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 pwrue exists

How it makes use of eBPF

Live(?) demos of pwrue 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 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 you would need to look at?
- How do you know what you don’t 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.
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 pwruln 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
data! 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

pwrut 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 pwrut 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, let’s 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-server pwr-u-talk]$ alias mytcpdump
alias mytcpdump='sudo tcpdump -i enp0s1 "src 192.168.66.10 and dst port 80"'
[jspaleta@c9-server pwr-u-talk]$ 

[jspaleta@c9-server pwr-u-talk]$ alias mypwr
alias mypwr='sudo pwr --filter-trace-tc --filter-track-skb --output-tuple "src 192.168.66.10 and dst port 80"'
[jspaleta@c9-server pwr-u-talk]$ 
```
Matching socket buffer kernel events by kernel function
Matches socket buffer owned by applications too!
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 pwrul catches
NFtables tracing catches the port 90 drop
pwruru 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.
tcpdump sees the packets arrive

pwrulu catches the port 80 drop
Demo recap

Key finding:
The pwrut 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
pwruru 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 an 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 it's not NFtables. What does pwru see?
tcpdump sees the packet arrive meant for port 80

pwrhu catches the unnamed eBPF function

pwrhu 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.

- pwr u was able to trace into the TC eBPF program by using the \texttt{--filter-trace-tc}
- pwr u was able to trace beyond that change in port using \texttt{--filter-track-skb}

Without either option the pwr u trace using the pcap matching filter for \texttt{dst port 80} would have ended at the kernel's \texttt{tc\_classifier} function call.

\textbf{PWRU IS MAGIC!!!!}
pwrU default options don't catch the port change
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 pwrul 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

**pwrul can impact performance (kprobes aren't free)**
- pwrul 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.

**pwrul can't yet trace xdp programs**
- Example: pwrul 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