bpfilter,
pare-feu Linux à la sauce eBPF

Quentin Monnet
<quentin.monnet@netronome.com>
@qeole
We make SmartNICs for data centers.

Features include vRouter, firewall, transparent HW offload for OvS... or eBPF!
eBPF: Programmability in the kernel

C source code
bpf_prog.c

ELF-compiled eBPF
bpf_prog.o

LLVM/clang

User program
(tc / ip / bcc tools...)

Verifier

JIT

User program

Maps

• Array
• Hashmap
• LPM
• ...

BPF program attached and run

Q. Monnet | bpfilter
eBPF hooks

Networking
- tc (traffic control)
- XDP (network driver)
- Lightweight Tunnel Encapsulation

Socket

Tracing / Monitoring
- Kprobe/Uprobe
- Tracepoint
- Perf event

Others to come?

Networking
- XDP (network driver)

Socket

Networking
- tc (traffic control)

Tracing / Monitoring
- Kprobe/Uprobe
- Tracepoint
- Perf event

Others to come?

eBPF

Networking
- XDP (network driver)

Socket

Networking
- tc (traffic control)

Tracing / Monitoring
- Kprobe/Uprobe
- Tracepoint
- Perf event

Others to come?
bpfilter, a new back-end for iptables in Linux, based on eBPF

- RFC posted to Linux network development (netdev) mailing list, mid-February 2018
- Code by David Miller (networking subsystem maintainer), Alexei Starovoitov and Daniel Borkmann (BPF tree maintainers)
- Not merged yet, everything that appears here is susceptible to change!
“BPF and firewalls? Reminds me of something.”

bpfilter not to be confused with...

- **xt_bpf** module (attach BPF program to Netfilter hook; rather an extension of xtables, and relies on classic BPF)

```
iptables -A INPUT \ 
  -p udp --dport 53 \ 
  -m bpf --bytecode "14,0 0 0 20,177 0 0 0,12 0 0 0,7 0 0 0, \ 
  64 0 0 0,21 0 7 124090465,64 0 0 4,21 0 5 1836084325, \ 
  64 0 0 8,21 0 3 56848237,80 0 0 12,21 0 1 0,6 0 0 1, \ 
  6 0 0 0,\" \ 
  -j DROP
```

(Matches a DNS query for “example.com”, credit goes to Cloudflare)

- **nftables**, designed as iptables/xtables successor
- **BPF in nftables** (posted to netdev in reaction to bpfilter)
- **NFP firewall on NetBSD** with classic BPF (≠ eBPF) and JIT-compiling
The `iptables` binary is left untouched

Rules are translated into an eBPF program, attached to e.g. XDP

bpfilter.ko: new kind of kernel module, here for rule translation
- ELF file running in user space!
- Based on user mode helpers (UMH)
- But shipped and built from kernel tree
- Should be compatible with `modprobe`, `modinfo`, etc.
- Run in a special thread, full privileges and in root namespace

Several objectives for this new kind of module
- Easier to develop, to debug, to test
- Reduce attack surface, cannot crash the kernel
- Clear decoupling between data plane (kernel) and control planes (user space)

bpfilter.ko module communicates with the kernel via `bpf()` syscall
bpfilter overview

bpfilter.ko
(ELF kernel module)

Rule translation

Userspace

Kernel

iptables

Netfilter subsystem

bpfilter() syscall

Verifier

JIT

BPF program attached and run

Kernel
(TC, generic XDP)

Driver
(native XDP)

NIC
(hardware offload)

wait()

translate & attach eBPF

Special thread
The benefits of bpfilter

- JIT compilation on x86_64, arm64, ppc64, sparc64, mips64, s390x, arm32
- Straightforward hardware offload on compatible NICs
- BPF verifier: security and safety
- User space ELF modules
- Existing BPF tooling; possibly writing rules in C?
- eBPF more and more used in the kernel, possibilities for integration with other subsystems?
Example usage, from the PoC

# ./bpfilter.ko  # Should eventually use modprobe

# iptables -t filter -A INPUT -i eth1 -d 10.0.0.4/32 -j DROP
# iptables -L

    Chain INPUT (policy ACCEPT)
    target prot opt source           destination
    DROP   all  --  anywhere          10.0.0.4

    Chain FORWARD (policy ACCEPT)
    target prot opt source           destination

    Chain OUTPUT (policy ACCEPT)
    target prot opt source           destination
Rule translated to an eBPF program

```
# bpftool prog dump xlated id 1337

0: (bf) r9 = r1
1: (79) r2 = (u64 )(r9 +0)
2: (79) r3 = (u64 )(r9 +8)
3: (bf) r1 = r2
4: (07) r1 += 14
5: (bd) if r1 <= r3 goto pc+2
6: (b4) (u32) r0 = (u32) 2
7: (95) exit
8: (bf) r1 = r2
9: (b4) (u32) r5 = (u32) 0
10: (69) r4 = (u16 )(r1 +12)
11: (55) if r4 != 0x8 goto pc+9
12: (07) r1 += 34
13: (2d) if r1 > r3 goto pc+7
14: (07) r1 += -20
15: (61) r4 = (u32 )(r1 +12)
16: (55) if r4 != 0x200000a goto pc+1
17: (04) (u32) r5 += (u32) 1
18: (61) r4 = (u32 )(r1 +16)
19: (55) if r4 != 0x400000a goto pc+1
20: (04) (u32) r5 += (u32) 1
21: (55) if r5 != 0x2 goto pc+2
22: (b4) (u32) r0 = (u32) 1
23: (95) exit
24: (b4) (u32) r0 = (u32) 2
25: (95) exit
```

E.g. instruction #19: check on 0x400000a, which is “ntohl(10.0.0.4)”
Comparison for simple packet drop between iptables, nftables, bpfilter

Setup:

- One single iptables or nftables rule (as in previous example)
- 64 byte long packets

Hardware:

- Intel® Xeon® CPU E5-2630 v3 @ 2.40 GHz
  - Single CPU, 8 cores 16 threads
- Netronome Agilio CX, 1 × 40 Gbps Ethernet

*Many thanks to my colleague David Beckett for running the tests!*
Performance test results

- 40 Gbps

**Q. Monnet | bpfilter**

- **iptables (legacy):** 10.9 Mpps
- **nftables:** 12.4 Mpps
- **bpfilter (host driver XDP, JIT):** 38.7 Mpps
- **bpfilter (hardware offload):** 59.5 Mpps

**40 Gbps**
68 replies on the thread, many comments from Netfilter people

- **Performance**
  - Many speed improvements from nftables over iptables
  - JIT-compiling, XDP hook, hardware offload: way faster, whereas Netfilter in general was not good enough and failed to get a wide adoption

- **Replication of iptables back-end**
  - Users’ assumptions regarding the behaviour of iptables, 100% perfect replication is impossible
  - Will make efforts to have the same, on as many use cases as possible

- **Why iptables in the first place?**
  - Maintainers trying to phase out the legacy interface, why not base bpfilter on nftables instead?
  - iptables widely spread and will remain for at least a decade, better improve performance and ease maintenance
Security

- Security concerns, mostly about the new ELF module mechanism
- Safety and security through BPF verifier; ELF module no less secure than kernel modules.

What about eBPF?

- Not so much deployed as of today
- Deployed in most major providers, used more and more in the kernel, for various taks

... but, really, eBPF?

- “BPF has many usability problems”
- Simply not true
What happens next?

- PoC must be refined to get a more complete, optimised version
- The proposal needs to be accepted by the community
- bpfilter very likely to be accepted: backed by influential developers
- Early March: follow-up for nftables, with a common intermediate representation with iptables
- Early March, too: repost of the patch for the new ELF kernel modules
- Next:
  - bpfilter merge to the kernel?
  - nftables support?
  - User space tooling update?
  - More hardware offload?
Questions?

Additional resources:

RFC on netdev mailing list “net: add bpfilter”, sent by Daniel Borkmann
https://www.mail-archive.com/netdev@vger.kernel.org/msg217095.html
and following emails of this thread

LWN.net: BPF comes to the firewalls
https://lwn.net/Articles/747551/

LWN.net: Designing ELF modules
https://lwn.net/Articles/749108/

Resources on BPF — Dive into BPF: a list of reading material
https://qmonnet.github.io/whirl-offload/2016/09/01/dive-into-bpf/

Netronome website
https://www.netronome.com/

We’re hiring!