Conntrack stuff

Florian Westphal

NFWS 2016, Amsterdam

June 2016

4096R/AD5FF600 fw@strlen.de
1C81 1AD5 EA8F 3047 7555
E8EE 5E2F DA6C F260 502D

Conntrack + net namespaces: current -next state

- single table, single size limit
- but maxlimit counts per net namespace
- 1k namespaces: max * 1k connections possible

several bugs when assuming rogue netns

- resource accounting: Its easy to fill up table from a container
- any container can just set its `conntrack_max` to ludicrous value
- could check vs. init namespace, but ...
- maxlimit reached? Create new ns, fill that up too ... (repeat)
- zones direction support added local DoS hole, from `deedb5903` commit log:

  ```
  Note that zone identifiers can not
  be included into the hash mix anymore
  ```

No clue how to fix this

Seems netns require trusted environment

misc. netfilter netns stuff

- stupid netns init dependencies
  - netns exit path (netlink event sk already zapped? crash)
  - error unwinding (did not reach nfqueue netns init? crash)
- both issues are fixed
- ... by making 3 ptrs per netns, even though all namespaces have same content

break

Questions so far?
Next up: conntrack extensions

Re-thinking conntrack extensions

- Connection is represented by `struct nf_conn`
- fixed-size, allocated from a kmem cache
- contains essential info, such as tuples, timer, refcnt
  - also tcp conntrack state (seq/ack, window, etc)
- has ptr to `struct nf_ct_ext` for extensions
- extension blob is kmalloc'd and free'd via rcu

Conntrack extensions

10 extensions exist at the moment

- helper support (e.g. ftp)
- NAT
- accounting (packet/byte stats)
- conntrack event cache
- ... and more

Pros and cons

- pro:
  - don't have to allocate mem for rarely-used extensions
  - allows non-fixed-size extensions
- con:
  - has some overhead: 40 bytes per conntrack just for metadata
  - need one extra deref to access data
- $\rightarrow$ move some frequently used extensions into the main structure (zone for instance)

some numbers ...

struct	description	size in bytes
nf_conn	base structure	288 (320 w. align)
nf_ct_ext	extension head	40
nf_conn_help	helper base struct	$>$ 24
nf_conn_nat	nat	32
nf_conn_seqadj	sequence adjustment	24
nf_conn_ecache	event cache	24
nf_conn_counter	accounting	32
nf_conn_tstamp	timestamp	16
nf_conn_timeout	timeout	8
nf_conn_synproxy	synproxy	12
nf_conn_label	conntrack labels	8 to 24

New extension grows `nf_ct_ext` struct by 2 byte

Move nat?

- pro:
  - could get rid of PREALLOC code
  - container use-cases normally require nat for external connectivity
  - will most likely NOT increase `nf_conn` size (hwalign padding)
  - 32 byte seems big, BUT it would be 16 (single-ll, no-need for ct backptr anymore)
- cons:
  - need to init unconditionally
  - seems someone needs to be volunteered to give this a try

conntrack gc

- TCP established default timeout is huge (5days)
- conntrack `early_drop` only tosses `!ASSURED`
- less chance to find evictable entry after recent table merge

Proposal: add conntrack gc worker. If we can't allocate new conntrack:

1. do `early_drop` (search adjacent buckets for `!ASSURED`
2. schedule worker
3. worker can walk entire table (bh on)
   - 3.1 also add 'soft timeout' (min lifetime) sysctl, e.g. 1 hour
   - 3.2 allow fast-recycle after that

per-conntrack spinlock

- used e.g. in tcp tracker code
- is this absolutely needed?
  - No state transitions in most cases (established)
  - But ack/seq etc updates expected
- just re-use normal per-bucket lock for this?
- not worth doing atm, just 2 bytes and we have to dirty CL anyway (refcnt)

get rid of fastpath refcnt

- Normally refcnt is one (owner by timer)
- every ct lookup bumps refcnt
- also every `skb_clone` or copy
- last put: kfree
- could avoid it for most cases: see `DST_NOREF`
- would need to switch to `call_rcu` – can't use `SLAB_DESTROY_BY_RCU` anymore, as lookup can't use `atomic_inc_not_zero`, and timer can fire in between