Packet Where aRe You

An eBPF based tool for diagnosing Linux networking

Presented by
Jef Spaleta - Isovalent
Technical Community Advocate and Curler

Why am I giving this talk?

I'm just a pwru user

I'm very good at breaking things in interesting ways

Most recently I've been doing a lot of breaking packet flow by learning how to write eBPF programs

And through that experience I've become a huge fan of the pwru tool and I think its something anyone who needs to diagnose Linux networking will want in their toolbelt

Talk Overview

Quick overview of why pwru exists

How it makes use of eBPF

Live(?) demos of pwru in action to help diagnose packets gone missing

What exactly is the problem?

Networking inside the Linux kernel is complicated

What exactly is the problem?

Networking inside Linux is complicated

- Network namespaces make it even more complicated!

What exactly is the problem?

Networking inside Linux is complicated

- Network namespaces make it even more complicated!
- **When a packet goes missing as a network engineer how do you know exactly where in the Linux kernel the problem is?**

What exactly is the problem?

Networking inside Linux is complicated

- Network namespaces make it even more complicated!
- When a packet goes missing as a network engineer how do you know exactly where in the Linux kernel the problem is?
- **And once you add eBPF and XDP networking programs into the mix, how can you know exactly all the possible code paths that you would need to look at?**

What exactly is the problem?

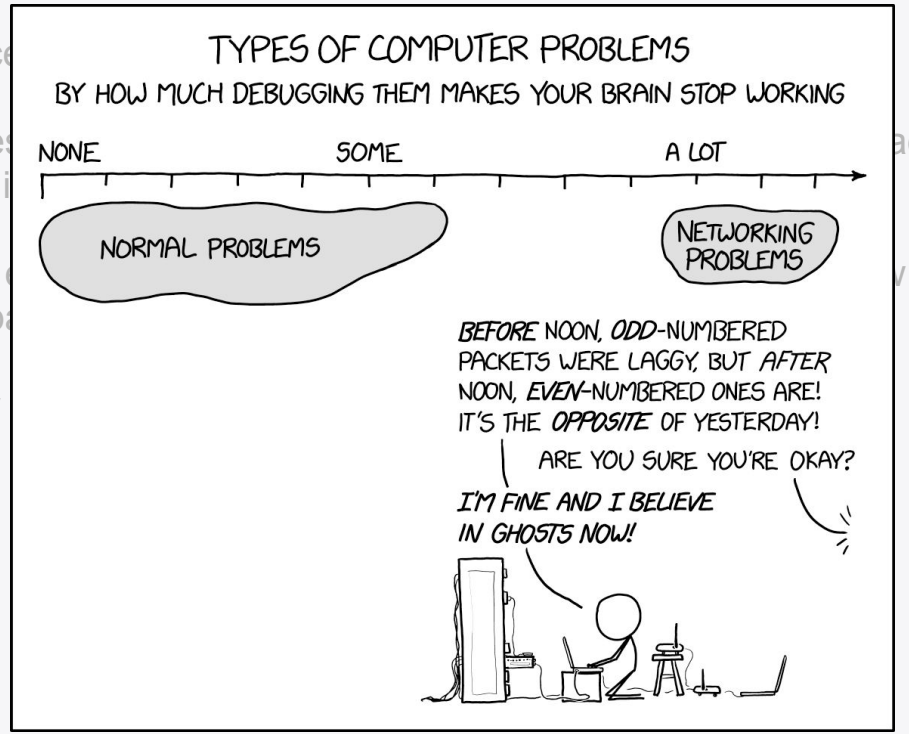
Networking inside Linux is complicated

- Network namespaces make it even more complicated!
- When a packet goes missing as a network engineer how do you know exactly where in the Linux kernel the problem is?
- And once you add eBPF and XDP networking programs into the mix, how can you know exactly all the possible code paths that you would need to look at?
- **How do you know what you don't know?**

What exactly is the problem?

Networking inside Linux is complicated

- Network namespac
- When a packet goes … actly where in the Linux kernel the problem i
- And once you add … y can you know exactly all the possible code pa
- **How do you know**



https://xkcd.com/2259/

Introducing (P)acket (W)here a(R)e (Y)ou

- Maintained as a networking diagnostic tool in the Cilium project https://github.com/cilium/pwru
- Linux networking tracing program using ebpf-go Golang library
- Uses pcap filtering semantics just like familiar CLI networking tools!
- Uses eBPF based Kprobes to instrument the path packets take through your Linux kernel

pwru gives you visibility into the Linux kernel functions that network packets flow through

eBPF? Why not just classic BPF?

- **eBPF programs can make use of key/value "maps" shared with userspace, that can be used for state tracking**

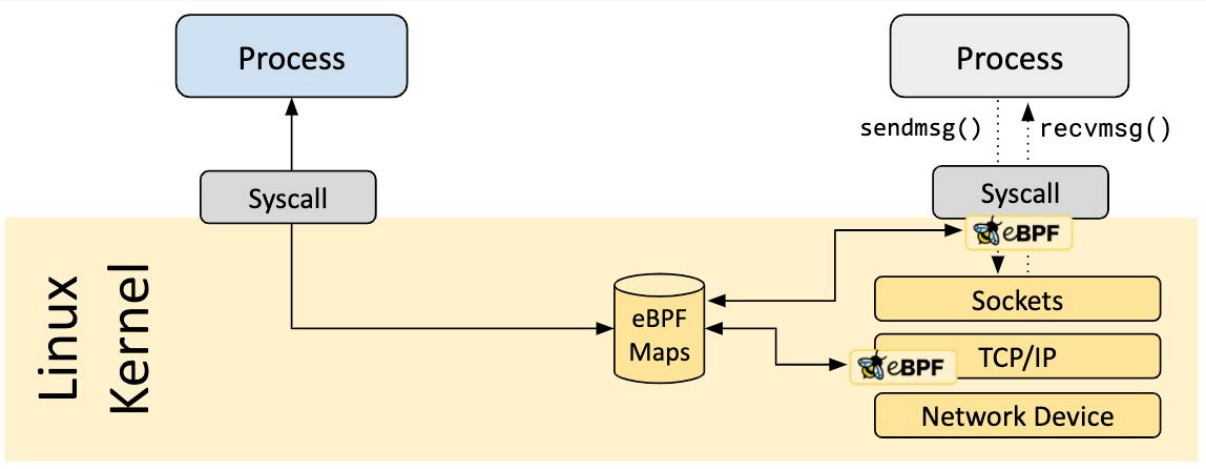


Image ref: https://ebpf.io/what-is-ebpf/#maps

eBPF? Why not just classic BPF?

- **eBPF lets you attach to nearly any kernel function (via *Kprobes*) not just network sockets**

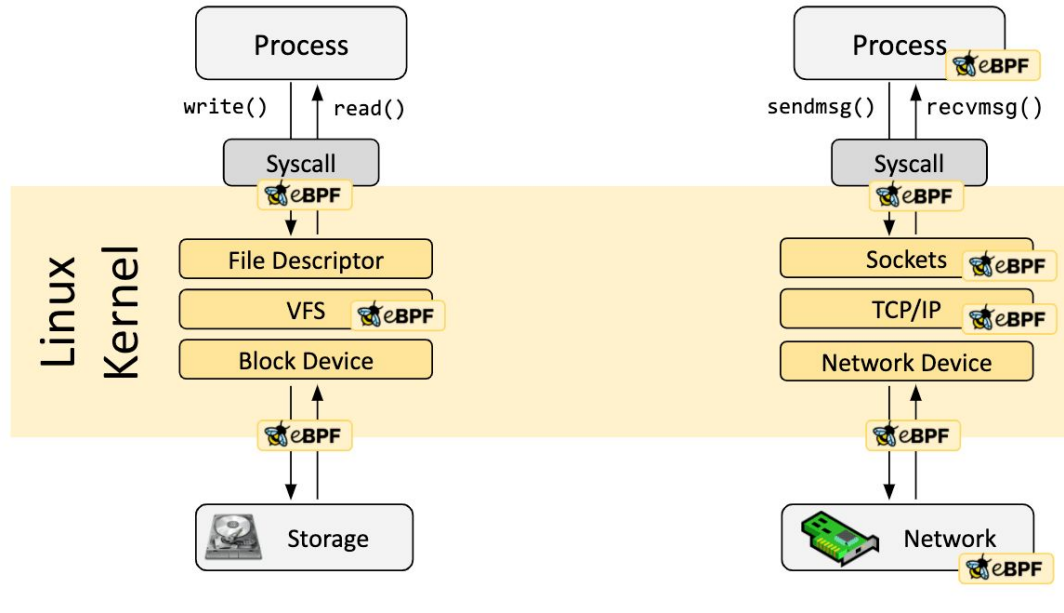


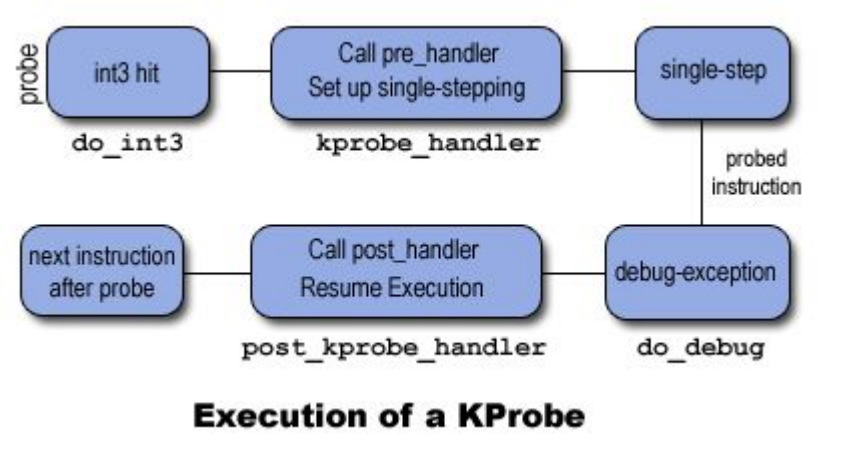
Image ref: https://ebpf.io/what-is-ebpf/#hook-overview

KProbes?

Event driven debugging tool for Linux kernel, that lets you trace specific kernel functions.

"When a KProbe is installed at a particular instruction and that instruction is executed, the pre-handler is executed just before the execution of the probed instruction. Similarly, the post-handler is executed just after the execution of the probed instruction"

Ref: https://lwn.net/Articles/132196/



PWRU what are you?

Is it a Linux function tracing program or is it a network tracing program?

It's a great tool for network engineers to identify Linux kernel* networking bugs, and provides enough kernel tracing context for kernel* engineers to address.

expansive use of kernel here to include eBPF programs

What I think makes pwru special

- Runtime injection of the optional pcap filter instructions into socket buffer related Kprobe eBPF programs
- Makes use of https://github.com/cloudflare/cbpfc under the covers to runtime compile pcap classic BPF filters into eBPF instructions

The result in a packet filterable view of the Linux kernel events

It's hard for packets to give pwru the slip

Because pwru is actually tracking the Linux kernel's socket buffer objects (with the help of eBPF maps!) it can also track changes to the socket buffer data that causes the pcap filter expression to no longer match

Manipulated packets can still be traced!

(I'll show an example of this near the end)

Built with container networking in mind

pwru was created to address the challenges of diagnosing connections between Linux network namespaces (a core function of the Cilium CNI)

- No need to nsenter to track packets flowing across network namespace boundaries
- Optionally filter by namespace using cmdline option (not possible via pcap filter language)

Exercise for the audience:

Install Kind cluster with Cilium and use pwru to trace internal cluster communications

Think of pwru as tcpdump for your in-kernel networking

- pwru picks up where tcpdump leaves off, tracing all the under-the-cover Linux kernel functions that packets flow through
- pwru helps diagnose ***some*** Linux kernel networking misbehavior that traditional network diagnostic tools can't see

Demo Time!

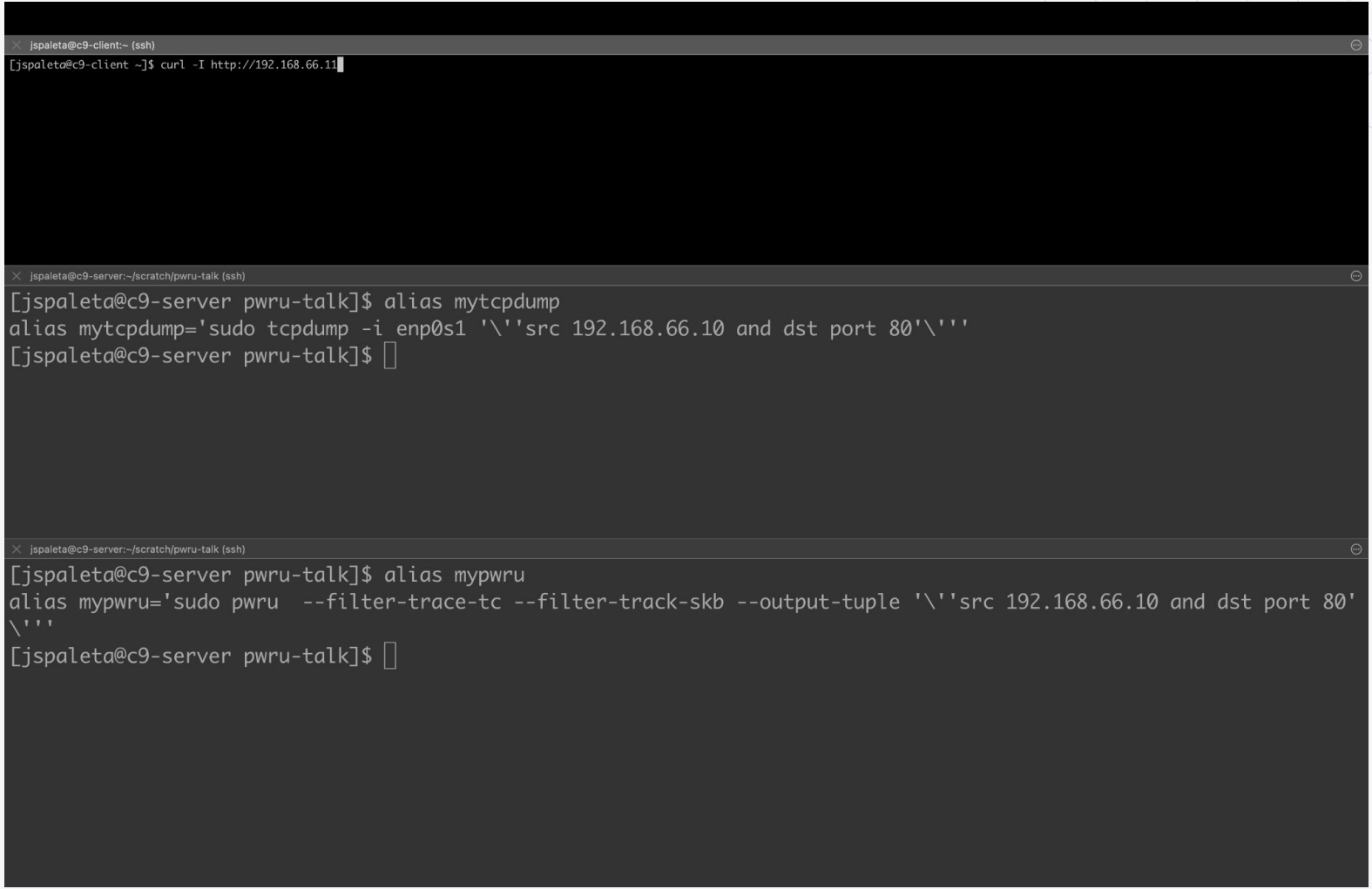
Let's start simple, lets use both tcpdump and pwru for a working curl command from a Linux host out to a local network server to get familiar with the value pwru provides.

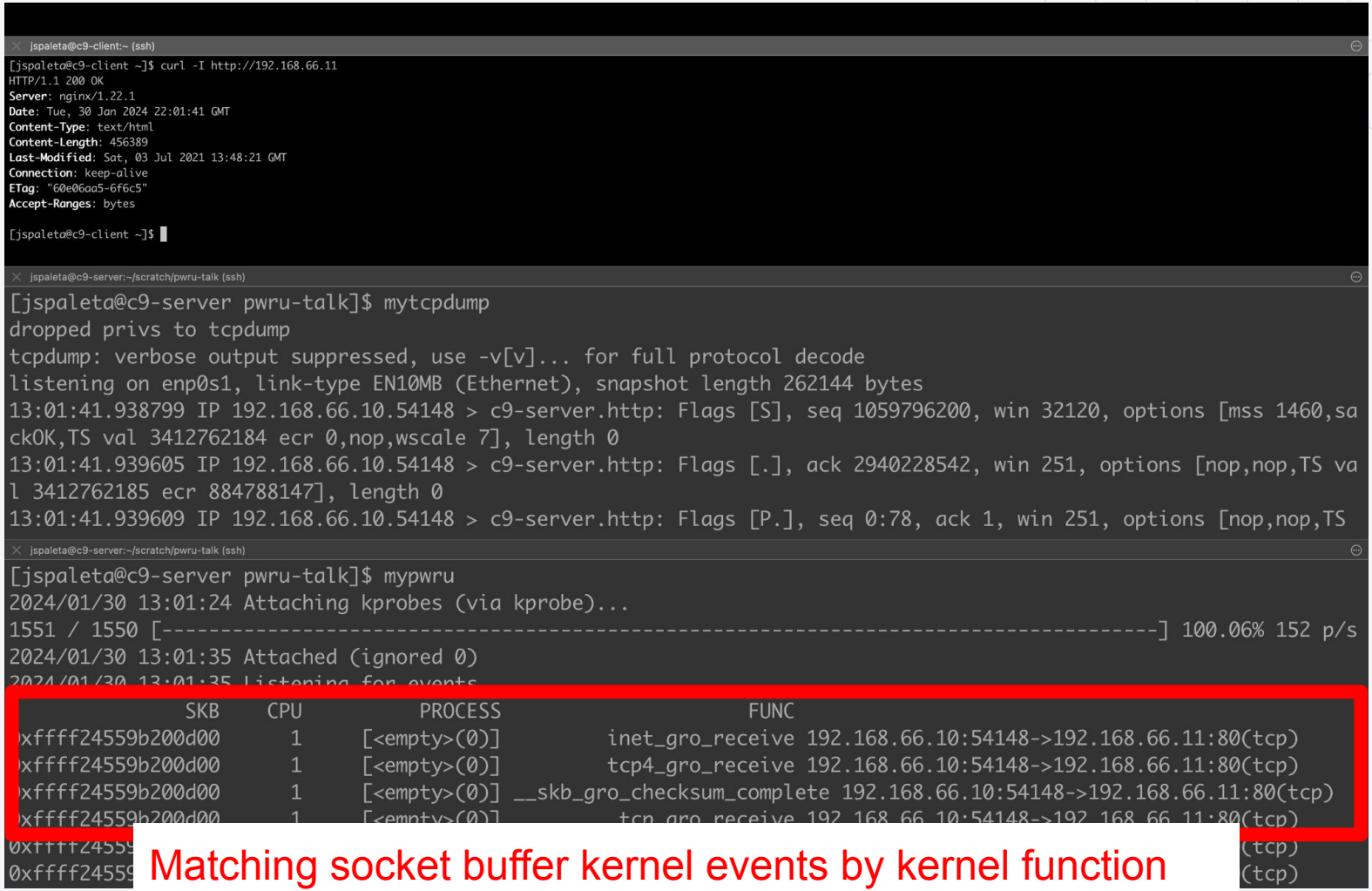
BASELINE DEMO: NFtables tracing works as expected

The scenario:

- **NFTables is configured to allow port 22 and port 80 on server VM**
- **Simple curl to http server from client VM**

Let's compare tcpdump with pwru using the same pcap filter running on the http server as I try to connect from a client.


jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
[jspaleta@c9-server pwru-talk]$ alias mytcpdump
alias mytcpdump='sudo tcpdump -i enp0s1 '\''src 192.168.66.10 and dst port 80'\'''
[jspaleta@c9-server pwru-talk]$
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
[jspaleta@c9-server pwru-talk]$ alias mypwru
alias mypwru='sudo pwru --filter-trace-tc --filter-track-skb --output-tuple '\''src 192.168.66.10 and dst port 80'
\'''
[jspaleta@c9-server pwru-talk]$



Matching socket buffer kernel events by kernel function

jspaleta@c9-client:~ (ssh)

```
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
HTTP/1.1 200 OK
Server: nginx/1.22.1
Date: Tue, 30 Jan 2024 22:01:41 GMT
Content-Type: text/html
Content-Length: 456389
Last-Modified: Sat, 03 Jul 2021 13:48:21 GMT
Connection: keep-alive
ETag: "60e06aa5-6f6c5"
Accept-Ranges: bytes

[jspaleta@c9-client ~]$
```

jspaleta@c9-server:~/scratch/pwru-talk (ssh)

```
[jspaleta@c9-server pwru-talk]$ mytcpdump
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
13:01:41.938799 IP 192.168.66.10.54148 > c9-server.http: Flags [S], seq 1059796200, win 32120, options [mss 1460,sackOK,TS val 3412762184 ecr 0,nop,wscale 7], length 0
13:01:41.939605 IP 192.168.66.10.54148 > c9-server.http: Flags [.], ack 2940228542, win 251, options [nop,nop,TS val 3412762185 ecr 884788147], length 0
13:01:41.939609 IP 192.168.66.10.54148 > c9-server.http: Flags [P.], seq 0:78, ack 1, win 251, options [nop,nop,TS
```

```
0xffff24559b200300      1   [pwru(400399)]          tcp_data_queue 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      1   [pwru(400399)]           tcp_queue_rcv 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      1   [pwru(400399)]      tcp_event_data_recv 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]             sock_rfree 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]            __kfree_skb 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]   skb_release_head_state 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]        skb_release_data 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]           skb_free_head 192.168.66.10:54148->192.168.66.11:80(tcp)
0xffff24559b200300      0    [nginx(4961)]            kfree_skbmem 192.168.66.10:54148->192.168.66.11:80(tcp)
```

```
0xffff24559b200d00
0xffff24559b200d00
```

Matches socket buffer owned by applications too!

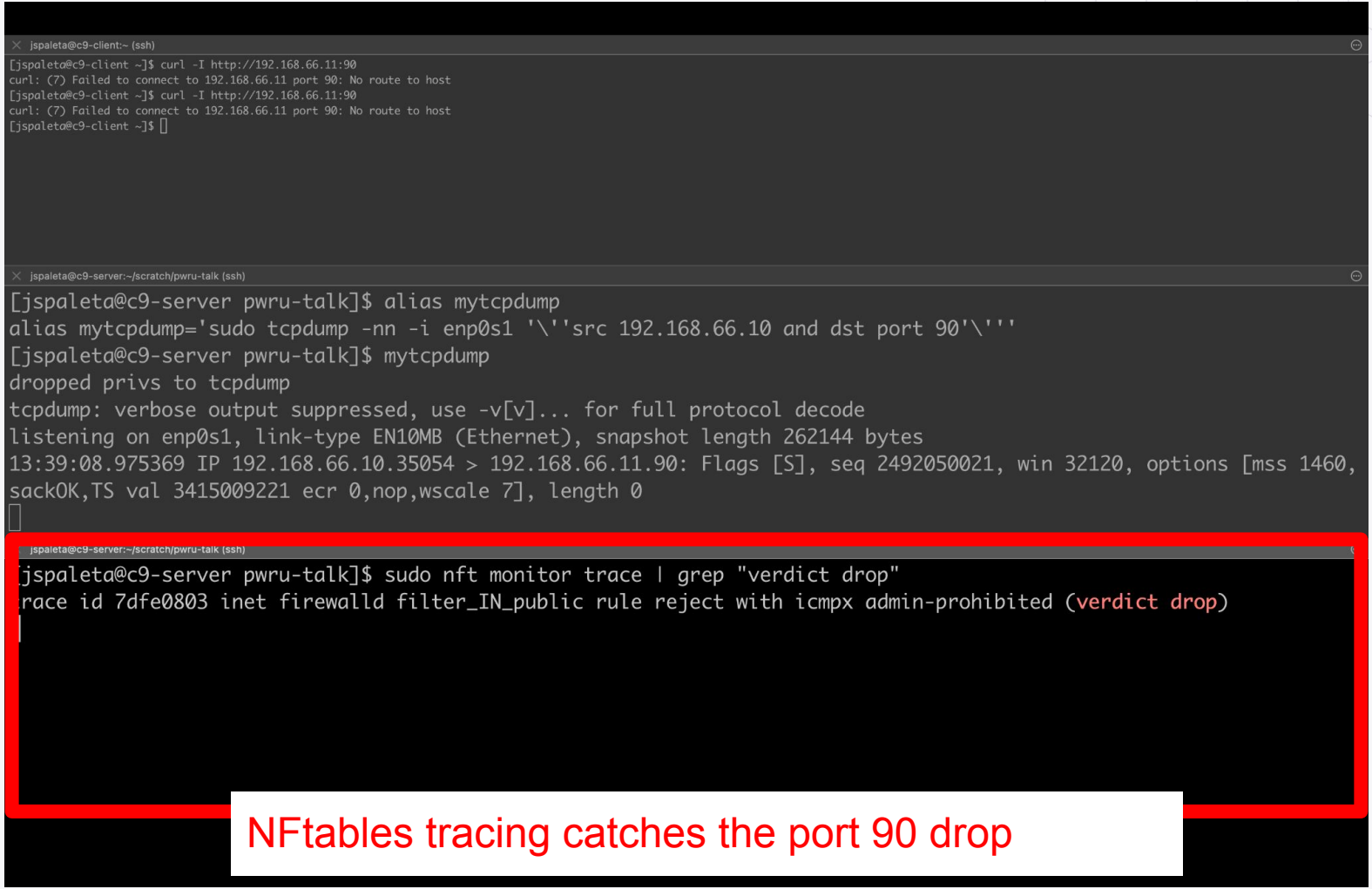
```
11:80(tcp)
66.11:80(tcp)
```

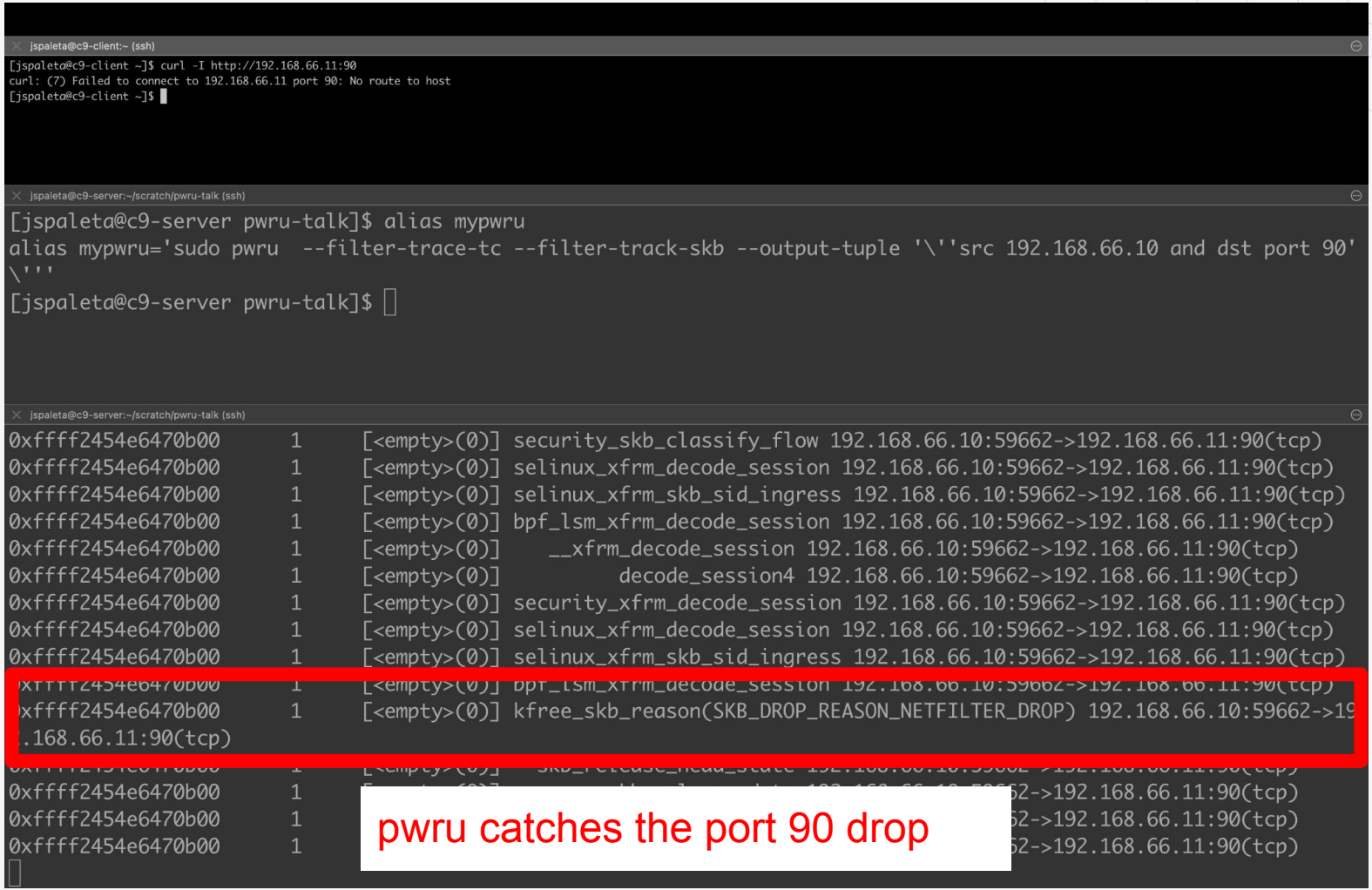
HAPPY DEMO: NFtables tracing works as expected

The scenario:

- **Same as before but add NFtable tracing rules for port 80, 90, and 8080 on server VM**

Let's try to access port 90 and watch as the NFtables rule denies access and see what pwru catches


jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11:90
curl: (7) Failed to connect to 192.168.66.11 port 90: No route to host
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11:90
curl: (7) Failed to connect to 192.168.66.11 port 90: No route to host
[jspaleta@c9-client ~]$
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
[jspaleta@c9-server pwru-talk]$ alias mytcpdump
alias mytcpdump='sudo tcpdump -nn -i enp0s1 '\''src 192.168.66.10 and dst port 90'\'''
[jspaleta@c9-server pwru-talk]$ mytcpdump
dropped privs to tcpdump
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on enp0s1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
13:39:08.975369 IP 192.168.66.10.35054 > 192.168.66.11.90: Flags [S], seq 2492050021, win 32120, options [mss 1460,
sackOK,TS val 3415009221 ecr 0,nop,wscale 7], length 0
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
jspaleta@c9-server pwru-talk]$ sudo nft monitor trace | grep "verdict drop"
race id 7dfe0803 inet firewalld filter_IN_public rule reject with icmpx admin-prohibited (verdict drop)
NFtables tracing catches the port 90 drop

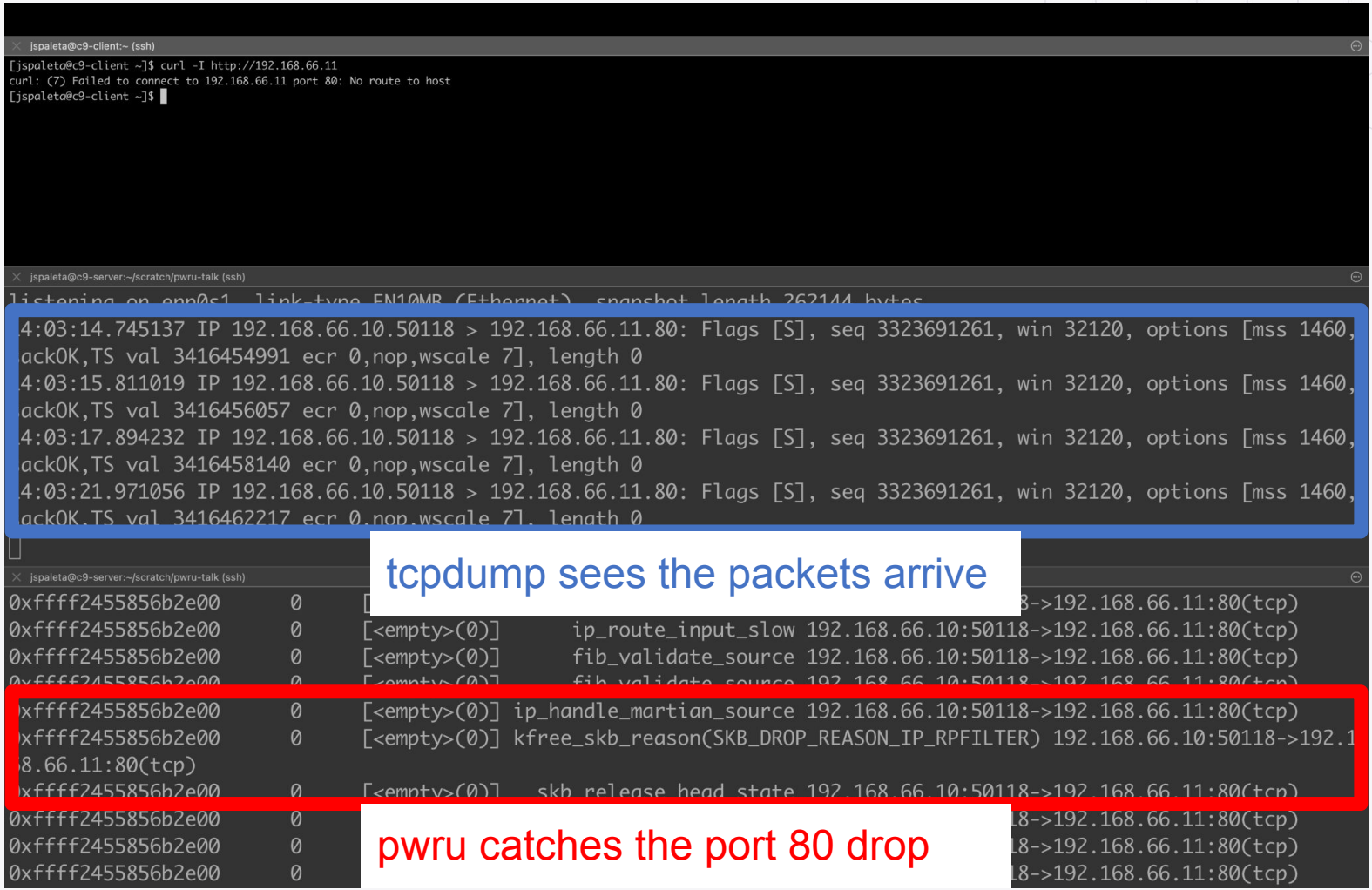

jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11:90
curl: (7) Failed to connect to 192.168.66.11 port 90: No route to host
[jspaleta@c9-client ~]$
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
[jspaleta@c9-server pwru-talk]$ alias mypwru
alias mypwru='sudo pwru --filter-trace-tc --filter-track-skb --output-tuple '\''src 192.168.66.10 and dst port 90'
\'''
[jspaleta@c9-server pwru-talk]$
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
0xffff2454e6470b00 1 [<empty>(0)] security_skb_classify_flow 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] selinux_xfrm_decode_session 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] selinux_xfrm_skb_sid_ingress 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] bpf_lsm_xfrm_decode_session 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] __xfrm_decode_session 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] decode_session4 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] security_xfrm_decode_session 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] selinux_xfrm_decode_session 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] selinux_xfrm_skb_sid_ingress 192.168.66.10:59662->192.168.66.11:90(tcp)
0xffff2454e6470b00 1 [<empty>(0)] kfree_skb_reason(SKB_DROP_REASON_NETFILTER_DROP) 192.168.66.10:59662->19
.168.66.11:90(tcp)
0xffff2454e6470b00 1
0xffff2454e6470b00 1
0xffff2454e6470b00 1
pwru catches the port 90 drop

LESS HAPPY DEMO: NFtables tracing is silent

The scenario:

- **I've disrupted communication between the client and the server via some other means that doesn't map to a NFtables drop rule.**

Let's see if pwru can provide a hint as to where the disruption is.


jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
curl: (7) Failed to connect to 192.168.66.11 port 80: No route to host
[jspaleta@c9-client ~]$
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
4:03:14.745137 IP 192.168.66.10.50118 > 192.168.66.11.80: Flags [S], seq 3323691261, win 32120, options [mss 1460,
ackOK,TS val 3416454991 ecr 0,nop,wscale 7], length 0
4:03:15.811019 IP 192.168.66.10.50118 > 192.168.66.11.80: Flags [S], seq 3323691261, win 32120, options [mss 1460,
ackOK,TS val 3416456057 ecr 0,nop,wscale 7], length 0
4:03:17.894232 IP 192.168.66.10.50118 > 192.168.66.11.80: Flags [S], seq 3323691261, win 32120, options [mss 1460,
ackOK,TS val 3416458140 ecr 0,nop,wscale 7], length 0
4:03:21.971056 IP 192.168.66.10.50118 > 192.168.66.11.80: Flags [S], seq 3323691261, win 32120, options [mss 1460,
tcpdump sees the packets arrive
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
0xffff2455856b2e00 0 [<empty>(0)] ip_route_input_slow 192.168.66.10:50118->192.168.66.11:80(tcp)
0xffff2455856b2e00 0 [<empty>(0)] fib_validate_source 192.168.66.10:50118->192.168.66.11:80(tcp)
0xffff2455856b2e00 0 [<empty>(0)] ip_handle_martian_source 192.168.66.10:50118->192.168.66.11:80(tcp)
0xffff2455856b2e00 0 [<empty>(0)] kfree_skb_reason(SKB_DROP_REASON_IP_RPFILTER) 192.168.66.10:50118->192.1
8.66.11:80(tcp)
pwru catches the port 80 drop

Demo recap

Key finding:

The pwru provided kernel function trace provides a hint to look at the system's routing configuration. The kernel's reverse path filtering is detecting a mismatch between configured outbound route and the inbound packet and dropping the inbound packet.

The culprit:

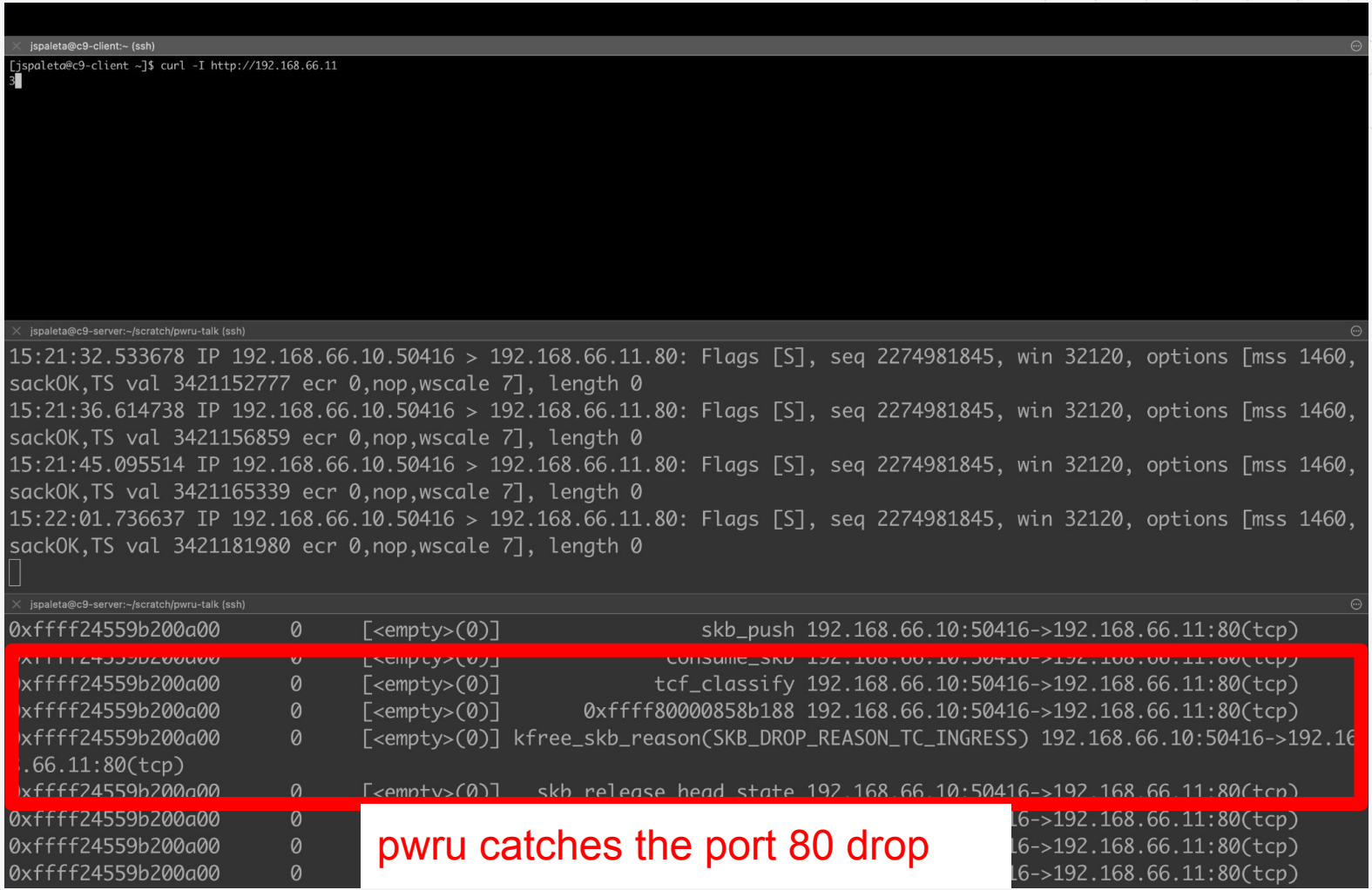
Turns on out inspection of the routes, there's an outbound blackhole route defined for the client IP address that the reverse path filtering is tripping over.

LESS HAPPY DEMO: Same but different

The scenario,

- I've disrupted communication between the client and the server via yet another means.

Let's see if pwru can provide a hint as to what is going on


jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
15:21:32.533678 IP 192.168.66.10.50416 > 192.168.66.11.80: Flags [S], seq 2274981845, win 32120, options [mss 1460, sackOK,TS val 3421152777 ecr 0,nop,wscale 7], length 0
15:21:36.614738 IP 192.168.66.10.50416 > 192.168.66.11.80: Flags [S], seq 2274981845, win 32120, options [mss 1460, sackOK,TS val 3421156859 ecr 0,nop,wscale 7], length 0
15:21:45.095514 IP 192.168.66.10.50416 > 192.168.66.11.80: Flags [S], seq 2274981845, win 32120, options [mss 1460, sackOK,TS val 3421165339 ecr 0,nop,wscale 7], length 0
15:22:01.736637 IP 192.168.66.10.50416 > 192.168.66.11.80: Flags [S], seq 2274981845, win 32120, options [mss 1460, sackOK,TS val 3421181980 ecr 0,nop,wscale 7], length 0
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
0xffff24559b200a00 0 [<empty>(0)] skb_push 192.168.66.10:50416->192.168.66.11:80(tcp)
0xffff24559b200a00 0 [<empty>(0)] tcf_classify 192.168.66.10:50416->192.168.66.11:80(tcp)
0xffff24559b200a00 0 [<empty>(0)] 0xffff80000858b188 192.168.66.10:50416->192.168.66.11:80(tcp)
0xffff24559b200a00 0 [<empty>(0)] kfree_skb_reason(SKB_DROP_REASON_TC_INGRESS) 192.168.66.10:50416->192.168.66.11:80(tcp)
pwru catches the port 80 drop

Demo recap

Key finding:

The kernel function trace provided by pwru gives a hint that the drop is because of a ingress traffic control filter.

The culprit:

Inspecting the TC ingress filters, there's an TC eBPF filter program attached to the device and is dropping inbound packets by direct action.

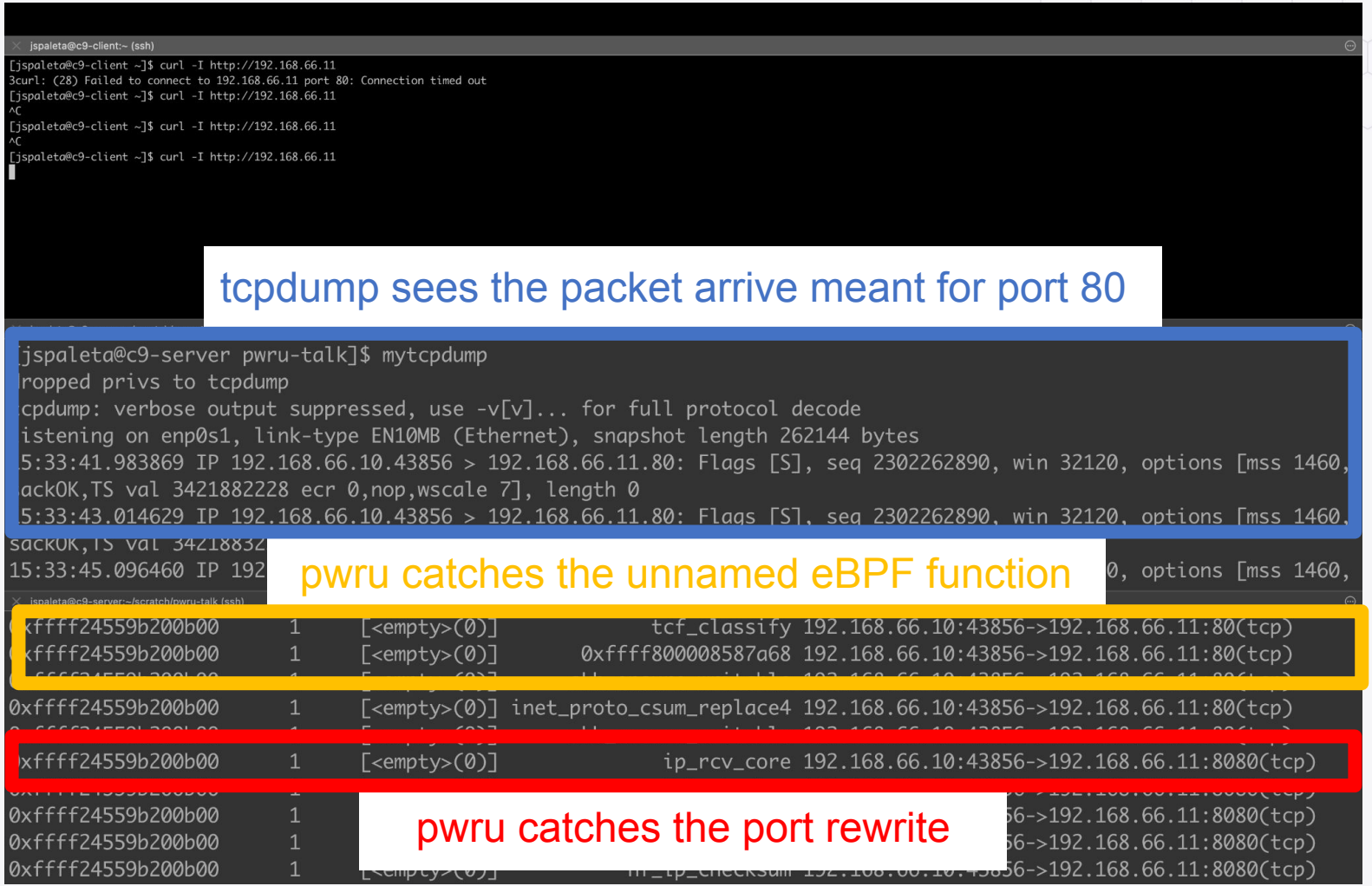
TC ingress eBPF code used here is slightly modified example from:
https://arthurchiao.art/blog/firewalling-with-bpf-xdp/#23-l4-example-drop-tcp80-packets-only

CHAOS DEMO: NFtables tracing not what is expected

The scenario:

- **NFtables trace has a deny for port 8080, even though we're connecting to server port 80**
- **Tcpdump sees the inbound port 80 packet**

Something is mangling or redirecting the packet and its not NFtables. What does pwru see?


jspaleta@c9-client:~ (ssh)
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
3curl: (28) Failed to connect to 192.168.66.11 port 80: Connection timed out
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
^C
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
^C
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
tcpdump sees the packet arrive meant for port 80
jspaleta@c9-server pwru-talk]$ mytcpdump
ropped privs to tcpdump
cpdump: verbose output suppressed, use -v[v]... for full protocol decode
istening on enp0s1, link-type EN10MB (Ethernet), snapshot length 262144 bytes
5:33:41.983869 IP 192.168.66.10.43856 > 192.168.66.11.80: Flags [S], seq 2302262890, win 32120, options [mss 1460,
ackOK,TS val 3421882228 ecr 0,nop,wscale 7], length 0
5:33:43.014629 IP 192.168.66.10.43856 > 192.168.66.11.80: Flags [S], seq 2302262890, win 32120, options [mss 1460,
sackOK,TS val 34218832
15:33:45.096460 IP 192
0, options [mss 1460,
pwru catches the unnamed eBPF function
jspaleta@c9-server:~/scratch/pwru-talk (ssh)
xffff24559b200b00 1 [<empty>(0)] tcf_classify 192.168.66.10:43856->192.168.66.11:80(tcp)
xffff24559b200b00 1 [<empty>(0)] 0xffff800008587a68 192.168.66.10:43856->192.168.66.11:80(tcp)
0xffff24559b200b00 1 [<empty>(0)] inet_proto_csum_replace4 192.168.66.10:43856->192.168.66.11:80(tcp)
xffff24559b200b00 1 [<empty>(0)] ip_rcv_core 192.168.66.10:43856->192.168.66.11:8080(tcp)
0xffff24559b200b00 1
0xffff24559b200b00 1
0xffff24559b200b00 1
pwru catches the port rewrite

Demo recap

The culprit,

There's an TC eBPF filter program attached to the device that is rewriting the inbound packets changing the destination port number.

- **pwru was able to trace into the TC eBPF program by using the --filter-trace-tc**
- **pwru was able to trace beyond that change in port using --filter-track-skb**

Without either option the pwru trace using the pcap matching filter for *dst port 80* would have ended at the the kernel's *tc_classifier* function call.

PWRU IS MAGIC!!!!

jspaleta@c9-client:~ (ssh)

```
^C
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
curl: (28) Failed to connect to 192.168.66.11 port 80: Connection timed out
[jspaleta@c9-client ~]$ ^C
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
curl: (28) Failed to connect to 192.168.66.11 port 80: Connection timed out
[jspaleta@c9-client ~]$ curl -I http://192.168.66.11
```

jspaleta@c9-server:~/scratch/pwru-talk (ssh)

```
[jspaleta@c9-server pwru-talk]$
[jspaleta@c9-server pwru-talk]$
```

jspaleta@c9-server:~/scratch/pwru-talk (ssh)

```
2024/01/30 15:53:12 Listening for events
              SKB   CPU           PROCESS                    FUNC
0xffff245588f0b400     1     [<empty>(0)]         inet_gro_receive 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]         tcp4_gro_receive 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)] __skb_gro_checksum_complete 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]          tcp_gro_receive 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]   skb_defer_rx_timestamp 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]              tpacket_rcv 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]                 skb_push 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]              consume_skb 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]            tcf_classify 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]      skb_ensure_writable 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)] inet_proto_csum_replace4 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b400     1     [<empty>(0)]      skb_ensure_writable 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b600     1     [<empty>(0)]         inet_gro_receive 192.168.66.10:43094->192.168.66.11:80(tcp)
```

pwru default options don't catch the port change

```
0xffff245588f0b600     1     [<empty>(0)]              tpacket_rcv 192.168.66.10:43094->192.168.66.11:80(tcp)
0xffff245588f0b600     1     [<empty>(0)]                 skb_push 192.168.66.10:43094->192.168.66.11:80(tcp)
```

BONUS DEMO: Diagnosing Cilium in a Kubernetes Cluster

The scenario:

- **multi-node Kind cluster running in Linux VM**
- **Using Cilium as its CNI**

pwru provides diagnostic visibility into all inner-cluster communications.

BONUS DEMO: recap

Big take away

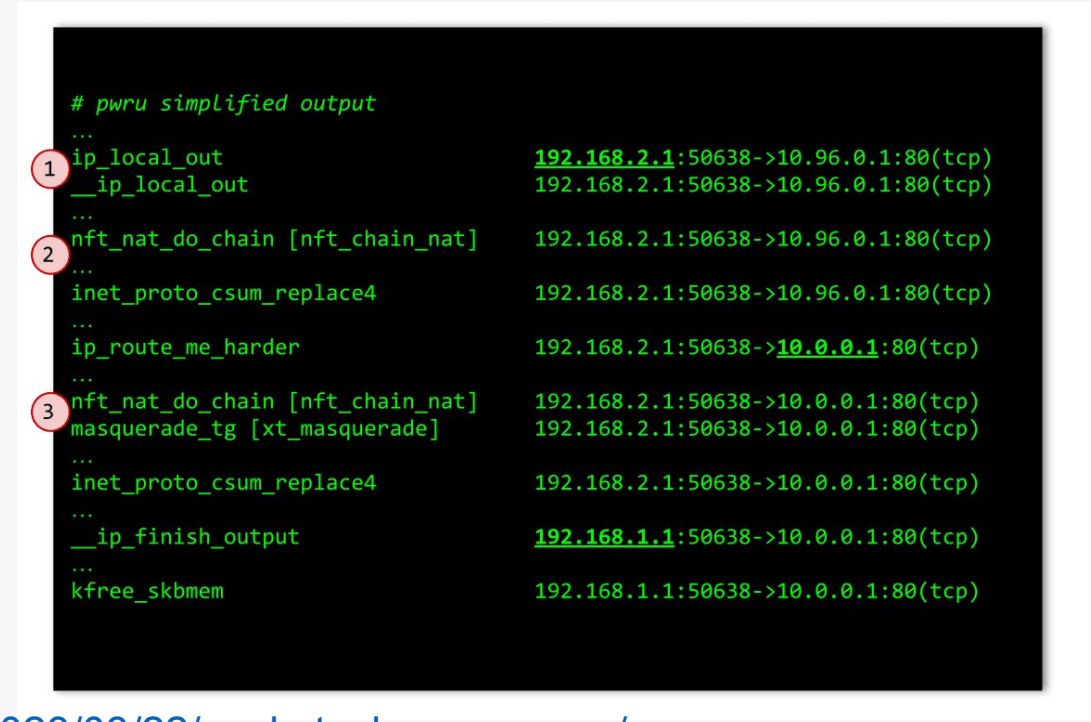
pwru doesn't need to be run inside of each network namespace in use by the nested containers in the cluster.

pwru can optionally filter by namespace for more control.

It's not just for eBPF programming bugs!

Linux codebase issue seen in the wild:

unexpected src IP address mangling in the linux masquerading logic in certain configuration corner cases



Blog Ref: https://cilium.io/blog/2023/03/22/packet-where-are-you/

Practical Knowledge: Important filter options I used

--filter-trace-tc

Let's you trace into eBPF programs loaded as TC filters.

Using this I was able to see the exact function call that changed the port number.

--filter-track-skb

Let's you track socket buffers, even if the packet information changes and no longer matches your initial pcap filter.

This let me trace after the port change and see the trace resolve to a netfilter deny

These do come with a cost, both these options involve installing additional Kprobes

Practical Knowledge: Useful pwru output options

--output-tuple

Highlights L4 information in the socket buffers for human readability. Great for an initial diagnostic view from a network admin perspective.

--output-meta

Outputs socket buffer metadata: mark, protocol, mtu, interface, packet length
useful when L4 information isn't enough and your concerned about malformed packet/socket buffer mangling.

--timestamp string ("relative", "absolute")

Add a timestamp column to output.
"relative" can really help make sense of trace boundaries when matching multiple packets.

This sounds amazing! What's the catch?

There are some limitations and caveats

pwru can impact performance (kprobes aren't free)

- pwru does let you limit which kernel functions it attaches Kprobes using RE2 regexp strings, but to make use of this feature you already need to sort of have an idea of what the problematic function calls are. I

- older kernels are slow at (a/de)ttaching many kprobes
  My testing of Fedora 38 vs CentOS 9 has several minutes difference to init/tear down the kprobes. Newer kernels have a multiple kprobe attach/detach mechanism that really speeds things up.

pwru can't yet trace xdp programs

- Example: pwru can't trace *xdp-filter* actions yet (patches welcome!)

Wrap Up

I think PWRU is a great devops tool for diagnosing Linux networking issues.

- Network operators (hopefully this audience) get pcap filtering centric view of packet flows through the Linux kernel.
- Linux/eBPF developers (probably in another room right now) get actionable function call tracing information they can use to pinpoint deficiencies in kernel and eBPF networking code.
- And there's still lots of opportunity to help make it better!

Thanks for coming to my talk!

Check out the pwru repo and give it a try:
https://github.com/cilium/pwru