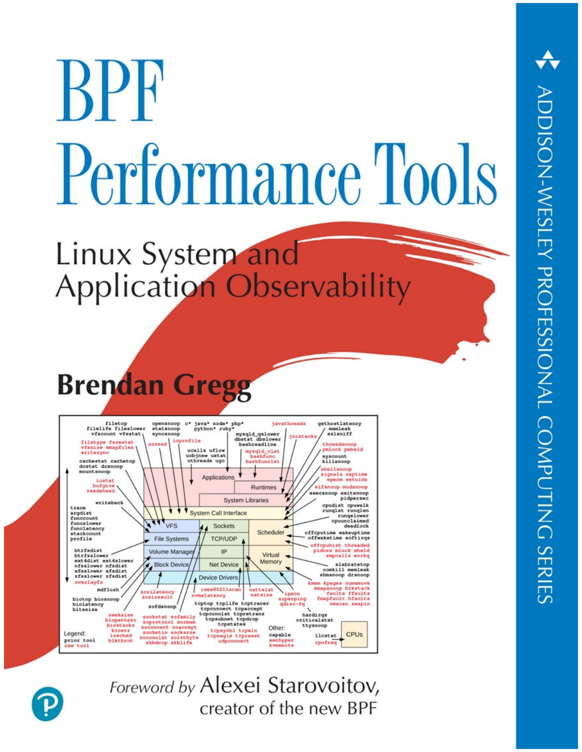
ebpf

jiri olsa

BPF Performance Tools by Brendan Greg

http://www.brendangregg.com/bpf-performance-tools-book.html

http://www.brendangregg.com/ebpf.html



NEXT BEST SOURCE

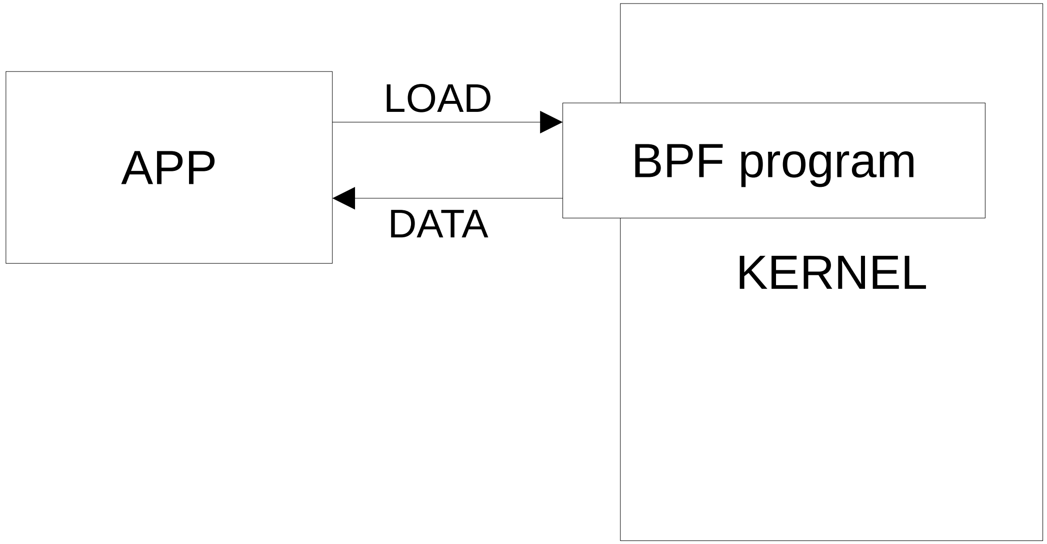
- kernel sources ;-)
- https://lore.kernel.org/bpf/

IN NUTSHELL..

Extended Berkeley Packet Filter

load code in kernel and execute it

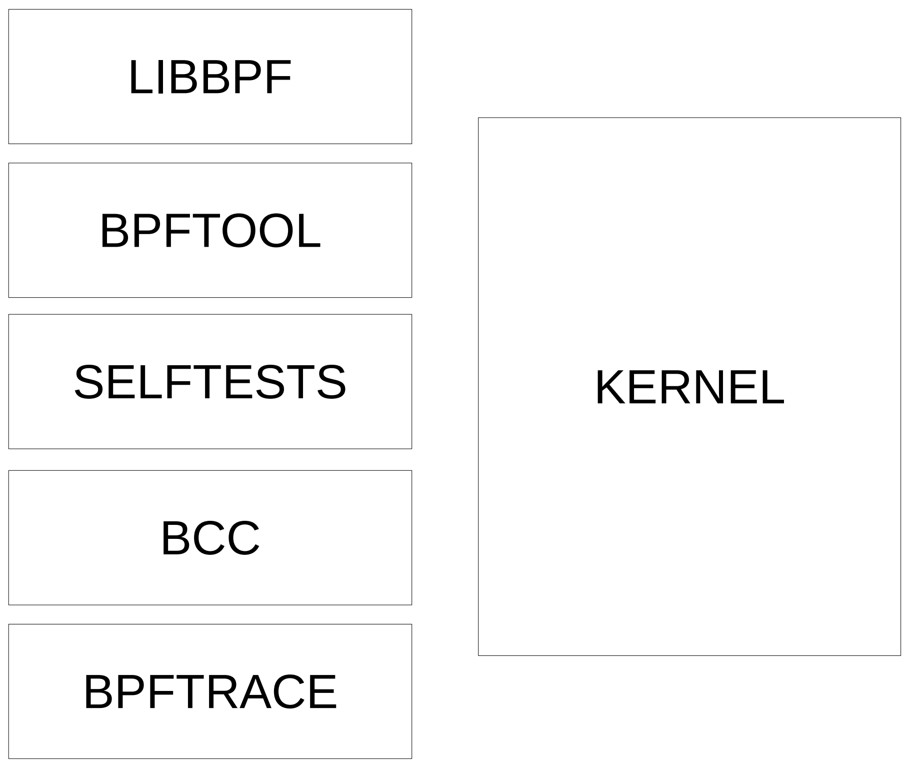
gather/send data to user



IN NUTSHELL..

tracing/networking

kernel subsystem + tools



BPF PROGRAM

- virtual machine
- verifier
- kernel JIT

```
BPF_PROG_TYPE_KPROBE
BPF_PROG_TYPE_TRACEPOINT
BPF_PROG_TYPE_PERF_EVENT
BPF_PROG_TYPE_RAW_TRACEPOINT
BPF_PROG_TYPE_RAW_TRACEPOINT_WRITABLE
BPF_PROG_TYPE_TRACING
BPF_PROG_TYPE_SOCKET_FILTER
BPF_PROG_TYPE_SCHED_CLS
BPF_PROG_TYPE_SCHED_ACT
BPF_PROG_TYPE_XDP
BPF_PROG_TYPE_CGROUP_SKB
BPF_PROG_TYPE_CGROUP_SOCK
BPF_PROG_TYPE_CGROUP_SOCK_ADDR
BPF_PROG_TYPE_LWT_IN
BPF_PROG_TYPE_LWT_OUT
BPF_PROG_TYPE_LWT_XMIT
BPF_PROG_TYPE_LWT_SEG6LOCAL
BPF_PROG_TYPE_SOCK_OPS
BPF_PROG_TYPE_SK_SKB
BPF_PROG_TYPE_SK_MSG
BPF_PROG_TYPE_FLOW_DISSECTOR
BPF_PROG_TYPE_CGROUP_DEVICE
BPF_PROG_TYPE_CGROUP_SYSCTL
BPF_PROG_TYPE_CGROUP_SOCKOPT
BPF_PROG_TYPE_LIRC_MODE2
BPF_PROG_TYPE_SK_REUSEPORT
BPF_PROG_TYPE_SK_LOOKUP
BPF_PROG_TYPE_STRUCT_OPS
BPF_PROG_TYPE_EXT
BPF_PROG_TYPE_LSM
```

VM ENVIRONMENT

```
r<1-10>     registers

r1 - r5     func arguments

r0          return value

r10         stack on entry

r1          context on entry

call func   helper call

map[id:X]   map descriptor
```

```
 0: (b7) r1 = 0
 1: (7b) *(u64 *)(r10 -16) = r1
 2: (b7) r1 = 1
 3: (7b) *(u64 *)(r10 -8) = r1
 4: (18) r1 = map[id:3]
 6: (bf) r2 = r10
 7: (07) r2 += -16
 8: (bf) r3 = r10
 9: (07) r3 += -8
10: (b7) r4 = 0
11: (85) call htab_map_update_elem
...
```

HELPERS

```
map_lookup_elem
map_update_elem
map_delete_elem
probe_read
ktime_get_ns
trace_printk
get_prandom_u32
get_smp_processor_id
skb_store_bytes
l3_csum_replace
l4_csum_replace
tail_call
clone_redirect
get_current_pid_tgid
get_current_uid_gid
get_current_comm
get_cgroup_classid
skb_vlan_push
skb_vlan_pop
skb_get_tunnel_key
skb_set_tunnel_key
perf_event_read
redirect
get_route_realm
perf_event_output
skb_load_bytes
get_stackid
csum_diff
skb_get_tunnel_opt
skb_set_tunnel_opt
skb_change_proto
skb_change_type
skb_under_cgroup
get_hash_recalc
get_current_task
probe_write_user
current_task_under_cgroup
skb_change_tail
this_cpu_ptr
redirect_peer
task_storage_get
skb_pull_data
csum_update
set_hash_invalid
get_numa_node_id
skb_change_head
xdp_adjust_head
probe_read_str
get_socket_cookie
get_socket_uid
set_hash
setsockopt
skb_adjust_room
redirect_map
sk_redirect_map
sock_map_update
xdp_adjust_meta
perf_event_read_value
perf_prog_read_value
getsockopt
override_return
sock_ops_cb_flags_set
msg_redirect_map
msg_apply_bytes
msg_cork_bytes
msg_pull_data
bind
xdp_adjust_tail
skb_get_xfrm_state
get_stack
skb_load_bytes_relative
fib_lookup
sock_hash_update
msg_redirect_hash
sk_redirect_hash
lwt_push_encap
lwt_seg6_store_bytes
lwt_seg6_adjust_srh
lwt_seg6_action
task_storage_delete
get_current_task_btf
bprm_opts_set
rc_repeat
rc_keydown
skb_cgroup_id
get_current_cgroup_id
get_local_storage
sk_select_reuseport
skb_ancestor_cgroup_id
sk_lookup_tcp
sk_lookup_udp
sk_release
map_push_elem
map_pop_elem
map_peek_elem
msg_push_data
msg_pop_data
rc_pointer_rel
spin_lock
spin_unlock
sk_fullsock
tcp_sock
skb_ecn_set_ce
get_listener_sock
skc_lookup_tcp
tcp_check_syncookie
sysctl_get_name
sysctl_get_current_value
sysctl_get_new_value
sysctl_set_new_value
strtol
strtoul
sk_storage_get
sk_storage_delete
send_signal
tcp_gen_syncookie
skb_output
probe_read_user
probe_read_kernel
probe_read_user_str
ktime_get_coarse_ns
ima_inode_hash
sock_from_file
probe_read_kernel_str
tcp_send_ack
send_signal_thread
jiffies64
read_branch_records
get_ns_current_pid_tgid
xdp_output
get_netns_cookie
get_current_ancestor_cgroup_id
sk_assign
ktime_get_boot_ns
seq_printf
seq_write
sk_cgroup_id
sk_ancestor_cgroup_id
ringbuf_output
ringbuf_reserve
ringbuf_submit
ringbuf_discard
ringbuf_query
csum_level
skc_to_tcp6_sock
skc_to_tcp_sock
skc_to_tcp_timewait_sock
skc_to_tcp_request_sock
skc_to_udp6_sock
get_task_stack
load_hdr_opt
store_hdr_opt
reserve_hdr_opt
inode_storage_get
inode_storage_delete
d_path
copy_from_user
snprintf_btf
seq_printf_btf
skb_cgroup_classid
redirect_neigh
check_mtu
```

MAPS

data interface

kernel/user API:

bpf_map_lookup_elem

bpf_map_update_elem

bpf_map_delete_elem

...

```
BPF_MAP_TYPE_ARRAY
BPF_MAP_TYPE_PERCPU_ARRAY
BPF_MAP_TYPE_PROG_ARRAY
BPF_MAP_TYPE_PERF_EVENT_ARRAY
BPF_MAP_TYPE_CGROUP_ARRAY
BPF_MAP_TYPE_CGROUP_STORAGE
BPF_MAP_TYPE_PERCPU_CGROUP_STORAGE
BPF_MAP_TYPE_HASH
BPF_MAP_TYPE_PERCPU_HASH
BPF_MAP_TYPE_LRU_HASH
BPF_MAP_TYPE_LRU_PERCPU_HASH
BPF_MAP_TYPE_LPM_TRIE
BPF_MAP_TYPE_STACK_TRACE
BPF_MAP_TYPE_ARRAY_OF_MAPS
BPF_MAP_TYPE_HASH_OF_MAPS
BPF_MAP_TYPE_DEVMAP
BPF_MAP_TYPE_DEVMAP_HASH
BPF_MAP_TYPE_SK_STORAGE
BPF_MAP_TYPE_SOCKMAP
BPF_MAP_TYPE_SOCKHASH
BPF_MAP_TYPE_INODE_STORAGE
BPF_MAP_TYPE_TASK_STORAGE
BPF_MAP_TYPE_CPUMAP
BPF_MAP_TYPE_XSKMAP
BPF_MAP_TYPE_REUSEPORT_SOCKARRAY
BPF_MAP_TYPE_QUEUE
BPF_MAP_TYPE_STACK
BPF_MAP_TYPE_STRUCT_OPS
BPF_MAP_TYPE_RINGBUF
```

VERIFIER

- static code analyzer
- instructions limit (1M)
- out of bound/malformed jump
- tracks pointers (arithmetic ops)
- helper calls

JIT (Just In Time) compiler

CONFIG_BPF_JIT

```
 0: (b7) r1 = 0
 1: (7b) *(u64 *)(r10 -16) = r1
 2: (b7) r1 = 1
 3: (7b) *(u64 *)(r10 -8) = r1
 4: (18) r1 = map[id:3]
 6: (bf) r2 = r10
 7: (07) r2 += -16
 8: (bf) r3 = r10
 9: (07) r3 += -8
10: (b7) r4 = 0
11: (85) call htab_map_update_elem
...
```

```
 0:   nopl   0x0(%rax,%rax,1)
 5:   push   %rbp
 6:   mov    %rsp,%rbp
 9:   sub    $0x10,%rsp
10:   push   %rbx
11:   push   %r13
13:   push   %r14
15:   push   %r15
17:   pushq  $0x0
19:   xor    %edi,%edi
1b:   mov    %rdi,-0x10(%rbp)
1f:   mov    $0x1,%edi
24:   mov    %rdi,-0x8(%rbp)
28:   movabs $0xffff889804e17400,%rdi
32:   mov    %rbp,%rsi
35:   add    $0xfffffffffffffff0,%rsi
39:   mov    %rbp,%rdx
3c:   add    $0xfffffffffffffff8,%rdx
40:   xor    %ecx,%ecx
42:   callq  0xffffffffe0ddb244
47:   xor    %eax,%eax
49:   pop    %rbx
4a:   pop    %r15
4c:   pop    %r14
4e:   pop    %r13
50:   pop    %rbx
51:   leaveq
52:   retq
```

COMPILE BPF

clang support

```
$ clang -target bpf
```

```
#include "bpf_iter.h"
#include "bpf_tracing_net.h"
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_tracing.h>

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

struct {
        __uint(type, BPF_MAP_TYPE_SK_STORAGE);
        __uint(map_flags, BPF_F_NO_PREALLOC);
        __type(key, int);
        __type(value, int);
} sk_stg_map SEC(".maps");

__u32 val_sum = 0;
__u32 ipv6_sk_count = 0;

SEC("iter/bpf_sk_storage_map")
int dump_bpf_sk_storage_map(struct bpf_iter__bpf_sk_storage_map *ctx)
{
        struct sock *sk = ctx->sk;
        __u32 *val = ctx->value;

        if (sk == (void *)0 || val == (void *)0)
                return 0;
```

COMPILE BPF

raw instructions

include/linux/filter.h

samples/bpf/bpf_insn.h

```
struct bpf_insn prog[] = {
   /*
    * Save sk_buff for future usage. value stored in R6 to R10 will
    * not be reset after a bpf helper function call.
    */
   BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
   /*
    * pc1: BPF_FUNC_get_socket_cookie takes one parameter,
    * R1: sk_buff
    */
   BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0,
                BPF_FUNC_get_socket_cookie),
   /* pc2-4: save &socketCookie to r7 for future usage*/
   BPF_STX_MEM(BPF_DW, BPF_REG_10, BPF_REG_0, -8),
   BPF_MOV64_REG(BPF_REG_7, BPF_REG_10),
   BPF_ALU64_IMM(BPF_ADD, BPF_REG_7, -8),
   /*
    * pc5-8: set up the registers for BPF_FUNC_map_lookup_elem,
    * it takes two parameters (R1: map_fd,  R2: &socket_cookie)
    */
   BPF_LD_MAP_FD(BPF_REG_1, map_fd),
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

PROBES

- **tracepoints**

  kprobes/uprobes

  trampolines

  LSM probes

  perf events

  cgroups

  iterators

```
/*
 * Mark the task runnable and perform wakeup-preemption.
 */
static void ttwu_do_wakeup(struct rq *rq, struct task_struct *p,
                           int wake_flags, struct rq_flags *rf)
{
        check_preempt_curr(rq, p, wake_flags);
        p->state = TASK_RUNNING;
        trace_sched_wakeup(p);

#ifdef CONFIG_SMP
        if (p->sched_class->task_woken) {
                rq_unpin_lock(rq, rf);
                p->sched_class->task_woken(rq, p);
                rq_repin_lock(rq, rf);
        }

        if (rq->idle_stamp) {
                u64 delta = rq_clock(rq) - rq->idle_stamp;
                u64 max = 2*rq->max_idle_balance_cost;

                update_avg(&rq->avg_idle, delta);

                if (rq->avg_idle > max)
                        rq->avg_idle = max;

                rq->idle_stamp = 0;
        }
#endif
}
```

PROBES

- tracepoints
- **kprobes/uprobes**
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

```
ffffffff810f20d0 <sched_fork>:
ffffffff810f20d0:       callq  ffffffff81050850 <__fentry__>
ffffffff810f20d5:       push   %rbp
ffffffff810f20d6:       mov    %rsi,%rdi
ffffffff810f20d9:       mov    %rsp,%rbp
ffffffff810f20dc:       push   %r15
ffffffff810f20de:       push   %r14
ffffffff810f20e0:       push   %r13
ffffffff810f20e2:       push   %r12
ffffffff810f20e4:       push   %rbx
ffffffff810f20e5:       mov    %rsi,%rbx
ffffffff810f20e8:       sub    $0x10,%rsp
ffffffff810f20ec:       callq  ffffffff810ea9d0 <__sched_fork.constpr
ffffffff810f20f1:       movq   $0x800,0x18(%rbx)
ffffffff810f20f9:       mov    %gs:0x16d00,%rax
ffffffff810f2102:       mov    0x74(%rax),%eax
ffffffff810f2105:       mov    %eax,0x6c(%rbx)
ffffffff810f2108:       testb  $0x1,0x504(%rbx)
ffffffff810f210f:       jne    ffffffff810f2290 <sched_fork+0x1c0>
ffffffff810f2115:       test   %eax,%eax
ffffffff810f2117:       js     ffffffff810f227c <sched_fork+0x1ac>
ffffffff810f211d:       cmp    $0x64,%eax
ffffffff810f2120:       mov    $0xffffffff823e0948,%rdx
ffffffff810f2127:       mov    $0xffffffff823e0a10,%rax
ffffffff810f212e:       cmovge %rdx,%rax
ffffffff810f2132:       mov    %rax,0x80(%rbx)
ffffffff810f2139:       lea    0xc0(%rbx),%r13
ffffffff810f2140:       lea    0x81c(%rbx),%r15
ffffffff810f2147:       mov    %r13,%rdi
ffffffff810f214a:       callq  ffffffff810fd180 <init_entity_runnable_
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

```
ffffffff810f20d0 <sched_fork>:
ffffffff810f20d0:       callq  ffffffff81050850 <__fentry__>
ffffffff810f20d5:       push   %rbp
ffffffff810f20d6:       mov    %rsi,%rdi
ffffffff810f20d9:       mov    %rsp,%rbp
ffffffff810f20dc:       push   %r15
ffffffff810f20de:       push   %r14
ffffffff810f20e0:       push   %r13
ffffffff810f20e2:       push   %r12
ffffffff810f20e4:       push   %rbx
ffffffff810f20e5:       mov    %rsi,%rbx
ffffffff810f20e8:       sub    $0x10,%rsp
ffffffff810f20ec:       callq  ffffffff810ea9d0 <__sched_fork.constpr
ffffffff810f20f1:       movq   $0x800,0x18(%rbx)
ffffffff810f20f9:       mov    %gs:0x16d00,%rax
ffffffff810f2102:       mov    0x74(%rax),%eax
ffffffff810f2105:       mov    %eax,0x6c(%rbx)
ffffffff810f2108:       testb  $0x1,0x504(%rbx)
ffffffff810f210f:       jne    ffffffff810f2290 <sched_fork+0x1c0>
ffffffff810f2115:       test   %eax,%eax
ffffffff810f2117:       js     ffffffff810f227c <sched_fork+0x1ac>
ffffffff810f211d:       cmp    $0x64,%eax
ffffffff810f2120:       mov    $0xffffffff823e0948,%rdx
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

```
ffffffff810f20d0 <sched_fork>:
ffffffff810f20d0:       callq  ffffffff81050850 <__fentry__>
ffffffff810f20d5:       push   %rbp
ffffffff810f20d6:       mov    %rsi,%rdi
ffffffff810f20d9:       mov    %rsp,%rbp
ffffffff810f20dc:       push   %r15
ffffffff810f20de:       push   %r14
ffffffff810f20e0:       push   %r13
ffffffff810f20e2:       push   %r12
ffffffff810f20e4:       push   %rbx
ffffffff810f20e5:       mov    %rsi,%rbx
ffffffff810f20e8:       sub    $0x10,%rsp
ffffffff810f20ec:       callq  ffffffff81
ffffffff810f20f1:       movq   $0x800,0x1
ffffffff810f20f9:       mov    %gs:0x16d0
ffffffff810f2102:       mov    0x74(%rax)
ffffffff810f2105:       mov    %eax,0x6c(
ffffffff810f2108:       testb  $0x1,0x504
ffffffff810f210f:       jne    ffffffff81
ffffffff810f2115:       test   %eax,%eax
ffffffff810f2117:       js     ffffffff81
ffffffff810f211d:       cmp    $0x64,%eax
ffffffff810f2120:       mov    $0xffffff
```

```
 0:   nopl   0x0(%rax,%rax,1)
 5:   push   %rbp
 6:   mov    %rsp,%rbp
 9:   sub    $0x10,%rsp
10:   push   %rbx
11:   push   %r13
13:   push   %r14
15:   push   %r15
17:   pushq  $0x0
19:   xor    %edi,%edi
1b:   mov    %rdi,-0x10(%rbp)
1f:   mov    $0x1,%edi
24:   mov    %rdi,-0x8(%rbp)
28:   movabs $0xffff889804e17400,%rdi
32:   mov    %rbp,%rsi
35:   add    $0xfffffffffffffff0,%rsi
39:   mov    %rbp,%rdx
3c:   add    $0xfffffffffffffff8,%rdx
40:   xor    %ecx,%ecx
42:   callq  0xffffffffe0ddb244
47:   xor    %eax,%eax
49:   pop    %rbx
4a:   pop    %r15
4c:   pop    %r14
4e:   pop    %r13
50:   pop    %rbx
51:   leaveq
52:   retq
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

```
   0:   nopl   0x0(%rax,%rax,1)
   5:   push   %rbp
   6:   mov    %rsp,%rbp
   9:   sub    $0x10,%rsp
  10:   push   %rbx
  11:   push   %r13
  13:   push   %r14
  15:   push   %r15
  17:   pushq  $0x0
  19:   xor    %edi,%edi
  1b:   mov    %rdi,-0x10(%rbp)
  1f:   mov    $0x1,%edi
  24:   mov    %rdi,-0x8(%rbp)
  28:   movabs $0xffff889804e17400,%rdi
  32:   mov    %rbp,%rsi
  35:   add    $0xfffffffffffffff0,%rsi
  39:   mov    %rbp,%rdx
  3c:   add    $0xfffffffffffffff8,%rdx
  40:   xor    %ecx,%ecx
  42:   callq  0xffffffffe0ddb244
  47:   xor    %eax,%eax
  49:   pop    %rbx
  4a:   pop    %r15
  4c:   pop    %r14
  4e:   pop    %r13
  50:   pop    %rbx
  51:   leaveq
  52:   retq
```

```
0: (bf) r6 = r1
1: (69) r7 = *(u16 *)(r6 +176)
2: (b4) w8 = 0
3: (44) w8 |= 2
4: (b7) r0 = 1
5: (55) if r8 != 0x2 goto pc+1
6: (b7) r0 = 0
7: (95) exit
```

PROBES

- tracepoints
- kprobes/uprobes
- **trampolines**
- LSM probes
- perf events
- cgroups
- iterators

```
  0:   nopl   0x0(%rax,%rax,1)
  5:   push   %rbp
  6:   mov    %rsp,%rbp
  9:   sub    $0x10,%rsp
 10:   push   %rbx
 11:   push   %r13
 13:   push   %r14
 15:   push   %r15
 17:   pushq  $0x0
 19:   xor    %edi,%edi
 1b:   mov    %rdi,-0x10(%rbp)
 1f:   mov    $0x1,%edi
 24:   mov    %rdi,-0x8(%rbp)
 28:   movabs $0xffff889804e17400,%rdi
 32:   mov    %rbp,%rsi
 35:   add    $0xfffffffffffffff0,%rsi
 39:   mov    %rbp,%rdx
 3c:   add    $0xfffffffffffffff8,%rdx
 40:   xor    %ecx,%ecx
 42:   callq  0xffffffffe0ddb244
 47:   xor    %eax,%eax
 49:   pop    %rbx
 4a:   pop    %r15
 4c:   pop    %r14
 4e:   pop    %r13
 50:   pop    %rbx
 51:   leaveq
 52:   retq
```

```
0: (bf) r6 = r1
1: (69) r7 = *(u16 *)(r6 +176)
2: (b4) w8 = 0
3: (44) w8 |= 2
4: (b7) r0 = 1
5: (55) if r8 != 0x2 goto pc+1
6: (b7) r0 = 0
7: (95) exit
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- **LSM probes**
- perf events
- cgroups
- iterators

```
static int __init bpf_lsm_init(void)
{
        security_add_hooks(bpf_lsm_hooks, ARRAY_SIZE(bpf_lsm_hooks),
                                   "bpf");
        pr_info("LSM support for eBPF active\n");
        return 0;
}


static int do_dentry_open(struct file *f,
                          struct inode *inode,
                          int (*open)(struct inode *, struct file *))
{
  static const struct file_operations empty_fops = {};
  int error;

  path_get(&f->f_path);
  f->f_inode = inode;

  ...

  error = security_file_open(f);
  if (error)
     goto cleanup_all;
  ...

}
```

PROBES

- tracepoints
- kprobes/uprobes
- trampolines
- LSM probes
- perf events
- cgroups
- iterators

It’s there.. ;-)

KERNEL - BPF syscall

```
long sys_bpf(int cmd, union bpf_attr *attr, unsigned int size);


union bpf_attr {
        struct { /* anonymous struct used by BPF_MAP_CREATE command */
                __u32   map_type;       /* one of enum bpf_map_type */
                __u32   key_size;       /* size of key in bytes */
                ...
        };

        struct { /* anonymous struct used by BPF_MAP_*_ELEM commands */
                __u32           map_fd;
                __aligned_u64   key;
                ...
        };

        struct { /* struct used by BPF_MAP_*_BATCH commands */
                __aligned_u64   in_batch;       /* start batch,
                __aligned_u64   out_batch;      /* output: next start batch
*/
                ...
}
```

KERNEL - BPF syscall

```
prog_fd = sys_bpf(BPF_PROG_LOAD, &attr, ...)

...


tp_fd = sys_bpf(BPF_RAW_TRACEPOINT_OPEN, &attr ...

...


close(tp_fd);
close(prog_fd);
```

VERIFIER

TRACEPOINTS

TRAMPOLINES

BTF

- BPF Type Format (not a debuginfo)
- extended with functions and source/line info
- used in verifier
- generated by:
  - clang -g for BPF programs
  - pahole/dwarves for kernel/modules

LIBBPF

- in kernel lib (tools/lib/bpf)
- github mirror for packaging
  - https://github.com/libbpf/libbpf
- core BPF/BTF functionality
- libbpf-* packages in Fedora/RHEL
- static/dynamic link

SELFTESTS

- in kernel tests (tools/testing/selftests/bpf)
- tight LLVM/CLANG upstream dependency
- no package in Fedora

BPFTOOL

- maintenance/developing tool
- in kernel (tools/bpf/bpftool)
- bpftool package in Fedora/RHEL

BCC

- high level library
- python tools
- C/C++ tools
- bcc-* packages in Fedora/RHEL
- https://github.com/iovisor/bcc

BCC TOOLS

```
from bcc import BPF
from bcc.containers import filter_by_containers
from bcc.utils import ArgString, printb
import bcc.utils as utils

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

...

BPF_PERF_OUTPUT(events);

static int __submit_arg(struct pt_regs *ctx, void *ptr, struct data_t *data)
{
    bpf_probe_read_user(data->argv, sizeof(data->argv), ptr);
    events.perf_submit(ctx, data, sizeof(struct data_t));
    return 1;
}

...

# initialize BPF
b = BPF(text=bpf_text)
execve_fnname = b.get_syscall_fnname("execve")
b.attach_kprobe(event=execve_fnname, fn_name="syscall__execve")
b.attach_kretprobe(event=execve_fnname, fn_name="do_ret_sys_execve")

...
```

BCC TOOLS

```
from bcc import BPF
from bcc.containers import filter_by_containers
from bcc.utils import ArgString, printb
import bcc.utils as utils
```

```
bpf_text = """
#include <uapi/linux/ptrace.h>
#include <linux/sched.h>
#include <linux/fs.h>

...

BPF_PERF_OUTPUT(events);

static int __submit_arg(struct pt_regs *ctx, void *ptr, struct data_t *data)
{
    bpf_probe_read_user(data->argv, sizeof(data->argv), ptr);
    events.perf_submit(ctx, data, sizeof(struct data_t));
    return 1;
}

...

# initialize BPF
b = BPF(text=bpf_text)
execve_fnname = b.get_syscall_fnname("execve")
b.attach_kprobe(event=execve_fnname, fn_name="syscall__execve")
b.attach_kretprobe(event=execve_fnname, fn_name="do_ret_sys_execve")

...
```

BCC TOOLS

```
from bcc import BPF
from bcc.containers import filter_by_containers
from bcc.utils import ArgString, printb
import bcc.utils as utils
```

```
bpf_text = """
#include <uapi/linux/ptrace.h>
#include <linux/sched.h>
#include <linux/fs.h>

...

BPF_PERF_OUTPUT(events);

static int __submit_arg(struct pt_regs *ctx, void *ptr, struct data_t *data)
{
    bpf_probe_read_user(data->argv, sizeof(data->argv), ptr);
    events.perf_submit(ctx, data, sizeof(struct data_t));
    return 1;
}

...
```

```
# initialize BPF
b = BPF(text=bpf_text)
execve_fnname = b.get_syscall_fnname("execve")
b.attach_kprobe(event=execve_fnname, fn_name="syscall__execve")
b.attach_kretprobe(event=execve_fnname, fn_name="do_ret_sys_execve")

...
```

BCC TOOLS

```
from bcc import BPF
from bcc.containers import filter_by_containers
from bcc.utils import ArgString, printb
import bcc.utils as utils

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

...

BPF_PERF_OUTPUT(events);

static int __submit_arg(struct pt_regs *ctx, void *ptr, struct data_t *data)
{
    bpf_probe_read_user(data->argv, sizeof(data->argv), ptr);
    events.perf_submit(ctx, data, sizeof(struct data_t));
    return 1;
}

...
```

```
# initialize BPF
b = BPF(text=bpf_text)
execve_fnname = b.get_syscall_fnname("execve")
b.attach_kprobe(event=execve_fnname, fn_name="syscall__execve")
b.attach_kretprobe(event=execve_fnname, fn_name="do_ret_sys_execve")

...
```

BCC TOOLS

```
# process event

def print_event(cpu, data, size):
    event = b["events"].event(data)
    skip = False

    if event.type == EventType.EVENT_ARG:
        argv[event.pid].append(event.argv)
    elif event.type == EventType.EVENT_RET:
        if event.retval != 0 and not args.fails:
            skip = True
        if args.name and not re.search(bytes(args.name), event.comm):
            skip = True
        if args.line and not re.search(bytes(args.line),
                                       b' '.join(argv[event.pid])):
            skip = True
        if args.quote:


...

# loop with callback to print_event
b["events"].open_perf_buffer(print_event)
while 1:
    try:
        b.perf_buffer_poll()
    except KeyboardInterrupt:
        exit()
```

BCC TOOLS

kernel change:

```
9bb48c82aced tty: implement write_iter

-static ssize_t tty_write(struct file *, const char __user *, size_t, loff_t *);
+static ssize_t tty_write(struct kiocb *, struct iov_iter *);
```

ttysnoop change:

```
-int kprobe__tty_write(struct pt_regs *ctx, struct file *file,
-                      const char __user *buf, size_t count)
+KFUNC_PROBE(tty_write, struct kiocb *iocb, struct iov_iter *from)
```

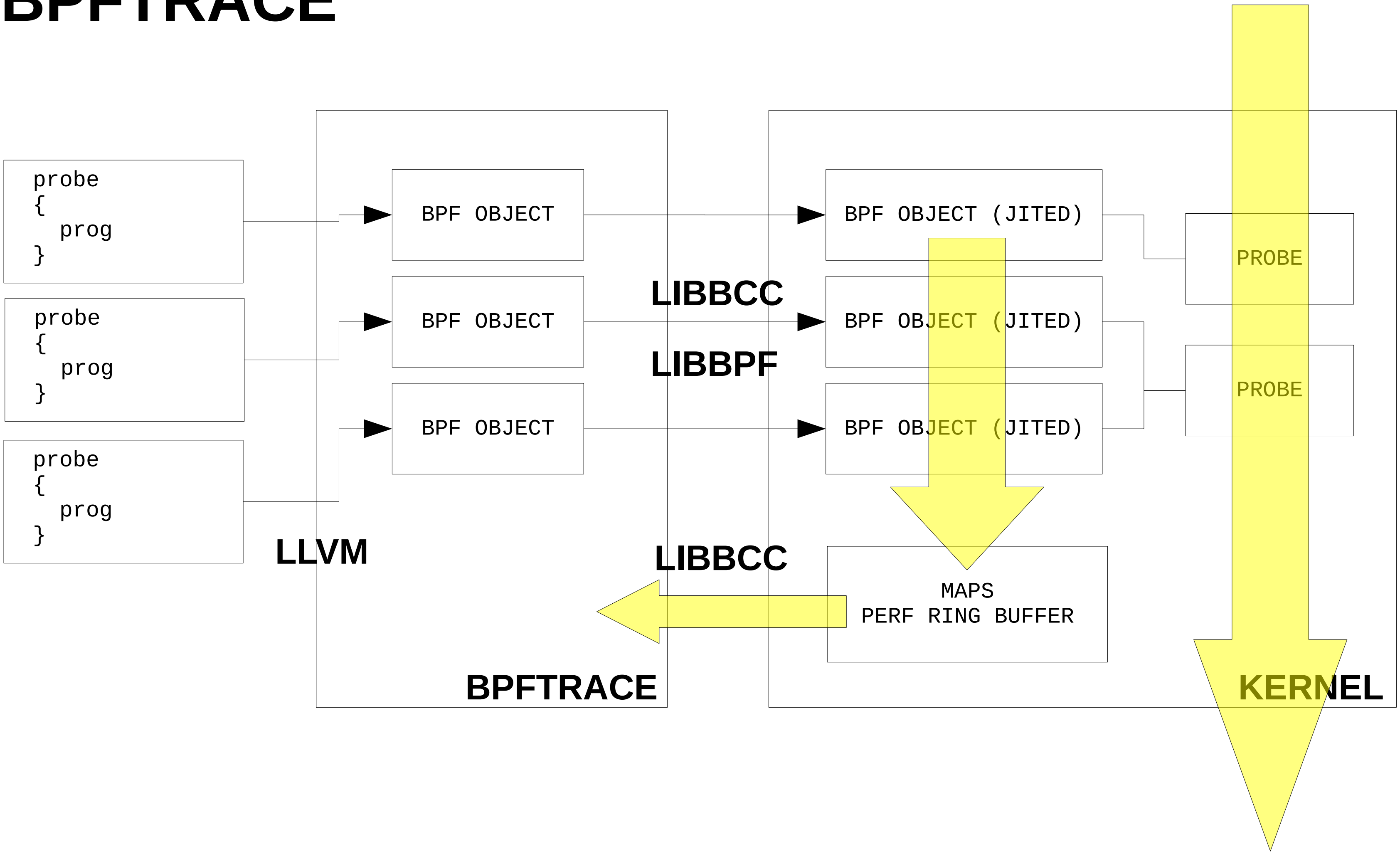
BPFTRACE

- dtrace functionality
- bpftrace package in Fedora/RHEL
- https://github.com/iovisor/bpftrace
- https://github.com/iovisor/bpftrace/tree/master/docs

```
probe /filter/
{
  prog
}
```

```
tracepoint:syscalls:sys_enter_nanosleep
/pid != 4321/
{
  printf("%s is sleeping.\n", comm);
}
```

BPFTRACE



probe
{
prog
}
probe
{
prog
}
probe
{
prog
}
LLVM
BPF OBJECT
BPF OBJECT
BPF OBJECT
BPFTRACE
LIBBCC
LIBBPF
LIBBCC
BPF OBJECT (JITED)
BPF OBJECT (JITED)
BPF OBJECT (JITED)
PROBE
PROBE
MAPS
PERF RING BUFFER
KERNEL

thanks, questions..

BPFTRACE

```
# Sleeping calls

#!/usr/local/bin/bpftrace
tracepoint:syscalls:sys_enter_nanosleep
{
  printf("%s is sleeping.\n", comm);
}

# 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); }'
```