BPF Debugging

Yonghong Song

Use Case

```
/* from linux/tools/testing/selftests/bpf/test_l4lb_noinline.c */
324 static __always_inline int
      process_packet(void *data, __u64 off, void *data_end,
325                  bool is_ipv6, struct __sk_buff *skb)
326 {
...
330       struct bpf_tunnel_key tkey = {};
332       struct real_definition *dst;
…
348       tkey.tunnel_ttl = 64;
…
425       if (!get_packet_dst(&dst, &pckt, vip_info, is_ipv6))
426             return TC_ACT_SHOT;
427
428       if (dst->flags & F_IPV6) {
429             cval = bpf_map_lookup_elem(&ctl_array, &v6_intf_pos);
430             if (!cval)
431                   return TC_ACT_SHOT;
432             ifindex = cval->ifindex;
433             memcpy(tkey.remote_ipv6, dst->dstv6, 16);
434             tun_flag = BPF_F_TUNINFO_IPV6;
435       }
```

```
      else {
436             cval = bpf_map_lookup_elem(&ctl_array,
                                          &v4_intf_pos);
437             if (!cval)
438                   return TC_ACT_SHOT;
439             ifindex = cval->ifindex;
440             tkey.remote_ipv4 = dst->dst;
441       }
442       vip_num = vip_info->vip_num;
443       data_stats = bpf_map_lookup_elem(&stats, &vip_num);
444       if (!data_stats)
445             return TC_ACT_SHOT;
446       data_stats->pkts++;
447       data_stats->bytes += pkt_bytes;
448       bpf_skb_set_tunnel_key(skb, &tkey, sizeof(tkey),
                                 tun_flag);
```

Problem: some tunnel remote ipv6 address not set up at line 448!

Hypothesis

```
/* from linux/tools/testing/selftests/bpf/test_l4lb_noinline.c */
324 static __always_inline int
      process_packet(void *data, __u64 off, void *data_end,
325                  bool is_ipv6, struct __sk_buff *skb)
326 {
...
330       struct bpf_tunnel_key tkey = {};
332       struct real_definition *dst;
…
348       tkey.tunnel_ttl = 64;
…
425       if (!get_packet_dst(&dst, &pckt, vip_info, is_ipv6))
426             return TC_ACT_SHOT;
427
428       if (dst->flags & F_IPV6) {
429             cval = bpf_map_lookup_elem(&ctl_array, &v6_intf_pos);
430             if (!cval)
431                   return TC_ACT_SHOT;
432             ifindex = cval->ifindex;
433             memcpy(tkey.remote_ipv6, dst->dstv6, 16);
434             tun_flag = BPF_F_TUNINFO_IPV6;
435       }
```

```
      else {
436             cval = bpf_map_lookup_elem(&ctl_array,
                                          &v4_intf_pos);
437             if (!cval)
438                   return TC_ACT_SHOT;
439             ifindex = cval->ifindex;
440             tkey.remote_ipv4 = dst->dst;
441       }
442       vip_num = vip_info->vip_num;
443       data_stats = bpf_map_lookup_elem(&stats, &vip_num);
444       if (!data_stats)
445             return TC_ACT_SHOT;
446       data_stats->pkts++;
447       data_stats->bytes += pkt_bytes;
448       bpf_skb_set_tunnel_key(skb, &tkey, sizeof(tkey),
                                 tun_flag);
```

Let us examine whether due to map lookup failures.

Break At A Particular Place

- Check whether we hit line 431 and 445 if dst->dstv6 == <ADDR>.

Checking Memory State

```
        else {
436             cval = bpf_map_lookup_elem(&ctl_array,
                                           &v4_intf_pos);
437             if (!cval)
438                     return TC_ACT_SHOT;
439             ifindex = cval->ifindex;
440             tkey.remote_ipv4 = dst->dst;
441     }
442     vip_num = vip_info->vip_num;
443     data_stats = bpf_map_lookup_elem(&stats, &vip_num);
444     if (!data_stats)
445             return TC_ACT_SHOT;
446     data_stats->pkts++;
447     data_stats->bytes += pkt_bytes;
448     bpf_skb_set_tunnel_key(skb, &tkey, sizeof(tkey),
                               tun_flag);
```

- Now suppose Line 445 is the issue.
- Possible actions:
  - print *vip_num*
  - dump *stats* map

High-Level Action

- Break at <file>:431 if dst->dstv6 == <ADDR>
- Break at <file>:445 if dst->dstv6 == <ADDR>
- Break at <file>:445 if dst->dstv6 == <ADDR> print vip_num, stats

Proposed Spec

bpftool prog inspect <PROG> <INSPECT_SPEC> <INSPECT_SPEC> …

INSPECT_SPEC := <LOC> <ACTION> if <CONDITION>

LOC := <func_name>:<offset>

ACTION := [skip <n>] print {EXPR | reg(<REG>) | mem(<ADDR>, <SIZE>),}+

CONDITION := EXPR | watch <ADDR>, <SIZE>, <TYPE>

Begin with debugging jited assembly only!

Extend to source base later (BTF support to map variables to registers/memory locations)

Debugging Command

bpftool prog inspect id 14
'process_packet:230
print "hit 1"
if $r1 == 0 and mem($r9, 0) == <ADDR1> and
mem($r9, 8) == <ADDR2>'

bpftool prog inspect id 14
'process_packet:289
print mem($r10 - 116, 8), stats
if $r1 == 0 and mem($9, 0) == <ADDR1> and
mem($r9, 8) == <ADDR2>'

Source annotated codes should help find register/stack correlation to local variables.

```
int process_packet(...):
; static int process_packet(...)
  25: (bf) r6 = r4
  26: (bf) r7 = r3
  27: (bf) r8 = r2
…..
 242: (bf) r2 = r10
; cval = bpf_map_lookup_elem(&ctl_array, &v6_intf_pos);
 243: (07) r2 += -88
 244: (18) r1 = map[id:184]
…
; if (!cval)
 255: (15) if r1 == 0x0 goto pc+79
; ifindex = cval->ifindex;
```

Kernel Support

- Construct condition and action from each INSPECT_SPEC to a bpf kprobe prog
- The bpf prog is attached to jited code at specified location.
- The bpf prog share maps/globals, in read-only mode, with to-be-debugged bpf programs.
- Kernel support:
  - Nested kprobe and all other bpf programs
  - kprobe infra change to permit bpf jit addresses and bpf_prog_<TAG> lookup and permit nested kprobes.
- Debugging at bpf assembly level requires easy mapping of insn and registers from bpf xlated codes to jited codes, which is the case for x64.

Single Step Support

- QEMU
- Sleepable BPF program