bpfilter: a BPF-based packet filtering framework
Quentin Deslandes

• Software Engineer @ Meta, member of the Linux Userspace team.
  - Contributing and promotion userspace open source tools at Meta.
• Working on a BPF-based packet filtering framework.
• This talk is about firewall.

gde@naccy.de - github.com/qdeslandes - twitter.com/Naccyde
• Originally developed as a UMH by Alexei Starovoitov, Daniel Borkmann, and David Miller.
• Catches `iptables-legacy` calls to `getsockopt()` and `setsockopt()`.
• Translates filtering rules into XDP BPF programs.
• Abandoned around 2021.
• Developed as a userspace daemon since earlier this year.
- Client is linked to libbpfilter.
- Communication between the library and the daemon goes through a Unix Domain Socket.
- bpfilter daemon will translate the client-specific data into a generic format, and generate the BPF bytecode.
- The daemon manages the BPF programs' lifetime.
libbpfilter
- Linked to the client, uses its data structures.
- Provides a lightweight interface to the daemon.
- Client-specific data is translated by the daemon.

```c
- ret = setsockopt(handle->sockfd, TC_IPPROTO, SO_SET_REPLACE, repl,
-            sizeof(*repl) + repl->size);
+ #if defined(ENABLE_BPFILTER) && !defined(_LIBIP6_TC_H)
+   if (handle->use_bpf)
+      ret = bf_ipt_replace(repl);
+   else
+ #endif
+ ret = setsockopt(handle->sockfd, TC_IPPROTO, SO_SET_REPLACE, repl,
+            sizeof(*repl) + repl->size);
```
IPTABLES

**iptables and bpfilter**

- `iptables` read/write the whole ruleset from/to `bpfilter` through the Unix Domain Socket.
- Programs are fully re-generated on rule change.
- Uses BPF_NETFILTER hooks (6.4).
- Support counters, IP, port, and protocol filtering.

```bash
iptables -I INPUT -p UDP -i eth0 -j DROP --bpf
```
nftables and bpfilter

- nftables sends Netlink messages directly to bpfilter through the Unix Domain Socket.
- Parsing is performed in the daemon using libnl.
- Uses XDP hook to offload nftables rules.
bpfilter: BPF bytecode generation

- Write BPF instructions directly into a memory buffer.
- Fixups to modify the bytecode once the generation is complete.
- Support for custom functions and stubs.
- Support XDP, TC, and BPF_NETFILTER program types.
- Uses BPF dynamic pointers to access packet data.
- Per-rule-interface packets and bytes counters.
// BF_ARG_1: counters map file descriptor.
EMIT_FIXUP(program, BPF_CODEGEN_FIXUP_MAP_FD, BPF_MOV64_IMM(BF_ARG_1, 0));

// BF_ARG_2: index of the current rule in counters map.
EMIT(program, BPF_MOV32_IMM(BF_ARG_2, bf_list_size(rules)));

// BF_ARG_3: packet size, from the context.
EMIT(program,
    BPF_LDX_MEM(BPF_DW, BF_ARG_3, BF_REG_CTX, BF_PROG_CTX_OFF(pkt_size)));
EMIT_FIXUP_CALL(program, BPF_CODEGEN_FIXUP_FUNCTION_ADD_COUNTER);

EMIT(program,
    BPF_MOV64_IMM(BF_REG_RET, program->runtime.ops->get_verdict(policy)));
EMIT(program, BPF_EXIT_INSN());
Therefore, the output of the command `sudo bpftool prog list` is as follows:

```
75: netfilter  name bpfltr_02000002  tag 43bf0d20e76f4997  gpl
   loaded_at 2023-09-12T11:51:28+0200  uid 0
   xlated 832B  jited 752B  memlock 4096B  map_ids 25

[...]

86: netfilter  name bpfltr_04000003  tag caf36bf4fc15b318  gpl
   loaded_at 2023-09-12T11:51:34+0200  uid 0
   xlated 832B  jited 752B  memlock 4096B  map_ids 36
```

Finally, the output of the command `sudo ls /sys/fs/bpf` is as follows:

```
bpfltr_m_02000002  bpfltr_m_03000002  bpfltr_m_04000002  bpfltr_p_02000002  bpfltr_p_03000002  bpfltr_p_04000002
bpfltr_m_02000003  bpfltr_m_03000003  bpfltr_m_04000003  bpfltr_p_02000003  bpfltr_p_03000003  bpfltr_p_04000003
```
Performance

- Dropping incoming packets on `bpfilter` and `iptables`.
- What about `nftables`?
Resources

- bpfilter repository: [github.com/facebook/bpfilter](https://github.com/facebook/bpfilter)
- iptables PoC: [github.com/qdeslandes/iptables/tree/bpfilter](https://github.com/qdeslandes/iptables/tree/bpfilter)
- Nftables PoC: [github.com/qdeslandes/nftables](https://github.com/qdeslandes/nftables)