Sockets support, BPF Maps
- 4.1  : Kprobe support
- 4.4  : Perf events
- 4.6  : Stack traces, per-CPU Maps
- 4.7  : Attach on Tracepoints
- 4.8  : XDP core and act
- 4.9  : Profiling, attach to Perf events
- 4.10 : cgroups support (socket filters)
- 4.11 : Tracerception – tracepoints for eBPF debugging

*Adapted from “BPF: Tracing and More” by Brendan Gregg (Linux.Conf.au 2017)
Program Types

- BPF_PROG_TYPE_UNSPEC
- BPF_PROG_TYPE_SOCKET_FILTER
- BPF_PROG_TYPE_KPROBE
- BPF_PROG_TYPE_SCHED_CLS
- BPF_PROG_TYPE_SCHED_ACT
- BPF_PROG_TYPE_TRACEPOINT
- BPF_PROG_TYPE_XDP
- BPF_PROG_TYPE_PERF_EVENT
- BPF_PROG_TYPE_CGROUP_SKB
- BPF_PROG_TYPE_CGROUP_SOCK
- BPF_PROG_TYPE_LWT_IN
- BPF_PROG_TYPE_LWT_OUT
- BPF_PROG_TYPE_LWT_XMIT
- BPF_PROG_TYPE_LANDLOCK

http://lxr.free-electrons.com/source/include/uapi/linux/bpf.h
eBPF Features & Support

Map Types

- BPF_MAP_TYPE_UNSPEC
- BPF_MAP_TYPE_HASH
- BPF_MAP_TYPE_ARRAY
- BPF_MAP_TYPE_PROG_ARRAY
- BPF_MAP_TYPE_PERF_EVENT_ARRAY
- BPF_MAP_TYPE_PERCPU_HASH
- BPF_MAP_TYPE_PERCPU_ARRAY
- BPF_MAP_TYPE_STACK_TRACE
- BPF_MAP_TYPE_CGROUP_ARRAY
- BPF_MAP_TYPE_LRU_HASH
- BPF_MAP_TYPE_LRU_PERCPU_HASH

http://lxr.free-electrons.com/source/include/uapi/linux/bpf.h
eBPF for Tracing

Frontends

- IOVisor BCC – Python, C++, Lua, Go (gobpf) APIs
- Compile BPF programs directly via LLVM interface
- Helper functions to manage maps, buffers, probes

Kprobes Example

```python
from bcc import BPF

prog = ""
int hello(void *ctx) {
    bpf_trace_printk("Hello, World!\n");
    return 0;
}
"

b = BPF(text=prog)
b.attach_kprobe(event="sys_clone", fn_name="hello")
print "PID MESSAGE"
b.trace_print(fmt="{1} {5}"")
```

Complete Program trace_fields.py

prog compiled to BPF bytecode

Attach to Kprobe event

Print trace pipe
eBPF for Tracing

Tracepoint Example (v4.7+)

```
# define EXIT_REASON 18

prog = """
TRACEPOINT_PROBE(kvm, kvm_exit) {
    if (args->exit_reason == EXIT_REASON) {
        bpf_trace_printk("KVM_EXIT exit_reason : %d\n", args->exit_reason);
    }
    return 0;
}

TRACEPOINT_PROBE(kvm, kvm_entry) {
    if (args->vcpu_id = 0) {
        bpf_trace_printk("KVM_ENTRY vcpu_id : %u\n", args->vcpu_id);
    }
}
"""

Output

# ./kvm-test.py
2445.577129000     CPU 0/KVM        8896   KVM_ENTRY vcpu_id : 0
2445.577136000     CPU 0/KVM        8896   KVM_EXIT exit_reason : 18
eBPF for Tracing

Uprobes Example

bpf_text = ""
#include <uapi/linux/ptrace.h>
#include <uapi/linux/limits.h>

int get_fname(struct pt_regs *ctx) {
  if (!ctx->si)
    return 0;
  char str[NAME_MAX] = {};
  bpf_probe_read(&str, sizeof(str), (void *)ctx->si);
  bpf_trace_printk("%s
", &str);
  return 0;
};

b = BPF(text=bpf_text)
b.attach_uprobe(name="/usr/bin/vim", sym="readfile", fn_name="get_fnum")

# ./vim-test.py
TASK   PID    FILENAME
vim     23707  /tmp/wololo
# eBPF for Tracing

## USDT Example

```python
going_to_local_variable
from bcc import BPF, USDT
.
.bpf_text = ""
#include <uapi/linux/ptrace.h>
int do_trace(struct pt_regs *ctx) {
    uint64_t addr;
    char path[128]={0};
    bpf_usdt_readarg(6, ctx, &addr);
    bpf_probe_read(&path, sizeof(path), (void *)addr);
    bpf_trace_printk("path:%s\n", path);
return 0;
}
"""

u = USDT(pid=int(pid))
u.enable_probe(probe="http_server_request", fn_name="do_trace")
b = BPF(text=bpf_text, usdt_contexts=[u])
```
eBPF for Tracing

**USD T Example**

```
# ./nodejs_http_server.py 24728
TIME(s)            COMM             PID    ARGS
24653324.561322998 node             24728  path:/index.html
24653335.343401998 node             24728  path:/images/welcome.png
24653340.510164998 node             24728  path:/images/favicon.png
```

**Supported Frameworks**

- MySQL :  --enable-dtrace (Build)
- JVM     :  -XX:+ExtendedDTraceProbes (Runtime)
- Node    :  --with-dtrace (Build)
- Python  :  --with-dtrace (Build)
- Ruby    :  --enable-dtrace (Build)
eBPF for Tracing

BPF Maps – Filters, States, Counters

```c
bpf_text = """ #include <uapi/linux/ptrace.h>
#include <net/sock.h>
#include <bcc/proto.h>

BPF_HASH(currsock, u32, struct sock *);

int kprobe__tcp_v4_connect(struct pt_regs *ctx, struct sock *sk) {
  u32 pid = bpf_get_current_pid_tgid();
  // stash the sock ptr for lookup on return
  currsock.update(&pid, &sk);
  return 0;
}
```

Program Excerpt
tcpv4connect.py

- **Key**
- **Value type**
- **Update hash map**
int kretprobe__tcp_v4_connect(struct pt_regs *ctx)
{
    int ret = PT_REGS_RC(ctx);
    u32 pid = bpf_get_current_pid_tgid();
    struct sock **skpp;
    skpp = currsock.lookup(&pid);
    if (skpp == 0) {
        return 0; // missed entry
    }
    if (ret != 0) {
        // failed to send SYNC packet, may not have populated
        currsock.delete(&pid);
        return 0;
    }

    struct sock *skp = *skpp;
    u32 saddr = 0, daddr = 0;
    u16 dport = 0;
    bpf_probe_read(&saddr, sizeof(saddr), &skp->__sk_common.skc_rcv_saddr);
    bpf_probe_read(&daddr, sizeof(daddr), &skp->__sk_common.skc_daddr);
    bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.skc_dport);
    bpf_trace_printk("trace_tcp4connect %x %x %d\n", saddr, daddr, ntohs(dport));
    currsock.delete(&pid);
    return 0;
}
eBPF for Tracing

BPF Maps – Filters, States, Counters

```
# ./tcpv4connect.py
PID    COMM         SADDR            DADDR            DPORT
1479   telnet       127.0.0.1        127.0.0.1        23
1469   curl         10.201.219.236   54.245.105.25    80
1469   curl         10.201.219.236   54.67.101.145     80
```

More Uses

- Record latency ($\Delta t$)
  - biosnoop.py
- Flags for keeping track of events
  - kvm_hypercall.py
- Counting events, histograms
  - cachestat.py
  - cpudist.py
eBPF for Tracing

BPF Perf Event Output

- Build perf events and save to per-cpu perf buffers

```
prog = ""
#include <linux/sched.h>
#include <uapi/linux/ptrace.h>
#include <uapi/linux/limits.h>

struct data_t {
    u32 pid;
    u64 ts;
    char comm[TASK_COMM_LEN];
    char fname[NAME_MAX];
};

BPF_PERF_OUTPUT(events);

int handler(struct pt_regs *ctx) {
    struct data_t data = {};
    data.pid = bpf_get_current_pid_tgid();
    data.ts = bpf_ktime_get_ns();
    bpf_get_current_comm(&data.comm, sizeof(data.comm));
    bpf_probe_read(&data.fname, sizeof(data.fname),
        (void *)PT_REGS_PARM1(ctx));

    events.perf_submit(ctx, &data, sizeof(data));
    return 0;
}
```

Program Excerpt
eBPF Trace Visualization

Current State
- Using ASCII histograms, ASCII escape codes
- eBPF trace driven Flamegraphs

```
# ./argdist -H 'p:c:write(int fd, void *buf, size_t len):size_t:len:fd==1'
[01:47:19]
p:c:write(int fd, void *buf, size_t len):size_t:len:fd==1
len : count distribution
  0 -> 1 : 0 |
  2 -> 3 : 0 |
  4 -> 7 : 0 |
  8 -> 15 : 3 |**********
 16 -> 31 : 0 |
 32 -> 63 : 5 |*****************
 64 -> 127 : 13 |**************************
```
eBPF Trace Visualization

Current State
- Using ASCII histograms, ASCII escape codes
- eBPF Flamegraphs, some web-based views
eBPF Trace Visualization

What We Need

- Modern visualizations, trace analysis, flame charts
- Data driven views, packaged with eBPF tools

Ceph traces from Mohamad Gebai (@mogeb88)
eBPF Trace Collection

Why collect traces?
- eBPF aggregates traces, no real trace storage
- Complement the live/snapshot usecase
- Fulfil long term analysis needs
- **Trace Compass** is a powerful visualization tool, we need to leverage its power!

Common Trace Format (CTF)
- Compact, binary format to save and store traces
- Very fast to write and read
- Well documented, stable, field-tested and used in industry-standard tools such as LTTng
- Easy to define trace streams and events
- Trace Compass supports CTF
eBPF Trace Collection

eBPF to CTF

- Currently uses libbabeltrace 2.0.0-pre Python APIs
- Just a PoC for now, APIs will change for sure

```python
from bcc import BPF, CTF, CTFEvent
import ctypes as ct

fields = {"pid": CTF.Type.u32, "comm": CTF.Type.string,
          "filename": CTF.Type.string}
c = CTF("sys_open", "/tmp/opentrace", fields)

def write_event(cpu, data, size):
    event = ct.cast(data, ct.POINTER(Data)).contents
    ev = CTFEvent(c)
    ev.time(c, int(event.ts))
    ev.payload('pid', event.pid)
    ev.payload('comm', event.comm.decode())
    ev.payload('filename', event.fname.decode())
    ev.write(c, cpu)

b["events"].open_perf_buffer(write_event)
while 1:
    b.kprobe_poll()
```

https://github.com/iovisor/bcc/tree/ctf/examples/tracing/ctf
eBPF Trace Collection

eBPF to CTF

$ babeltrace /tmp/opentrace
[11:32:19.482715248] (+0.000068367) 0 sys_open: { },
{ comm = "java", filename = "/proc/self/stat", pid = 10912 }
[11:32:19.514412607] (+0.031697359) 0 sys_open: { },
{ comm = "iio-sensor-prox", filename = "/dev/iio:device1", pid = 904 }
[11:32:19.514569626] (+0.000157019) 0 sys_open: { },
{ comm = "iio-sensor-prox", filename = "/dev/iio:device2", pid = 904 }
eBPF Trace Collection

eBPF to CTF

$ babeltrace /tmp/opentrace
[11:32:19.482715248] (+0.000068367) 0 sys_open: { },
{ comm = "java", filename = "/proc/self/stat", pid = 10912 }
[11:32:19.514412607] (+0.031697359) 0 sys_open: { },
{ comm = "iio-sensor-prox", filename = "/dev/iio:device1", pid = 904 }
[11:32:19.514569626] (+0.000157019) 0 sys_open: { },
{ comm = "iio-sensor-prox", filename = "/dev/iio:device2", pid = 904 }

It's Something...
What’s Next

VM Analysis
- BCC tool to monitor and analyze VMs
- Currently supports vCPU usage report only

Trace Storage & Display
- Use Babeltrace directly or BareCTF to generate custom trace writing code
- Explore if we can package analysis/views and trace data together
- Other trace formats for storage/display (Catapult)
References

Papers


References

Links

- IOVisor BPF Docs
- bcc Reference Guide
- bcc Python Developer Tutorial
- bcc/BPF Blog Posts
- Dive into BPF: a list of reading material (Quentin Monnet)
- Cilium - Network and Application Security with BPF and XDP (Thomas Graf)
- Landlock LSM Docs (Mickaël Salaün et al.)
- XDP for the Rest of Us (Jesper Brouer & Andy Gospodarek, Netdev 2.1)
- USDT/BPF Tracing Tools (Sasha Goldshtein)
- Linux 4.x Tracing : Performance Analysis with bcc/BPF (Brendan Gregg, SCALE 15X)
- The Common Trace Format (EfficiOS/Diamon Workgroup)
- babeltrace Library (EfficiOS/Diamon Workgroup)
- Trace Compass
- BPF/bcc for Oracle Tracing
- Weaveworks Scope HTTP Statistics Plugin
Ack

EfficiOS
Ericsson
DORSAL Lab, Polytechnique Montréal
IOVisor Project
LTTng Project
Eclipse Trace Compass Project
Fin!

suchakrapani.sharma@polymtl.ca
@tuxology

All the text and images in this presentation drawn by the authors are released under CC-BY-SA. Images not drawn by authors have been attributed either on slides or in references.