improvements to conntrack table overflow handling

Florian Westphal
Red Hat
fw@strlen.de

Abstract

connection tracking keeps a state table that uses the addresses of communication endpoints, e.g. ip address and port number, or ip address and GRE call id to identify packets belonging to the same connection. netfilters Network address translation facilities also make use of connection tracking.

The next section gives a high-level overview, then current handling of connection tracking table overflow is described. The later sections deal with possible improvements to table exhaustion problems.

connection tracking engine in the stack

The conntrack engine uses the hooks provided by the netfilter framework, which are also used by the `ip(6)tables` and `nftables` tools.

The hooks are placed at particular decision points in the internet protocol stack of the linux kernel to allow inspection and, if needed, mangling of packets as they pass through the network stack. For more information on the hook infrastructure provided by the netfilter subsystem, please see[6].

The **prerouting** hook is located before the ip routing decision that decides if a packet has to be forwarded or is going to be delivered locally. All inbound IP packets will traverse this hook.

The **input** hook is for packets that will be delivered to the local machine, whereas **forward** is for packets that have to be forward to another machine[1].

Packets generated on behalf of local processes will pass through **output**. The last hook point, **postrouting**, is passed by both locally originating packets and forwarded ones. This diagram shows the hooks and their ordering:

conntrack hooks everywhere except FORWARD. Prerouting sits before the route lookup, all incoming packets traverse this hook. Conntrack uses this to perform the initial lookup in the conntrack table. It extracts the layer 3 and layer 4 connection information (ip addresses, ports), and checks if the flow tuple – the combination of the two endpoints ip address and the layer 4 information (usually, port numbers) – is already stored in the table.

[1] this may include containers or virtual machines running on the same host

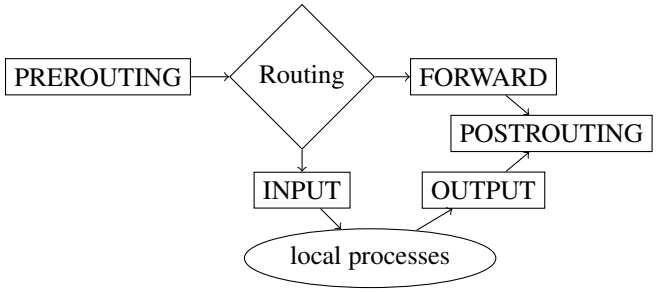

Figure 1: netfilter hooks

If there is a match, the lookup returns the `nf_conn` connection tracking structure, which is then assigned to the skb[2].

A similar hook is placed into the **output** hook list, to look up flows where the local machine is the initiator or responder (as opposed to a router that merely forwards packets).

In case no connection is found, the higher-level protocol tracker, such as tcp, can decide that the packet is initiating a new connection or that the packet is related to an existing connection in some way (such as ICMP error messages where the inner header matches a known tuple). In both cases, a new `nf_conn` structure is allocated.

The new structure is associated with the packet, but it is not yet committed to the main connection tracking table – this is delayed until the packet has passed all iptables rules. This has two advantages:

1. as a newly allocated `nf_conn` entry is only exposed to the CPU processing the packet, several types of conntrack manipulations, such as initializing nat transformations, do not need to acquire locks.
2. We don't need to acquire the global locks needed to insert the entry into the table in case the packet gets dropped by the packet filter rules.

Committing a new entry to the main table is handled by "confirm" hooks, one in **input**, one in **postrouting**. These are called after ip(6)tables rules have been consulted.

The remaining two hooks concern handling of connection tracking helpers, which are dedicated modules that deal with

[2] structure that contains packet payload and packet metadata, such as device and routing information for the packet

higher-level protocols such as ftp or sip which need special handling to track them in a stateful fashion. These are also located in the **input** and **postrouting** hook points. In brief, a helper module reads packet payload data to discover announcements of expected connections, such as extracting the address and port number of the data connection from a ftp

```
PORT 192,168,1,1,5,6\r\n
```

reply on the ftp command connection. Helpers then can add the seen address/port pair to the so-called expectation table. When the ftp data connection is initiated, it will then be automatically set as `RELATED`, for easier filtering. The expected connection will also follow a possible network address translation of the master conntrack entry without a need to configure explicit nat rules for it.

Historically, the connection tracking helpers were always active once the corresponding kernel modules were loaded. Nowadays, users should explicitly assign them for security reasons[1].

This is done by using nftables `ct set helper hname` or iptables `CT` target with the `--helper` option. This also has the advantage that one can e.g. ask to track ftp on a different port, such as 2121[3].

Its important to note that all connection tracking helpers are best-effort only. netfilter does not perform tcp stream reassembly, for instance, so checking payload split across packets does not work. Furthermore, keeping the helpers in kernel can require complex and error prone parsing of higher-level protocols such as XML or ASN1, so it is a good idea to perform these tasks in userspace instead. For this purpose userspace can add entries to the expectation table, see the section about ctnetlink for more information.

conntrack states

There are 5 distinct connection tracking states:

1. NEW. first packet of a connection (no previous record), a new conntrack was allocated.
2. RELATED. Same as new, but the packet is somehow related to an existing connection. This is true for ftp data connections, or ICMP path MTU errors where the inner header matches existing tuple.
3. ESTABLISHED. The packet matches an existing entry and the l4 tracker checks of the packet passed. In the tcp case, this for example means that the tcp sequence numbers were within the expected window.
4. INVALID. The packet is not associated with a conntrack entry.

   This happens for example when the connection tracker deems that the packet doesn't fit the criterion for a new packet. This happens for instance when `net.netfilter.nf_conntrack_tcp_loose=0` and a tcp packet neither matches an existing entry nor has the `SYN` flag set.

[3] the old method involved "magic" module parameters to change the port number the helper monitors by default

struct	description	size in bytes
nf_conn	base structure	256
nf_ct_ext	extension head	40
nf_conn_help	helper base struct	> 24
nf_conn_nat	nat	8
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	16

Table 1: current extensions and sizes on 64bit architectures

5. UNTRACKED. This means that either the user has added an explicit filter rule to not track the packet using nftables `notrack` keyword or the iptables `CT --notrack` target, or with certain ICMP6 packets such as neighbour discovery.

conntrack extensions

Connection tracking extensions are essentially just extra structures containing information about a connection, but such information is considered to be used infrequently, so it is not placed in the main `nf_conn` struct to save space.

They are allocated on demand, for instance the GRE or ftp helpers use this to store extra information required for tracking that is not needed for normal flows.

data structures

The connection tracking engine uses a single hash table to store conntrack tuples. Every connection is added twice, once with the reverse tuple. This needed for network address translation, where the tuple of one direction doesn't match the other one. See [2] for more details on how tuples are linked within the table.

Lookups (read access) in the table are lockless, insertions can occur in parallel by different processors provided inserts occur in different parts of the table, i.e. the hash value not only determines the slot where the tuple is inserted, but also the lock that is responsible for protecting insertions and deletions for this bucket.

The table has a fixed size (tune-able via `net.netfilter.nf_conntrack_buckets`) and a fixed upper limit (`net.netfilter.nf_conntrack_max`). Once this number of connections is reached, the kernel will log the message `nf_conntrack: table full, dropping packet` and all further new connection requests are dropped until the table is below the maximum limit again. This is unfortunate, especially in DoS scenarios, the next section discusses this in more detail.

Lockless lookups are achieved by use of RCU[3]. `nf_conn` structures are allocated from a kmem cache using the `SLAB_DESTROY_BY_RCU` feature. This means that a conntrack entry that is freed can be re-allocated instantly

```
$ 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=60..
[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=3428 [UNREPLIED]
```

without waiting for a rcu grace period to elapse. Readers deal with this by re-checking the tuple after obtaining a reference on the entry. If the tuple has changed, the entry was freed right before the lookup, in this case the reader has to indicate that the lookup did not yield a result.

It is also possible (albeit unlikely), for an `nf_conn` entry to move from one hash bucket to another one. This can be detected by a so-called nulls marker, every hash bucket of the conntrack main hash table has a unique marker at the end of the chain, so in case the lookup finds the "wrong" nulls value the lookup has to be re-tried.

Timeouts are handled passively – each `nf_conn` structure stores a timeout value (in jiffies). On every conntrack lookup all `nf_conn` structures in the bucket list whose timeout stamp is in the past are removed.

In order to catch timed-out entries on idle systems a work queue is used to periodically scan the table for old entries.

ctnetlink

Conntrack provides a netlink[5]-based protocol for userspace to interact with the connection tracker.

userspace can subscribe to ct events:

ctnetlink events can be used for flow accounting in userspace. The extension infrastructure contains extensions to allow per-connection packet and byte traffic accounting, and precise timestamping of when connections were created and closed.

ctnetlink also allows to delete or even add new entries or expectations from userspace. Entries that get added via ctnetlink can also set up nat binding independent from the nat table.

Adding expectations can be used to implement connection tracking helpers in userspace. For instance one could use netfilters TPROXY facilities to redirect control protocol traffic to a dedicated daemon/proxy, process them there and then use ctnetlink to add the tuple of the expected data connections into the expectation table. For instance a SIP proxy would only process SIP control messages such as invites while the actual calls are forwarded by the kernel.

packet floods

Netfilter conntrack treats entries that have not (yet) seen two-way communication specially – they can be evicted early if the connection tracking table is full. If insertion of a new entry fails because the table is full, the kernel searches the next 8 adjacent buckets of the hash slot where the new connection was supposed to be inserted at for an entry that hasn't seen a reply[4]. If one is found, it is discarded and the new connection entry is allocated.

When dealing with tcp syn floods from random source address, most entries can be early evicted because the tcp connection tracker sets the "assured" flag only once the 3-way handshake has completed.

In the udp case the assured flag is set once a packet arrives after the connection has already seen at least one packet in the reply direction. In other words, request/response traffic does not have the assured bit set and can therefore be early-dropped at any time.

The table can still get exhausted, however – either because the table is improperly sized for the workload, or because timeouts are too large.

Lowering timeouts might not be a universal solution, however – especially when using NAT/PAT the conntrack entry holds the nat transformation/mapping information, so destroying such entries breaks connectivity in case end hosts resume transfer after an idle period.

conntrack error handling

Currently, conntrack error handling appears to be inconsistent. Packets that are deemed invalid (e.g. out of window) will be accepted by conntrack, i.e. it is up to the user to decide to drop such `INVALID` packets in the nftables or iptables rule sets[5].

When a connection tracking entry cannot be allocated due to limits being reached however, conntrack DROPs the packet instead. Changing this so conntrack doesn't drop in this case anymore doesn't help in all cases (NAT depends on connection tracking), but it might still help in cases where NAT is not used.

In current kernels connection tracking uses a garbage collection work queue to catch entries that have timed out. It would be easy to extend this gc worker to perform additional checks in stress situations, the following section discusses some ideas.

early drop via l4 trackers

Instead of only doing the fast `early_drop` scan from the packet path, the kernel could instead query the layer 4 trackers about a connection.

For instance, one could also consider the layer 4 state and prefer to also evict tcp flows that are in `WAIT` state (after seeing a tcp `FIN` in one direction).

Furthermore, it would be easy to add a new "soft timeout", which will evict an entry if it was idle for this period only while the table is full. This would allow to keep large TCP default established timeouts while at the same time making compromises in stress situations.

custom timeout policies

Another idea is to extend the concept of per-connection timeout policies. Currently the kernel already supports timeout policies that allows to attach custom timeout policies to a flow, e.g. one can set lower timeouts for incoming tcp connections on port 53 while allowing lower timeouts for outgoing connections.

[4] these are also called non-assured entries

[5] ct state invalid drop, -m conntrack –state INVALID -j DROP

Instead of attaching these timeouts via `-j CT --timeout policyname` based on properties of the matched packet[6], one could add a way to query the occupancy ratio of the conntrack table from the packet path, e.g.

```
\verb+-m conntrack --table-above 75 \
  -j CT --timeout ...+.
```

This would permit adaptive timeouts very similar to the ones offered by BSDs `pf` tool[4] based on usage without making this a fixed/hard coded property of connection tracking.

connection probing

Instead of just closing a connection without warning, it would be possible to actively probe endpoints similar to what is done by the `SO_KEEPALIVE` mechanism described in the tcp manual page[7] by injecting packets after the connection has been idle for some time. This would allow to detect connections where an endpoint has disconnected without shutting its connections down. Conntrack could then evict these connections automatically. It also improves reporting of such connections from a accounting perspective as we will not have to wait for the (possibly large) established timeout to elapse.

Summary and future work

The previously discussed measures help when dealing with un-cooperative peers that do not close properly, e.g. by ignoring FIN packets, or by re-sending FIN packets to keep the connection open – conntrack would be able to free these resources in a more timely manner without user intervention.

Aside from the suggestions in this paper there are other items that might be worth looking at:

- free conntrack extension area with `kfree` instead of `kfree_rcu`. This requires an audit of the code base to ensure that all read accesses to the extension area occur after obtaining a reference on the conntrack entry.
- check if we can remove the need for variable-sized extensions. It might be possible to instead provide a scratch area similar to `skb->cb[]` for helpers to use. Most connections do not use a helper and allocation of the extension area would be safer as we could add compile-time assertions on the maximum size of the extension area.

Aside from the helper extension all other extensions have fixed sizes. Removing the need for this would allow to add compile-time assertions for possible overflows of the extension offset calculations. It would also all minor simplicifcation of the extension allocation path as we no longer need to pass/consider a variable size.

References

[1] Leblond, E. 2012. Secure use of iptables and connection tracking helpers. blog. https://home.regit.org/netfilter-en/secure-use-of-helpers/.

[2] Magnus Boye. 2012. Netfilter Connection Tracking and NAT Implementation. https://wiki.aalto.fi/download/attachments/69901948/netfilter-paper.pdf.

[3] McKenney, P. E. 2003. Using RCU in the Linux 2.5 kernel. *Linux Journal* 1(114):18–26. Available: `http://www.linuxjournal.com/article/6993` [Viewed March 17, 2017].

[4] OpenBSD. 2017. pf.conf manual page. *OpenBSD documentation*. Available: `http://man.openbsd.org/OpenBSD-6.0/man5/pf.conf.5`, [Viewed March 20, 2017].

[5] Pablo Neira Ayuso, Rafael M. Gasca, Laurent Lefèvre. 2010. Communicating between the kernel and user-space in Linux using Netlink sockets. *Software: Practice and Experience* 40(9).

[6] Pablo Neira Ayuso. 2006. Netfilter's Connection Tracking System. *In :LOGIN;, The USENIX magazine, Vol. 32, No. 3, pages 34-39*.

[7] 2017. Linux man pages project: tcp - tcp protocol. *man 7 tcp*. `https://www.kernel.org/doc/man-pages/`.

Author Biography

Florian Westphal is a contributor to the Linux kernel network stack, in particular netfilter. He is also a member of the netfilter core team.

[6] e.g. the tcp destination port