bpfilter: packet filtering with BPF and nftables
Quentin Deslandes

- Software Engineer @ Meta, working from France
- Member of the Linux Userspace team: we aim to make significant contributions to upstream userspace projects
- Working on bpfilter since September 2022

qde@naccy.de - github.com/qdeslandes
01 FILTERING PACKETS IN 1998

About iptables

- Created by Rusty Russels in 1998
- 1998’s iptables is not 2024’s iptables
- It defines a structure we are familiar with:
  - Tables to decide whether to NAT, filter, or mangle
  - Chains to attach rules to the networking stack
  - Rules to filter packets on specific criteria
How does it work?

- What is the workflow?
  - Read and validate command line arguments
  - Uses `getsockopt()` to retrieve the whole ruleset from the kernel
  - Modify the ruleset from userspace and send it back using `setsockopt()`
- The data is sent to / received from the kernel in a binary format (i.e. `struct ipt_entry`)
Let’s talk about the caveats

- 1998 was a long time ago, technology evolved (a lot) since then
- Packet filtering and firewall rules become more and more complex
- `iptables`’s architecture is not suited for modern network requirements
- If your firewall can’t keep up: you drop packets
- Can we improve the situation?
“Let there be eBPF”

Alexei Starovoitov, probably
Tutorial: speeding up iptables

1. Define a new UMH module
2. Plug the module to net/ipv4/ip_sockglue.c
3. Translate iptables rules to BPF programs
4. Enjoy!

As it turns out, I should retire that tweet, since now we also need someone who knows eBPF, XDP, nftables ...
So far, so good

• Alexei Starovoitov, Dave Miller, and Daniel Borkmann created the first version of \texttt{bpfilter}.
• Dmitrii Banshchikov tried to implement the BPF bytecode generation
  - Stopped at v2
• I tentatively submitted a v3
Relocation to userspace

• The architecture was difficult to maintain
• bpfilter was tied to the kernel development process
• The project being under heavy development, it’s difficult to get timely review
New bpfilter

- Complete refactor of the project
- Two main parts now:
  - libbpfilter
  - bpfilter daemon
nftables 101

- Packet filtering framework
- Replaces iptables
- Uses Netlink, not {get,set}sockopt()

```
Userspace
nft add rule inet $TABLE $CHAIN tcp dport 22 drop

Kernel
ip $TABLE $CHAIN
  [ meta load l4proto => reg 1 ]
  [ cmp eq reg 1 0x00000006 ]
  [ payload load 2b @ transport header + 2 => reg 1 ]
  [ cmp eq reg 1 0x00001600 ]
  [ immediate reg 0 drop ]
```
**libbpfILTER**

- Lightweight library
- Aims to ease integration to `bpfilter`
- Data-independant

```c
nft add rule inet $TABLE $CHAIN tcp dport 22 drop

ip $TABLE $CHAIN
    [ meta load 14proto => reg 1 ]
    [ cmp eq reg 1 0x00000006 ]
    [ payload load 2b @ transport header + 2 => reg 1 ]
    [ cmp eq reg 1 0x0001600 ]
    [ immediate reg 0 drop ]

struct bf_request {
    metadata: {...},
    nft_bytecode:
        ip $TABLE $CHAIN
            [ meta load 14proto => reg 1 ]
            [ cmp eq reg 1 0x00000006 ]
            [ payload load 2b @ transport header + 2 => reg 1 ]
            [ cmp eq reg 1 0x0001600 ]
            [ immediate reg 0 drop ]
}

bpfilter daemon
```
The daemon

- Listens on a Unix Domain Socket
- Perform the heavy lifting:
  - Translation of the client-specific format
  - Generation of the BPF programs
  - Management of the BPF programs
Translation

- Dedicated front-end for each client
- Convert client-specific data into an internal format
- Allows for code reuse during bytecode generation
Generation

• This is the compilation step
• Outputs 1 or more BPF programs
• Creates a prologue and an epilogue which are specific to the BPF program type
• Rules are unrolled at BPF bytecode
03 BPFILTER NOW

Loading

- Use BPF subsystem to attach a program
- Up to 1 program per interface
- Program replacement is atomic: no down time

```
struct bf_codegen {
    program_type: BPF_PROG_TYPE_XDP
    hook: ...
    rules: [...],
    programs: [
        struct bf_program {},
    ]
}
```
```bash
vm > bci shell -r

~

ping 10.211.55.5

qdeslandes@qdeslandes-mbp:~$ sudo ./bpfilter --transient --verbose --no-iptables
```
Benchmarks

- 2 servers connected through a 10G link.
- Using Linux’ pktgen to generate packets at ~10Gbps.
- Increase the number of rules to increase overhead.
- Last rule drop all UDP packets
Current features and capabilities

- iptables and nftables are available (from a fork)
- Filter packets based on:
  - Source/destination IP address and/or port
  - L3 protocol
  - Source network interface.
- Collecting statistics
- Support XDP, TC, BPF_NETFILTER programs
- Supports kfuncs, BPF helpers, BPF dynamic pointers, custom functions...
Future work

• IPv6 (in progress)
• Sets support
• Partial rules re-generation
• Generic client
• CGroups support
Resources

- bpfilter repository: [github.com/facebook/bpfilter](https://github.com/facebook/bpfilter)
- nftables fork: [github.com/qdeslandes/nftables/tree/bpfilter_support](https://github.com/qdeslandes/nftables/tree/bpfilter_support)
- iptables fork: [github.com/qdeslandes/iptables/tree/bpfilter](https://github.com/qdeslandes/iptables/tree/bpfilter)
- Status report and project’s progress: [naccy.de](https://naccy.de)
- Email: [qde@naccy.de](mailto:qde@naccy.de)