IPSec/NF Workshop 2023 Summary, (NF) technical debt

Florian Westphal
4096R/AD5FF600 fw@strlen.de
80A9 20C5 B203 E069 F586
AE9F 7091 A8D9 AD5F F600

Sep 2023

IPsec workshop

- IPsec on Android (N. Harold)
- IP-TFS / RFC 9347 (C. Hopps)
- flowtable (forward path sw bypass) extensions for IPsec (P. Neira, S. Klassert)
- IKEv2 support for per-queue Child SAs (S. Klassert et al)
- Misc IKE/Signalling topics
- BEET revival/resurrection
- `PF_KEY` deprecation, `ESPINUDP_NON_IKE` removal

Bummer: `PF_KEY` details leaked into IPsec offload API

NFWS 2023

- bpf+nf
  - make existing ct related kfuncs available to nf bpf progtype (D. Xu)
  - bpfilter, current status (Q. Deslandes)
- Register tracking infra (Pablo)
- Misc Discussion topics
  - userspace API, cookie attribute, grammar ambiguities in parser
  - bug and patchwork backlog
  - Existing test cases, lowlevel test API (Pablo)
- Multiple complaints about design issues (myself)

netfilter: introspection

poor interacticon of netfilter+dropwatch/drop monitoring tools

- drop anywhere in a netfilter subsystem? blames `nf_hook_slow()` due to `NF_DROP` design
- needs to be improved

netfilter: introspection, plan

- add internal verdict: `NF_DROP_REASON`, not exposed to userspace
  - Works like `NF_STOLEN`, but provides errno to stack
  - incremental conversion:
    `s/return NF_DROP/kfree_skb_reason(); return NF_DROP_REASON(errno)/`
- will at least pinpoint *where* drops happen (nftables, iptables, conntrack helper, ...).
- Keep back but in mind for later:
  - ideally dropmonitor could tell rule/chain too, but 16 bit reason value is not enough context
  - pcpu scratchpad to stash chain/table info?
  - tracepoint for `NF_xxx` return values in table eval loop?

Other items cooking on nf side

- bridge netfilter removal: would like to do it, not realistic – too popular
- kernel without arptables and ebtables support (via distro)?
  - minor Kconfig plumbing
  - arptables-nft/ebtables-nft would continue to work
- eventually same treatment for ip/ip6tables (setsockopt bits)
- most matches and targets would stay around
- code removal, e.g. rbtree set backend

Conntrack on bridge

- best case: you'll eventually get a `WARN_ONCE` splat from conntrack core
- worst case: UaF crash
- The problem is with first skb of a flow
- Once connection is in hash table all is fine

Root cause: `skb_clone`'d reference same `struct nf_conn`

Conntrack assumptions for new packets

1. newly allocated `nf_conn` is owned exclusively by the skb
2. skb will *NOT* leave ip stack between pickup and table insertion
   - for local: `OUTPUT` and `POSTROUTING` (confirm is last hook)
   - for remote to local: `PREROUTING` and `INPUT` (confirm is last hook)
   - for forward: `PREROUTING` and `POSTROUTING` (confirm is last hook)

Also means:

- no locking when (re)allocating/writing to conntrack extension area
- `nf_conn` is not in in hash table, so other cores cannot find it
- After insertion (confirm), extension area is no longer reallocated

Conntrack + bridge

Mostly worked just fine, even though bridge has to clone often:

```
for_each_port_in_bridge(p, br)
  deliver_clone(skb, p)
```

Several ways to turn this into problem:

- Macvlan: handles b/mcast in work queue
- bridge netfilter: you can now use nfqueue

Clone gets moved to different core with not-yet-committed `nf_conn`? Race begins

How to fix this?

- don't want to add code to `skb_clone` path
- don't want to add locking in conntrack

Best proposal I could think of so far:

- register additional nfhook at bridge `INPUT` and `POSTROUTING`
- from those hooks: make a full copy of `nf_conn`, iff:
  1. `nf_conn` not yet in hash (check `nf_conn->status`) AND
  2. `nf_conn` refcount is 1

→ more hacks to keep bridge nf afloat. Problems:

- Can't deal with all extension types, some contain list pointers
- No idea yet what to do in that case (mark as untracked?)
- what happens if the "original" skb is dropped? Can't confirm in that case.

Unprivileged netns, CVEs

- as normal user: `unshare -Unr` → you get uid 0
  - theoretically restricted to one network namespace, full sandbox
  - but in practice a large swath of code paths now become accessible
- large number of bugs in `nf_tables` and other subsystems that allow privilege escalation
- fix: `sysctl user.max_user_namespaces=0`
- but breaks software, notably chrome

Unprivileged netns, CVEs (2)

Huge influx of issues. Some bug classes are generic

- also occur in e.g. packet classifiers, xfrm, etc
- some are unique to `nf_tables`: transaction handling
  - error unwinding
  - contradicting requests in transaction
  - underlying api has more features/capabilities than whats used by nftables itself
- most of these bugs are rather old
- some are unique to `netfilter`: mostly local DoS
  - memory exhaustion
  - large rulesets/graphs
- also ancient problems, but no practical/realistic solutions in sight

→ New sysctls to disable nftables/iptables in unpriv netns by default?

Counterargument: tech debt is everywhere

- virtually all bug fixes are now CVEs
  - ... if one assumes malicious local user
  - ... fuzzers find plenty of those
  - in core networking, fs, mm, ...
- kernel not ready for unpriv userns
- Assuming DoS: plenty of rope here
  - Cascade of stacks of virtual devices: ,,team-on-team-on-team-on-bond...“

Time to start breaking things?

- Netfilter already doing this to a small degree
  - 512MB max size of (classic) iptables rulesets (100m rules) since 2018
  - `CLUSTERIP` target
  - reduce `nf_tables` api capabilities (things nft doesn't do)
    - add/delete from anonymous sets
    - add/remove flags in same transaction, etc.
- Does xfrm policy need to deal with ipv6 mobility header?
- Is there really a use case for team-on-team-on-team?
- adding sysctl limits not universal solution
  - more variance, different behaviours
  - ,,This bug only occurs if you set sysctls x/y/z to combo ...“
  - we're reluctant to remove them again (but not consistenly so)