Linux bcc/eBPF tcpdrop

31 May 2018

While debugging a production issue of kernel-based TCP packet drops, I remembered seeing a new function added in Linux 4.7 by Eric Dumazet (Google) called tcp_drop(), which I can trace using kprobes and bcc/eBPF. This lets me fetch extra context to explain why these drops are happening. Eg:

```
# tcpdrop
TIME     PID    IP SADDR:SPORT          > DADDR:DPORT          STATE (FLAGS)
05:46:07 82093  4  10.74.40.245:50010   > 10.74.40.245:58484   ESTABLISHED (ACK)
    tcp_drop+0x1
    tcp_rcv_established+0x1d5
    tcp_v4_do_rcv+0x141
    tcp_v4_rcv+0x9b8
    ip_local_deliver_finish+0x9b
    ip_local_deliver+0x6f
    ip_rcv_finish+0x124
    ip_rcv+0x291
    __netif_receive_skb_core+0x554
    __netif_receive_skb+0x18
    process_backlog+0xba
    net_rx_action+0x265
    __softirqentry_text_start+0xf2
    irq_exit+0xb6
    xen_evtchn_do_upcall+0x30
    xen_hvm_callback_vector+0x1af

05:46:07 85153  4  10.74.40.245:50010   > 10.74.40.245:58446   ESTABLISHED (ACK)
    tcp_drop+0x1
    tcp_rcv_established+0x1d5
    tcp_v4_do_rcv+0x141
    tcp_v4_rcv+0x9b8
    ip_local_deliver_finish+0x9b
    ip_local_deliver+0x6f
    ip_rcv_finish+0x124
    ip_rcv+0x291
    __netif_receive_skb_core+0x554
    __netif_receive_skb+0x18
    process_backlog+0xba
    net_rx_action+0x265
    __softirqentry_text_start+0xf2
    irq_exit+0xb6
    xen_evtchn_do_upcall+0x30
    xen_hvm_callback_vector+0x1af

[...]
```

This is tcpdrop, a new tool I've written for the open source bcc project. It shows source and destination packet details, as well as TCP session state (from the kernel), TCP flags (from the packet TCP header), and the kernel stack trace that led to this drop. That stack trace helps answer the why (you'll need to browse the code behind those functions to make sense of it). This is also information that's not on the wire, so you can never see this using packet sniffers (libpcap, tcpdump, etc).

I can't help but highlight this small but significant change from Eric's patch (tcp: increment sk_drops for dropped rx packets):

```
@@ -6054,7 +6061,7 @@  int tcp_rcv_state_process(struct sock *sk, struct sk_buff *skb)

    if (!queued) {
 discard:
-       __kfree_skb(skb);
+       tcp_drop(sk, skb);
    }
    return 0;
```

__kfree_skb() is called from many paths to free socket buffers, including routine codeapths. Tracing it was too noisy: you'd have your packet drop code paths lost among many normal ones. But with the new tcp_drop() function, I can trace just the TCP drops. I've already suggested some enhancements to Eric today at netconf18, such as adding a "reason" argument somewhere that I can trace for a more human description of why the packet was dropped. Maybe tcp_drop() should become a tracepoint too.

Here's some more code worth mentioning, this time some eBPF/C from my tcpdrop tool:

```
[...]
    // pull in details from the packet headers and the sock struct
    u16 family = sk->__sk_common.skc_family;
    char state = sk->__sk_common.skc_state;
    u16 sport = 0, dport = 0;
    u8 tcpflags = 0;
    struct tcphdr *tcp = skb_to_tcphdr(skb);
    struct iphdr *ip = skb_to_iphdr(skb);
    bpf_probe_read(&sport, sizeof(sport), &tcp->source);
    bpf_probe_read(&dport, sizeof(dport), &tcp->dest);
    bpf_probe_read(&tcpflags, sizeof(tcpflags), &tcp_flag_byte(tcp));
    sport = ntohs(sport);
    dport = ntohs(dport);

    if (family == AF_INET) {
        struct ipv4_data_t data4 = {.pid = pid, .ip = 4};
        bpf_probe_read(&data4.saddr, sizeof(u32), &ip->saddr);
        bpf_probe_read(&data4.daddr, sizeof(u32), &ip->daddr);
        data4.dport = dport;
        data4.sport = sport;
        data4.state = state;
        data4.tcpflags = tcpflags;
        data4.stack_id = stack_traces.get_stackid(ctx, 0);
        ipv4_events.perf_submit(ctx, &data4, sizeof(data4));
[...]
```

My prior tcp tools in bcc have made do with struct sock members alone (eg, tcplife). But this time I needed packet info to see TCP flags, and the direction of the packet. So it's the first time I've accessed TCP and IP headers in eBPF. I added skb_to_tcphdr() and skb_to_iphdr() in tcpdrop to help with this, as well as a new tcp bcc library for later processing in Python. I'm sure this code will get reused (and improved) over time.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).

About this blog