Contributors	**The first thoughts**
Jan Rekorajski <baggins@pld.org.pl>	LOG
Imran S. Patel <imran@cs.ucsb.edu> <Imran.Patel@nokia.com>	extension headers (2001.08.) – it has lost length
Brad Chapman <kakadu_croc@yahoo.com>	copied CT mangle 4 the 5 hooks
Maciej Soltysiak <solt@dns.toxicfilms.tv>	hl, fuzzy, random, nth, time, quota HL cleanups
Herve Eychenne <rv@wallfire.org>	man limit
Andras Kis-Szabo <kisza@securityaudit.hu>	icmpv6(fix), mac(fix), eui64, headers(2002.12.), frag, rt, ah, esp, hbh, dst LOG(fix, tunnel) testsuite tools test packets initial man pages SMP (Andreas Herrmann) save/restore, userspace & all
Helpers	
Harald Welte <laforge@gnumonks.org>	
David S. Miller <davem@redhat.com>	
Peter Bieringer <pb@bieringer.de>	Tests, tests and tests…
YOSHIFUJI Hideaki <yoshfuji@linux-ipv6.org>	USAGI
Pekka Savola <pekkas@netcore.fi>	
We are not alone ...	
Laurence J. Lane <ljlane@nontoxic.org>	Debian feedbacks
Andreas Herrmann <AHERRMAN@de.ibm.com>	SMP problem
Florian Lohoff <flo@rfc822.org>	sparc64 problems
Norbert Sendetzky <norbert@linuxnetworks.de>	Multiport
Art Haas <ahaas@airmail.net>	C99
Patrick McHardy <kaber@trash.net>	/proc/net matches/targets ownercmd, addrtype
Frank 'xraz' Fricke <xraz@rwxr-xr-x.de>	ROUTE
Andrew Shirrayev <andrews@gate.ort.spb.ru>	MARK(ext)
Pellegrini Giuliano <fanabe@tin.it>	length(test)
Lubomir Bulej <pallas@kadan.cz>	POSTROUTING 2 the mangle
KUNITAKE Koichi <kunitake@linux-ipv6.org>	LOG(fix)
Guillaume Morin <guillaume@morinfr.org>	LOG(ext-ECN)
James Morris <jmorris@intercode.com.au>	lipipq handler
Fernando Anton <fanton@it.uc3m.es>	QUEUE
Stephane Ouellette <ouellettes@videotron.ca>	Condition match
Anders Fugmann <afu@fugmann.dhs.org>	MARK target
Erik Hensema <erik@hensema.net>	Promote MARK values for SITx
Sneppe Filip <Filip.Sneppe@cronos.be>	USAGI & ip6tables
Iain Barnes <zenadsl3030@zen.co.uk>	DiffServ (DSCP/dscp)
John Wells <wells@ieee.org>	Mobile IPv6

Connection tracking

Harald Welte <laforge@gnumonks.org>
Patrick Schaaf <bof@bof.de>
Jozsef Kadlecsik <kadlec@blackhole.kfki.hu>

NAT, NAT-PT, SIIT (depends on IPv4/IPv6 independent CT system)
http://www.securityaudit.hu/Netfilter/NAT.txt

History

July, 2001 1st steps

2.1 Ports to IPv6 are in the work

Configuration options for IPv4 vs options for IPv6 – there are LOT more for IPv6.

The IPv6 code was an integrated contribution, but we don't have anybody who is really maintaining the IPv6 code. We (the four core maintainers) try to keep it up to date, but there isn't much testing. Currently we need somebody to really look after the IPv6 code. There is still a lot missing. This is really very EASY work – you don't really need to know much about kernel programming to port IPv4 netfilter code to IPv6. Right now netfilter is DEEPLY bound to the network level, so we right now we still need to use copy/paste pattern coding...

December, 2001 Roadmap

- common codebase
  - 'inline function' based code
  - changes: tools, libraries, kernel
  - requires 4:
    - connection tracking
    - eliminate the doubled codes
  - plan: it's a really hard work, so it's really a plan. Some plan will be available in the near future. It is requires changes in iptables (maybe only in Makefile and in the comparisons) and in the ip6tables (maybe all the function names!)
- 'iptables2' support
  - ipv6 part of the 'nf' command (from Jay)
- ipv6 extension headers
  - regular match (maybe a cleanup of Brad's code)
  - matches for the extension headers (some of them is easy, some of them is not)
- connection tracking
  - only on the 'common codebase'
  - depends on the 1st point
- the tunnel support is only the future.
- the full extension header functionality is not in our scope. We will support only one from all types in one packet.

- IPv6 main header: we are using some basic match. Future requirements:
  - traffic class (match/mangle)
  - flow label (match)
  - next header (match)
  - hop limit (match/mangle)
- Hop-by-Hop
  - next header, length, options ...
- Destination options
- Routing header
  - next header, length
  - routing type (match)
  - segments left (match)
  - reserved-field checking (must be 0) (match/mangle) [covert channel]
  - there's a rolling list of the internal addresses (end pointer: segments left)

- Fragment header
    - reserved fields (match/mangle)
    - next header (match)
    - identification
- AH/ESP (can be delivered from IPv4/IPSec)
- Extensions to header options: Pad1 and PadN (in any headers!)
- general header match (exists or not – bitmask based?)
- IPV6-ICMP It's a very complicated protocol...
    - error and general messages
    - management messages
        - router solicitation
        - router advertisement
        - neighbor solicitation
        - neighbor advertisement
        - redirect
        - options (in any message)
            - source link-layer address
            - target link-layer address
            - prefix information
            - redirected header
            - MTU
        - DOCUMENTATION: usage of these messages!!!

So, I think that we have enough work till the new API :)

A comment on the fragments:
The Linux kernel assumes that all the exthdrs fit to the first fragment at sending. (At receiving it must be something like this - I'm a little bit tired to check this now.)
In the Netfilter code: we assume that all the exthdrs AND the most significant part of the next protocol (eg: the TCP header) fit to the first fragment, and the remained fragments contain only protocol data.

January, 2002 NAT-PT

All the functions are bounded to L3, we cannot change the headers in the modules.
Full L3 independent L2 subsystem with HOOKs.
"Imran S. Patel" <imran@cs.ucsb.edu> has made a plain NAT-PT support, but that code still closed.

July, 2002 IPv6 connection tracking

Brad Chapman's connection tracking has been abandoned and lost.
There was a flame on the feature freeze. IPv6 connection tracking has been delayed. Code reuse had been explicitly denied.
Patrick Schaaf:
Right now, conntrack keys on four items: source IP, source port, destination IP, and destination port. All of them as per IPv4, 12 byte in total. Oh, and the protocol byte. 13 in total.
I have never personally worked with IPv6, but from what I heard, apart from the 128bit addresses, TCP carries over (almost?) unchanged, with two 16-bit port numbers, as before. I assume that the protocol byte also remains unchanged.

So, what we need, is exactly the same as the current conntracking, only the IP addresses are 128 bit instead of 32 bit.
Now, all the code which uses the conntracking structures (state match, NAT, expectations, etc,) must be reimplemented to be address-neutral.

August, 2002 IPv6 and NAT

Imran Patel:
But, it is good that you opened up this topic of IPv6 NAT. Implementing IPv6 conntrack has been discussed on and off on this list and NAT-PT is cited often as one of the reasons for doing it the "right" way.
I think there are two ways of doing NAT-PT correctly:
1. Let us assume that we just port IPv6 conntrack from IPv4 (which i guess has been attempted). Then, we can basically hack-up NAT-PT to work like this:
A packet from a v6 node to v4 world will have srcaddr = v6addr and destaddr=prefix::v4 where prefix is the 64 bit advertised prefix. At the NAT box, we take this pkt, and src NAT it from v6addr to pre::v4addr where v4addr is our public IPv4 address (pre can be a 0 prefix). The NATTed packet has srcadd = 0::v4addr and dstaddr = prefix::v6addr. Then the NAT-PT module just does straight IPv6-IPv4 stateless NAT and produces an IPv4 pkt with sraddr = v4addr and destaddr = v4. This packet then goes through Local NAT hook in the IPv4 stack and is then put on the wire. Reply packets follow the almost same logic.
v6-------v6-NAT: NAT-PT: v4-NAT------------v4

This is an incredibly ugly way of doing this "correct" using the existing pieces. The stateless NAT-PT module will need to have stateless NAT helpers also.

2. An elegant way of doing this would be be by using conntrack code which is L3 independent. Then we will need to introduce a special IPv6<->IPv4 conntrack+NAT module besides the usual v6-v6 and v4-v4 modules and all the l3 independent helpers.

Andras Kis-Szabo (on NAT-PT):
I think the userspace interface is the last 10-15% of the total work. :) (Like udp/icmp checksum calculation, udp-frag, ipv6 frag hdr, icmp-translation, internal nat at the icmp error messages, internal and external ip pools, prefix advertisements are the hardest part, i think :))
With a -working- NAT-PT/SITT support we will get a great weapon on IPv6 :)
Without connection tracking we must maintain in-kernel translation-tables :(
2nd point:
At this scenario we must reinject the packet. The IPv4 and the IPv6 conntrack/nat/filter modules have to examine it. With the simple reinjection the kernel will perform the protocol-transformation from IPv6 to IPv4. Who and when will perform the necessary translation on the IP-level headers?
There're two other tasks, too:

- IPv6 node - IPv6 node communication over that prefix.
  IPv6 - NAT-PT - IPv4 - IPv4 - NAT-PT - IPv6. We should avoid this :)
- routing loops.

April, 2003 iplimit 4 IPv6 – CT, again

Jozsef Kadlecsik
A blind porting of the IPv4 conntrack is unacceptable due to the code duplication.
A blind union with IPv4 conntrack is unacceptable due to the sheer wasting of memory.
An intelligent unification of IPv4/6 conntrack is possible. That itself is not so easy and one should keep in mind the relation with NAT, which makes it at the end complicated and hard.

May, 2003 tracking IPv6 connections by Patrick Schaaf

before the thought leaves me, again, here's an idea of how to cope with IPv6 addresses in the context of our current conntrack structure.

The basic idea is to have a separate hash, with an 128 bit IPv6 address as the key, and a value like this:

```
      struct unique_ipv6_address {
            struct list_head hash_link;
            atomic_t usage;
            unsigned char unused[4];
            unsigned char addr[16];
      /* 32 byte */ };
```

Now, given such a hash, we can use the 32-bit 'struct unique_ipv6_address *' in the place of the IPv4 address in our conntrack tuples. If we completely mix IPv4 and IPv6 in one big conntrack hash, we need one or two bit more per tuple, to indicate whether we have a v4 or v6 address. How nice: our dst.protonum field is 8 bits too long, so we have room for that!
The 'usage' of the unique_ipv6_address would count how many tuples reference that address, freeing the address hash entry when usage drops to 0.

Harald Welte
Well, what is basically clear is that we don't want a simple 'port' to IPv6. The new conntrack should be layer 3 independent. So abstracting it on level higher (like we now have layer 4 modules like ip_conntrack_proto_tcp.c, we should have something like net_conntrack_l3proto_{ipv4,ipv6}.c).
This way the connection tracking code can be used for any other kind of connection (think about LLC, ...) and also for connections with two ends in different layer 3 protocols. IPv4 <-> IPv6 transition NAT people would definitely like that.
The biggest problem we introduce is: Yet a bigger ip_conntrack, because the tuples will grow. instead of two 32bit addresses we now have two 128bit addresses. This means 48 bytes more per conntrack entry, independent if it is ipv6 (and really needs that space) or isn't (and has empty space. Solutions? Well, we could have differently-sized conntrack structs, depending on their l3 protocol(s). This would mean we'd have to shift l3 specific members to the end of the structure, and all generic members to the front. Then have one slab cache for every l3 protocol.

June, 2003 USAGI

Yausyuki KOZAKAI <yasuyuki.kozakai@toshiba.co.jp>
I'm a member with USAGI Project, and writing the codes for IPv6 Connection tracking.
I know the patches by Mr. Brad Chapman and the talks about that the Connection tracking system is going to be restructured to make it more generic and more easily ported to other layer-3 protocols. But I can't find any talks about that recently.
Could you tell me the status of that ? Is there the restructured system ? If so, is it going to be ported to Linux 2.5 ?