Application / Service / Task specific netfilter rules

NFWS 2016

Daniel Mack, Red Hat
<daniel@zonque.org>

Current ways for matching traffic

We currently have ways to filter network traffic based on

- Destination, source, address type, …
- Source port, destination port, …
- CPU, owner, socket, …
- …

What we'd like to see

- Apply netfilter rules to individual tasks (processes/PIDs), or group of tasks (applications/services)
- Make task matches orthogonal to other rule details, such as destination/source port/IP
- Anticipated use cases are not limited to filtering, but cover accounting as well (this comes for free in the netfilter framework)

For instance …

- “Allow all outbound traffic from Firefox”
- “Allow inbound traffic to all ports bound by postgresql, if it comes from a 10.10.0.0/16 IP”
- “Count all traffic sent and received by nginx”
- “Tell me which application used up all my mobile bandwidth”

Grouping tasks

- Applications usually consist of multiple tasks (processes/PIDs)
- Grouping them into logical units is essential for applying resource limits and handle them as “applications”
- systemd uses the terms “unit” and “service” (systemd is however not the only userspace component that is in need for such mechanisms)

Examples for task grouping

```
CGroup: /system.slice/nginx.service
        ├─23285 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
        ├─23286 nginx: worker process
        ├─23287 nginx: worker process
        ├─23288 nginx: worker process
        └─23289 nginx: worker process
```

```
CGroup: /system.slice/system-postgresql.slice/postgresql@9.4-main.service
        ├─ 8062 postgres: db1 db1 ::1(44389) idle
        ├─15150 /usr/lib/postgresql/9.4/bin/postgres -D /var/lib/postgresql/9.4/main
        ├─15152 postgres: checkpointer process
        ├─15153 postgres: writer process
        ├─15154 postgres: wal writer process
        ├─15155 postgres: autovacuum launcher process
        ├─15156 postgres: stats collector process
        ├─20014 postgres: db2 db2 ::1(42452) idle
        └─28625 postgres: db3 db3 ::1(35615) idle
```

cgroup matches

- cgroups v1 (indirection through net class):

```
# echo 1 > /sys/fs/cgroup/…/net_cls.classid
# iptables -A OUTPUT -m cgroup --cgroup 1 …
```

- cgroups v2 (directly via cgroup path):

```
# iptables -A OUTPUT -m cgroup –path … …
```

So – all is good?

- No, unfortunately not
- The current implementation only works reliably for **egress** traffic
- For **ingress**, rules are not executed unless the stream is established
- Hence, the current implementation leaks initial packets and is unusable

iptables-extensions(8)

IMPORTANT: when being used in the INPUT chain, the cgroup matcher is currently only of limited functionality, meaning it will only match on packets that are processed for local sockets through early socket demuxing. Therefore, general usage on the INPUT chain is not advised unless the implications are well understood.

Possible solutions

- How can this be fixed?
- 3 competing solutions have been implemented

Solution #1: early demux

- Add early demux support to xt_cgroup
- No userspace change required

Problems:

- Does not work for multiple targets (ie, UDP)
- Does currently only work for simple protocols (not for SCTP, DCCP etc)

Solution #2: postpone checks via flag

- In case we cannot make decision on the packet at ingress time, set a flag and redo the check later
- No userspace change required

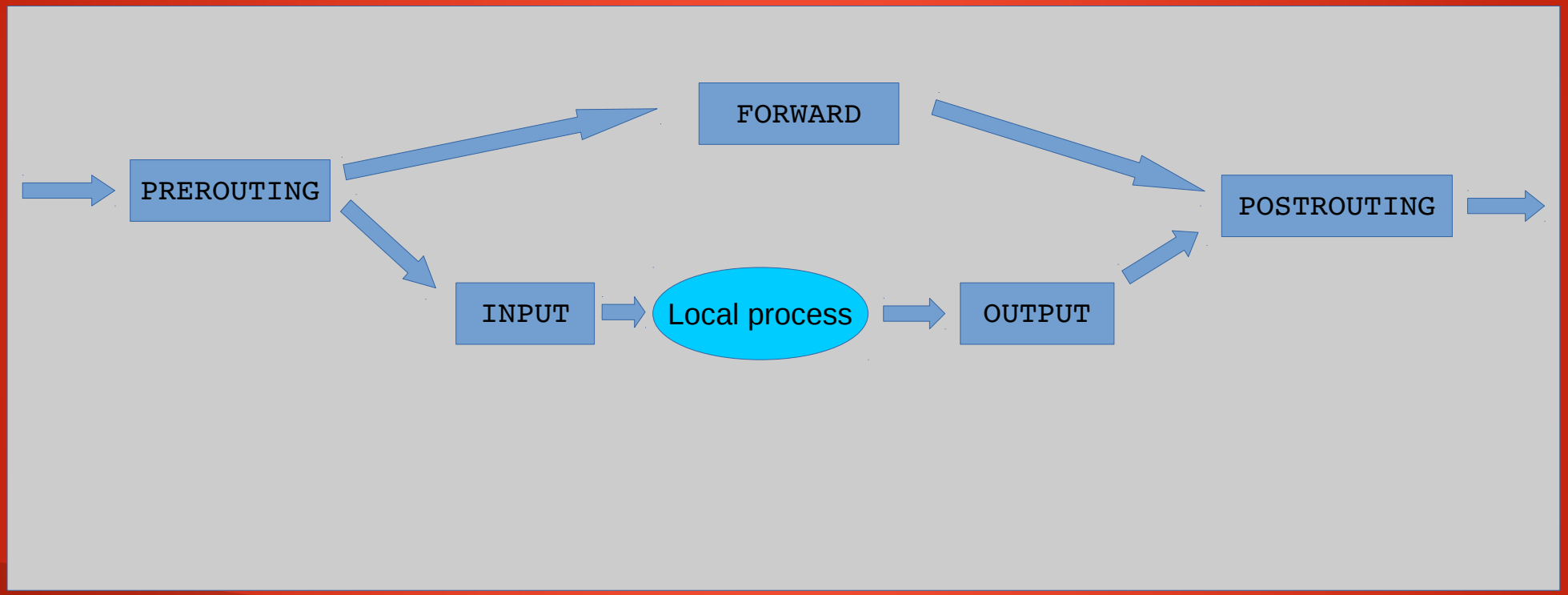
Problems:

- Flagged packets will travel INPUT rules twice

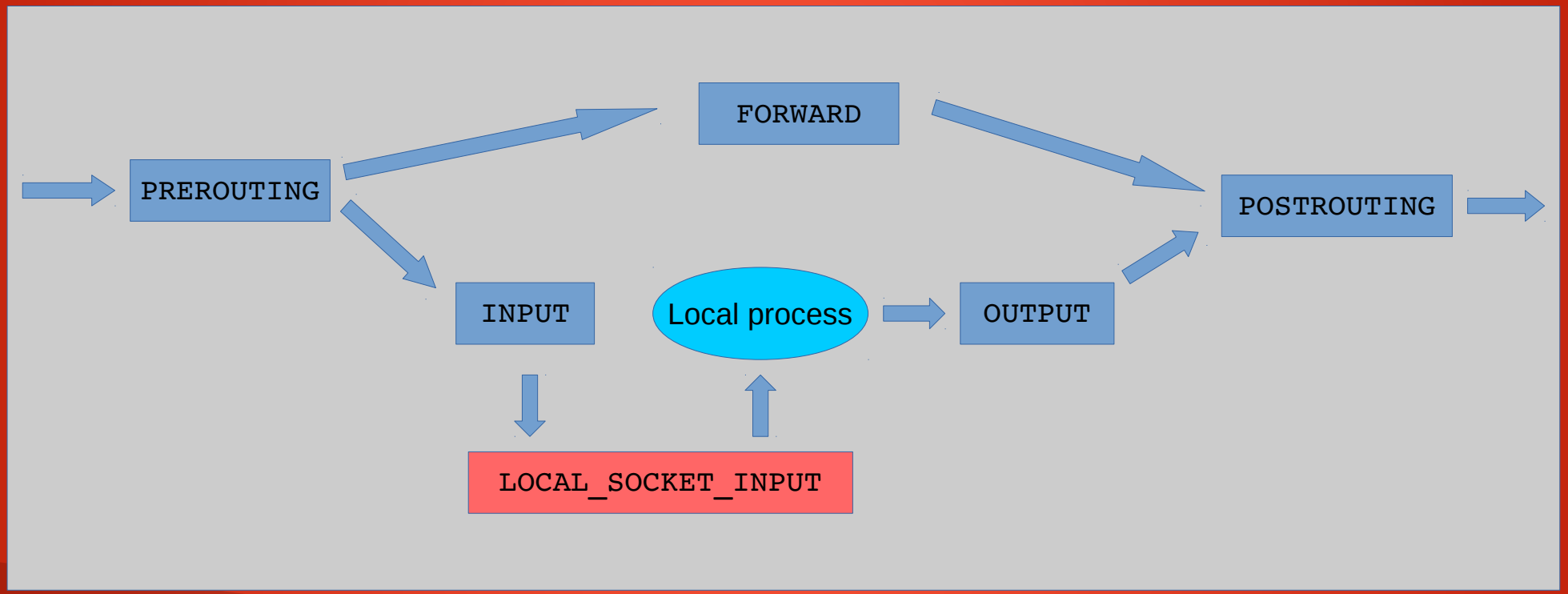
Solution #3: LOCAL_SOCKET_IN

- Add a new chain type that is executed after demux, but before forwarding to local process
- Needs a backwards-compatible change in userspace
- Needs hookups in several protocols
- Chain is run for every recipient, ie, possibly multiple times for a single UDP packet

Current layout



LOCAL_SOCKET_INPUT



Possible issues with LOCAL_SOCKET_INPUT

- It adds load to the input path if enabled (same as LOCAL_INPUT, but per receiving task)
- Semantic change: not all packets that are ACCEPTed in LOCAL_IN will make it to LOCAL_SOCKET_IN
- What about raw sockets?

Thoughts?

Let's discuss the pitfalls and possible alternatives!

Thanks!