netfilters connection tracking subsystem

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

Red Hat

netdev 2.1, Montreal, April 2017

connection tracking

- flow tracking by addresses of endpoints (L3/L4, e.g. ip + port, ip + GRE call id, . . .)
- split in layer 3 tracking (ip, ipv6) and layer 4 tracking (tcp, udp, sctp, ICMP, . . .)
- L4 tracker is agnostic of lower protocol
- L4 trackers attempt to keep state, e.g. tcp: tracks state, checks sequence numbers. Example:
  - new tcp packet? SYN bit set?
  - tcp sequence number in expected window?
  - unacknowledged data? $\rightarrow$ adjust timeout
  - rst? fin? $\rightarrow$ delete connection and/or adjust timeout
- NAT is built on top – conntrack itself never alters packets
- uses netfilter hooks to look at packets as they come in/leave

conntrack events

userspace can subscribe to ct events:

```
$ conntrack -E
[UPDATE] tcp  6 432000 src=192.168.0.7 dst=10.. sport=3...
[UPDATE] tcp  6 120 FIN_WAIT src=192.168.0.7 dst=10.16...
[UPDATE] tcp  6 60 CLOSE_WAIT src=192.168.0.7 dst=10.16...
[NEW] udp    17 30 src=10.26.2.2 dst=192.168.0.7 sport=5..
[NEW] tcp     6 120 SYN_SENT src=192.168.0.7 dst=.. sport=6
[UPDATE] tcp  6 60 SYN_RECV src=192.168.0.7 dst=192..
[UPDATE] tcp  6 432000 ESTABLISHED src=192.168.0.7 ...
[DESTROY] tcp 6 src=202:8071:.. dst=202:26f0.. sport=34284
```

NEW event sent once entry is placed in conntrack table
it is possible to restrict what events are generated (CT target)

common misconceptions

- `iptables -A INPUT -m conntrack --ctstate ...` doesn't do connection tracking
- ... rather, it tests conntrack state (`skb->nfct->status == ...`)
- same for `nft ct state ...`: no lookup of any kind
- conntrack doesn't look at socket states, only packets

conntrack states

- ESTABLISHED – packet matches existing entry and l4 tracker checks pass
- NEW
  - first packet of a connection (no previous record)
  - a new connection entry is created after failed lookup
  - ... but NOT placed in main conntrack table
  - ... only done after packet traversed all hooks (iptables) in INPUT or POSTROUTING
  - ... in conntrack speak, the entry is now confirmed (in main table)
- RELATED – same as NEW, except it somehow relates to another existing connection
  - ICMP error, and the header inside matches an existing connection
  - conntrack helper created an entry in the "expectation table"
- UNTRACKED – packet was intentionally not tracked (ipv6 neigh discovery for instance)

INVALID – packet not seen or rejected by l4 trackers (`skb->_nfct is 0`)

connection tracking helpers

some protocols are harder to track/NAT, e.g. SIP or FTP

- kernel module monitors "control channel", e.g. tcp port 21
  - can add 'expectations', i.e. if new connection is coming from S to D on port P, then mark as `RELATED`
  - also can apply NAT if needed
  - allows doing FTP, SIP etc. without opening up many ports or adding lots of 1:1 nat translations
- best-effort only, e.g. no tcp stream reassembly in kernel
- in-kernel XML/ASN.1 parsing required for sip, h323, etc.
  - might be preferable to use real proxies
  - its possible to add expectations from userspace
  - e.g. could implement transparent SIP proxy that only processes call setup messages, and allows actual calls to just pass through

main conntrack table

- hash table, using rcu (lookups are lockless) and hashed locks (i.e. add/delete is parallel if they occur in other part of the table)
- table has a fixed size (`net.netfilter.nf_conntrack_buckets`) and fixed upper limit (`net.netfilter.nf_conntrack_max`)
- no automated growth, initial sizing based on available memory
- each entry is hashed twice (original+reply) to deal with nat

conntrack extensions

idea: keep data of rarely used features outside of main `nf_conn` struct

- pro:
  - don't have to allocate mem for rarely-used features
- con:
  - overhead: 40 bytes per conntrack just for metadata
  - need one extra deref to access data

Examples: helper, counter, tstamp, ...

NAT

- built on top of connection tracking
- NAT mappings are set up at conntrack creation time
- ... which is why iptables 'nat' table only "sees" first packet of flow
- one extra hash table: nat bysource table
  - used to ensure addr:port is unique when adding new mapping
- all connections have nat mapping once a nat hook is active

overflow handling

`nf_conntrack: table full, dropping packet` main assumption: most entries are non "assured"

- assured – flag set by l4 protocol tracker at certain point (tcp: 3whs completed)
- over limit?
  1. search up to 8 buckets for non-assured entry
  2. destroy it and allocate new conntrack entry in its place
- otherwise, drop the new packet

problems

- Only non-assured entries can be early-dropped
- no way to know if new packet is 'more important' than any other state table entry
- can't toss random entries: would kill valid connections
- doesn't play nice with nat/pat
- what about overflow w. legitimate traffic patterns?

suggestions (1)

- remove very strange conntrack error handling
  - packet invalid? `NF_ACCEPT` (let user decide what to do in iptables ruleset)
  - can't alloc conntrack/over limit? `NF_DROP` (user can't change this behavior)
  - can this be fixed in a backwards-compatible fashion?

doesn't solve table exhaustion problem for all cases, e.g. can't NAT non-tracked packets

suggestions (2)

- add `early_drop` function to the l4 trackers
  - e.g. could prefer evicting tcp flow in `WAIT` state in favor of new connection
- TCP established default timeout is huge (5days)
  - add 'soft timeout' (min lifetime) sysctl, e.g. 5 minutes and allow fast-recycle after this
  - do periodic ack probing/keepalives (i.e., elicit RST if connection was closed)
  - adaptive timeouts like *BSD? Combine `CT --timeout` with match on (used) table size?
  - early evict if no nat?

problem: under flood, even 1 minute is too long
helps with peers that don't close properly

conntrack – summary

- mature code base
- lots of features
- but still room for improvements:
  - overflow handling
  - free extensions via kfree, not via rcu
  - remove variable sized extensions?