

IPV6 SECURITY



Version 1.1 - 14 march 2014

Written for: SURFnet

By: Iljitsch van Beijnum

SURF **NET**

IPV6 SECURITY

Version 1.1, 14 mrt 2014

Written for

SURFnet

by Iljitsch van Beijnum



For this publication Creative Commons Licentie "Attribution 3.0 Unported" applies
More information about this licence can be found on <http://creativecommons.org/licenses/by/3.0/>


CONTENTS

1. Introduction.....	3
2. IPv6 features	3
2.1. Address representation.....	3
2.2. Address configuration	4
2.3. No NAT	5
2.4. Identifying users by their IPv6 address.....	6
2.5. (No longer) mandatory IPsec	6
2.6. Link-local addresses	7
2.7. Neighbor Discovery and SEND.....	7
2.8. Hop Limit = 255 for local packets	8
3. Migration from and coexistence with IPv4	8
3.1. Dual stack.....	8
3.2. Tunnels.....	9
3.3. Translators (NAT64)	10
3.4. IPv4 addresses in IPv6 addresses.....	10
4. IPv6 threats	11
4.1. Scanning	11
4.2. Network mapping using multicast packets.....	12
4.3. Neighbor Discovery cache exhaustion	12
4.4. Rogue Router Advertisements and RA Guard	13
4.5. Rogue DHCPv6 servers	13
5. Threat mitigation.....	14
5.1. Address configuration security tradeoffs	14
5.2. Filtering, extension headers and fragmentation.....	15
5.3. Extension headers.....	16
5.4. Hardware versus software filters/firewalls.....	16
5.5. (Unique) Site Local addresses	17
5.6. Global unicast addresses	17
5.7. Stateful firewalling to replace NAT	17
5.8. Address prefix and ICMPv6 filtering	18
5.9. Prefix length filtering in BGP	20
6. Disabling IPv6.....	20
7. Acknowledgments.....	21
8. References.....	21

I. INTRODUCTION

Although for many of us, IPv6 is new, it's not **that** new. IPv4 celebrated its 30th birthday a few years ago, and IPv6 has been around for about half that time: the first IPv6 specification was published in 1995. This was before many things we take for granted now were added to IPv4, such as DHCP and NAT. So in some ways, IPv6 is (or seems) **older** than IPv4. Of course IPv6 also has some new features or details that IPv4 doesn't have. Other things are just different. From a security perspective, it's important to know what all these differences are, and how they relate to security issues. Often, the temptation will present itself to make IPv6 fit with the current way of doing things, but such "IPv4 thinking" may stand in the way of realizing certain IPv6 benefits, or create problems down the road. This document will present the most prominent IPv6 features from a network security perspective, and discuss how to manage those features, as well as issues that relate to the coexistence between IPv6 and IPv4. There are of course many other security aspects to connecting PCs/workstations/hosts to the (IPv6 or IPv4) internet that fall outside the scope of this document; please use best practices as appropriate.

The target audience for this document is system and network administrators of SURFnet-connected (or similar) organizations running campus networks.

 Because most widely used operating systems have IPv6 enabled by default, certain IPv6 security issues also arise in networks that have are not connected to the IPv6 internet. We therefore urge administrators of networks that don't run IPv6 to familiarize themselves with the issues discussed in this document, too.

2. IPV6 FEATURES

In addition to the longer addresses and the obvious changes that this requires, like the new IPv6 header, the new AAAA DNS record and new or extended routing protocols, the IETF took advantage of the fact that changes were happening to add additional new features.

The one thing where IPv6 is radically different from IPv4 is the size of its addresses. IPv4 has 32-bit addresses, allowing for a maximum of 4.3 billion addresses. (Although many can't be used in practice for a variety of reasons.) IPv6 quadruples the number of address bits. The 128-bit IPv6 address space allows for a maximum of 340 billion billion billion billion addresses.

The most immediate consequence of that large address space is that scanning IPv6 addresses is much, much harder than scanning IPv4 addresses, as discussed in section 4.1. However, the additional bits also allow for additional uses with varying security implications. The large address space also allows for a different way of obtaining addresses for hosts: stateless autoconfig with and without privacy extensions, as discussed in section 2.2.


2.1. Address representation

IPv4 addresses are written as four decimal numbers ranging 0 - 255 separated by periods: 192.0.2.31. Although some systems may accept leading zeros (192.000.002.031) or use old, class-based representations (16.1 rather than 16.0.0.1), these are rare, and there's only a single canonical format.

With IPv6, addresses are represented in text as eight four-digit hexadecimal values separated by colons: 2001:db8:31:1af:20a:95ff:fe5:246e. Like with IPv4, leading zeros may be present, but are typically left out; the letters in IPv6 addresses are usually in lower case, but upper case is also allowed. When writing down IPv6 addresses, one sequence of zeros separated by colons may be left out: 2001:db8:31:0:0:0:0:53 becomes 2001:db8:31::53.


But when there are two or more sequences of colon-separated zeros, only one may be omitted. So 2001:db8:0:0:1:0:0:1 may be represented as 2001:db8::1:0:0:1 or 2001:db8:0:0:1::1. (But **not** 2001:db8::1::1!) Finally, the last 32 bits of an IPv6 address may be written down as an IPv4 address. So 2001:db8:31::192.0.2.1 is a valid way to write down 2001:db8:31::c000:201. In many cases, when an address is entered by the user in the former format, it is then represented back by the system in the latter format. For instance:

```
$ ping6 2001:db8:31::192.0.2.1
PING6(56=40+8+8 bytes) 2001:db8:3100:a006:1:: --> 2001:db8:31::c000:201
```

-  The above creates two issues. First, it's very hard to do input validation that allows all legal ways to represent an IPv6 address, while rejecting text strings that are not IPv6 addresses. Second, when searching for text representations, like in databases, spreadsheets or text documents, the address may not be found while it is in fact present.

To avoid these issues where possible, [RFC 5952](#) recommends the following canonical text representation for IPv6 addresses:

- Leading zeros must be suppressed
- :: must be used to shorten sequences of zeros as much as possible
- But a single :0: must not be replaced with ::
- When replacing two or more sequences of zeros with :: would result in the same length, replace the first one
- Letters in IPv6 addresses must be in lower case
- The last 32 bits in an IPv6 address may only be in IPv4 form if the address falls within a well-known prefix, such as ::ffff:0:0/96

-  It's a good idea to use existing IPv6 address parsing/validation libraries where possible rather than come up with your own code, it's too easy to make mistakes.

Another issue with text representation of IPv6 addresses is when a port number is also included. The most common and recommended way to do this is [2001:db8:31::c000:201]:80, but this is application-dependent.

2.2. Address configuration

When IPv6 was created, DHCP was a new protocol that wasn't in common use yet. However, the ability to obtain an address automatically without manual configuration was an obvious useful addition, so IPv6 got stateless autoconfiguration. (Sometimes called stateless address autoconfiguration or SLAAC.) Stateless autoconfig works a lot like the old IPX protocol: a host combines a 64-bit prefix sent out by routers in periodic Router Advertisements with a 64-bit "interface identifier" to create a 128-bit IPv6 address.

The interface identifier is basically a 64-bit MAC address. A normal 48-bit MAC address is extended to 64 bits by adding the bits **ffe** in the middle, and the unique/local bit is flipped. So the MAC address 40:6c:8a:32:4b:c3 results in the interface identifier 426c:8aff:fe32:4bc3. The result of stateless autoconfiguration is that a host always generates the same address, even though there is no configuration or database to keep track of addresses.

However, when using stateless autoconfiguration, hosts that connect to different networks can be tracked by the MAC address embedded in their IPv6 addresses. For instance, if a user has address 2001:db8:31:1af:426c:8aff:fe32:4bc3 at home and 3ffe:a00:6:993:426c:8aff:fe32:4bc3 at work, services used by that user will be able to conclude that it's the same host contacting the server in both cases. To avoid this, [RFC 4941](#) specifies privacy extensions to stateless autoconfig: rather than using a stable MAC address, the lower bits of the IPv6 address are filled with a random number. A new random number is typically generated each time the host (re-) connects.

connects to the network or after 24 hours. Windows and Mac OS X as of version 10.8 Mountain Lion generate a privacy address as well as a stable address; the former is used for outgoing connections, the latter can be used for incoming connections. With Windows, the stable address is not (directly) based on the system's MAC address. Linux and the BSD family operating systems support privacy addresses, but whether those are used depends on the defaults set in the distribution and the configuration that is used.

In the mid-2000s, DHCPv6 was added, which works largely the same as IPv4 DHCP. However, in addition to an IP address and information such as DNS server addresses, IPv4 DHCP also provides a netmask and the address of a default router / default gateway. With IPv6, a netmask is replaced by a prefix length. However, DHCPv6 doesn't provide a prefix length or default gateway address. IPv6 subnets are supposed to have a /64 prefix length and things usually still work even if no prefix length is configured, but obviously the default gateway address is required for external connectivity. So routers must send out Router Advertisements to supply this information. Also, the Router Advertisements tell hosts whether to use stateful DHCPv6 (to configure addresses), stateless DHCPv6 (just for additional information such as DNS server addresses) or no DHCPv6 at all.

Another difference between DHCP for IPv4 and DHCPv6 is that DHCPv6 uses a DHCPv6 Unique Identifier (DUID) to identify DHCPv6 clients, rather than the client's MAC address or a client ID string. This makes adding client-specific settings to a DHCPv6 server configuration more difficult. [RFC 6936](#) aims to solve this problem.

It is possible to use both stateless autoconfig and DHCPv6 for address configuration; hosts that support both will then configure three addresses: a stable autoconf-based address, a privacy address and a DHCPv6-based address. All IPv6-capable systems support stateless autoconfig. Windows supports DHCPv6 since Windows Vista, Mac OS X supports DHCPv6 since 10.7 Lion. Most open source operating systems / distributions support DHCPv6, but sometimes it needs to be enabled or installed explicitly.

A Router Advertisement can contain multiple prefixes for the purpose of stateless autoconfig. Different routers may send out Router Advertisements with the same or different subnet prefixes and/or different DHCPv6 configurations. Generally, hosts will use a superset. So if one router sends out two prefixes and no DHCPv6, and another one a third prefix and stateful DHCPv6, hosts will create six addresses using stateful autoconfig (three stable and three privacy addresses) and contact a DHCPv6 server for a seventh address.

2.3. No NAT

The reason Network Address Translation (NAT) exists is to allow for multiple IPv4 hosts to share a single public IPv4 address. To accomplish this, these hosts are given private ([RFC 1918](#)) addresses and a NAT device translates between a public address given to the NAT device and these internal addresses. This breaks or at least complicates protocols that carry addresses as part of the protocol, such as FTP and SIP. Otherwise, it works for sessions initiated by the hosts behind the NAT: incoming packets are sent to the host that initiated earlier matching outgoing packets. But for new connections initiated from the outside, the NAT device has no way of knowing where to send these packets.


So as a side effect, NATs filter incoming TCP sessions and unsolicited UDP packets. However, there are different types of NAT, and some types do allow incoming sessions/packets in certain situations. For instance, when an outgoing session to destination A uses port X, then many NATs allow incoming sessions/packets from remote hosts other than A towards port X.

Because IPv6 has so many addresses, there is no reason to have multiple hosts share a single address. As such, NAT is not routinely used with IPv6, and products that support IPv4 NAT typically don't support IPv6 NAT. Because NAT is not expected with IPv6, workarounds that

make IPv4 work through NAT may not exist for IPv6 so if NAT is applied with IPv6, it is likely to create more issues than with IPv4.

[RFC 4864](#) provides an overview of security mechanisms that can be deployed with IPv6 to duplicate the security benefits of NAT without address translation.

There is an experimental specification for IPv6 prefix translation ([RFC 6296](#)), but here an entire prefix is translated, providing every internal host with its own external IPv6 address. So this type of NAT doesn't provide the same filtering of incoming sessions/packets that typical IPv4 NATs provide.

 When providing IPv6 connectivity to hosts that are behind IPv4 NAT, it may be necessary to install an IPv6 stateful firewall. See section 5.7. Also realize that either native or tunneled IPv6 connectivity will use publicly visible IPv6 addresses and may expose information about the network that was previously hidden by the presence of an IPv4 NAT.

2.4. Identifying users by their IPv6 address

Many network administrators feel uncomfortable with hosts generating their own addresses using stateless autoconfig, and thus prefer using DHCPv6. It is then possible to log (or pre-configure) the IPv6 address - MAC address relationships.

However, using DHCPv6 for IPv6 address configuration only makes it possible to identify hosts under routine circumstances; a malicious user may still generate a non-DHCP address or steal a DHCP address from another host. Logging of the MAC address - IPv6 address relationship addresses this issue, except that MAC addresses, too, may be faked. To avoid this, use 802.1x on wired networks and WPA2 enterprise for Wi-Fi and log which user got which IPv6 address.

Note that even when stateless autoconfig is used exclusively for address configuration, DHCPv6 is often used to configure the IPv6 addresses of DNS resolvers. There is also an RA option for doing this, but not all routers or operating systems (notably Windows) implement it. (And in dual stack networks IPv4 may be used to perform DNS queries.)

There are several situations where operators of services may want to identify users by their IP address. For instance, a service may want to block a user, or limit how many times or how often a user may perform a certain action. This is often done based on a user's IP address. In the IPv4 world, this can be problematic if multiple users share the same public IPv4 address, but with IPv6 the opposite can easily happen: a user may get a different IPv6 address relatively frequently even when connected to the same network. A possibility would be to filter on the first 64 bits of the IPv6 address, which would provide a similar "resolution" as filtering on individual IPv4 addresses. However, many ISPs provide more than a single /64 to their subscribers, so motivated users would be able to circumvent filtering. Ottow et al. (2012) suggest escalating to filtering larger and larger blocks as undesired actions continue.

Ultimately, there is no single approach that fully solves the issue of filtering users that don't want to be filtered; and using IP addresses for this purpose is harder with IPv6 than it is with IPv4.

2.5. (No longer) mandatory IPsec

[RFC 4294](#), the IPv6 Node Requirements, states that IPv6 nodes (hosts and routers) **must** support IPsec. However, [RFC 6434](#) relaxes this, saying they **should** implement IPsec. IPsec is a mechanism to authenticate and/or encrypt communications on a per-packet basis, which makes it more general-purpose and more effective than TLS/SSL.

Most systems that can run IPv6 do in fact support IPsec. However, IPsec only helps if you actually use it. Which is very hard to do. So in practice, IPsec is only used on a significant scale

for VPN tunnels. IPsec is also no longer a differentiator between IPv4 and IPv6, because even though IPsec was originally developed for IPv6, it works equally well with IPv4.

2.6. Link-local addresses


Probably the most important new feature other than the longer addresses is the fact that IPv6 has link-local addresses. All interfaces on which IPv6 is enabled and that are up **always have a link-local address**. These addresses are in the `fe80::/64` prefix and the bottom 64 bits are the stable interface identifier that is also used for non-privacy stateless autoconfig addresses. The `fe80::/64` prefix is reused on each interface. This is possible because these addresses are only valid within a "link" or subnet; they can't pass through routers. However, it does mean that when using these addresses, it's necessary to specify the output interface. Most systems attach a percent sign followed by the interface name to link-local addresses when necessary. For instance, this ping command uses the `en0` interface:

```
$ ping6 fe80::8a1f:a1ff:fe29:923c%en0
PING6(56=40+8+8 bytes) fe80::bae8:86ff:fe3a:69f2%en0 -->
fe80::8a1f:a1ff:fe29:923c%en0
```

This one uses the `en4` interface:

```
$ ping6 -c 3 fe80::8a1f:a1ff:fe29:923c%en4
PING6(56=40+8+8 bytes) fe80::426c:8fff:fe3b:4bc3%en4 -->
fe80::8a1f:a1ff:fe29:923c%en4
```

Link-local addresses allow IPv6 communication when systems don't have a "real" IPv6 address (known as a global unicast address), or when they have addresses in different subnet prefixes. For this reason, link-local addresses are used for IPv6 house keeping purposes, such as sending router advertisements, ICMPv6 redirect messages and so on.


-  A side effect of link-local addresses is that when systems make services available over IPv6, those services will be reachable for anyone on the local subnet. So it's very important to always have the appropriate IPv6 access restrictions, even if systems will not have any IPv6 connectivity.

2.7. Neighbor Discovery and SEND

IPv4 uses ARP over Ethernet and other layer 2 networks that behave like Ethernet, such as Wi-Fi. IPv6 uses Neighbor Discovery to determine which MAC address goes with which IP address. Neighbor Discovery uses several ICMPv6 messages:

- Neighbor Solicitation, to inquire which MAC address goes with which IPv6 address
- Neighbor Advertisement, to answer a Neighbor Solicitation
- Router Solicitation, to ask for a Router Advertisement
- Router Advertisement, to indicate the presence of routers and for address configuration

IPv6 doesn't support broadcasts, so when they're not addressed at a single, known system, these messages are sent to multicast addresses instead; see [RFC 4861](#) for details.

-  Attackers can of course spoof Neighbor Discovery messages to disrupt IPv6 traffic, or to reroute packets for the purpose of inspection and modification. (Similar to ARP spoofing with IPv4.) Or disrupting the Duplicate Address Detection phase of address configuration to trick a system into thinking its address is already in use, potentially leaving the system without a working IPv6 address. A particularly troublesome variation of this attack is using it against the link-local address of a router. If successful, the router is left without a link-local address and unable to route IPv6 packets over the affected interface.

SEcure Neighbor Discovery (SEND, [RFC 3971](#)) addresses allows systems to reject false Neighbor Discovery packets sent by untrusted systems on the local subnet. With SEND, hosts and routers have a certificate that can be trusted through a trust path. Addresses are then tied to a certificate through Cryptographically Generated Addresses (CGAs). A CGA is an IPv6 address where the interface identifier portion consists of a hash of a certificate's public key. Signatures are carried in option fields in the Neighbor Discovery packets.

SEND allows hosts and routers implementing it to perform Neighbor Discovery functions correctly even in the presence of malicious systems on the local subnet. In particular, unauthorized Router Advertisements are rejected.

However, SEND is not widely implemented.

2.8. Hop Limit = 255 for local packets

Neighbor Discovery and ICMPv6 redirect messages are only meaningful between hosts and routers connected to the same subnet. So when those packets arrive over the internet, someone is trying to do something malicious. To avoid attacks based on these packets, IPv6 systems check whether these messages were generated by a system connected to the local subnet, or were sent from elsewhere (across the internet).

When a system sends an ND or redirect message, it sets the Hop Limit field is set to 255, the maximum value that fits in the field. The receiver then checks the Hop Limit field. If it's 255, the packet is accepted as being sent by a local system and processed.

If the packet is non-local and thus passed through one or more routers, the routers would have decremented the field. So non-local packets can never have a Hop Limit of 255, and packets with a Hop Limit below 255 are ignored. This check allows systems to throw out spoofed packets without the need for filters or other complex security mechanisms. This security mechanism is part of the base IPv6 specifications and can't be disabled.

3. MIGRATION FROM AND COEXISTENCE WITH IPV4

Running a pure IPv6 network is more straightforward than running an IPv4 network. Subnets are (almost) always a /64, and there is enough address space to set aside subnets for special purposes, and there's no NAT. However, running a dual stack IPv4 + IPv6 network entails all the complexity of IPv4, all the complexity of IPv6, and then additional complexity for the two to exist side-by-side.

3.1. Dual stack

Dual stack means running IPv4 and IPv6 side-by-side. As it's rarely realistically possible to turn off IPv4 yet, dual stack is currently the most common way to implement IPv6. Although for applications, the same API calls are typically used for both IPv4 and IPv6, "on the wire" the two protocols are completely separate. If a DHCP server gives out an IPv4 address and an IPv4 default gateway, the host runs IPv4. If Router Advertisements provide an IPv6 default gateway and RAs or DHCPv6 provide IPv6 addresses, a host runs IPv6. When both happen, the host runs dual stack.

Note that the same router may serve as both be the IPv4 router and the IPv6 router on a subnet, or there may be one router handling IPv4 and another router handling IPv6. When multiple IPv6 routers are present, IPv6 hosts can automatically switch between them.

So for the most part, dual stack comes down to running normal IPv4 and running normal IPv6 independently. When making outgoing connections, hosts that have IPv4 connectivity look up A records in the DNS and communicate over IPv4. Hosts that have IPv6 connectivity look up AAAA records and connect over IPv6. Dual stack hosts, however, look up both A and AAAA records, and if both are available, they need to choose whether to connect over IPv4 or IPv6.

Until not too long ago, dual stack hosts would prefer to use IPv6 when possible. However, this means that if IPv6 connectivity doesn't work or is slow, the user experience suffers. To avoid this, Mac OS X and Windows currently implement "happy eyeballs" techniques that try to determine the quality of the IPv4 and IPv6 connectivity, and use whichever is better. As a result, it's hard to predict whether an outgoing connection is going to use IPv4 or IPv6. (However, Windows and most Unix(-like) systems provide a way to influence these decisions through the [RFC 6724](#) policy table.)

3.2. Tunnels

IPv6 tunnels are a mechanism for carrying IPv6 packets across parts of the network that only support IPv4. There is a surprisingly large number of tunneling mechanisms; consult [RFC 7059](#) for an overview.

A regular PC or other system connected to the network can be configured to route IPv6 packets between an IPv6 tunnel and its Ethernet or Wi-Fi interface, sending out Router Advertisements to attract IPv6 packets from other hosts. If those RAs aren't filtered, hosts connected to the same subnet will start sending their IPv6 packets to the "rogue" router.

This is best addressed using RA Guard to filter out rogue RAs rather than trying to filter the resulting tunnel packets, because it's relatively easy to identify rogue IPv6 routers, while tunnels may be encrypted or otherwise obfuscated.

When using tunnels for external IPv6 connectivity, it's important to treat them the same as other external connections, and perform the appropriate filtering. This is more complex when using automatic tunneling mechanisms such as 6to4, where there is a relationship between certain bits in the IPv6 address in the IPv6 header, and the IPv4 address used in the IPv4 header that encapsulates the IPv6 packet.

But malicious users may use such tunnel mechanisms to bypass source address filters. For instance, someone on the internal network with address ranges 192.0.2.0/24 and 2001:db8:31::/48 wouldn't normally be able to generate packets with source address 145.0.2.10 or 2001:610:188:301:145::2:10. However, someone may attempt to send a 6to4-encapsulated packet with IPv4 source address 192.0.2.15 that holds an IPv6 packet with source address 2001:610:188:301:145::2:10. By sending small DNS requests that generate large responses in this way, a malicious user would be able to launch a DNS amplification attack that would be very hard to trace back to the source.

6to4 tunneling is enabled by default on Windows systems when they have a public IPv4 address but no global unicast IPv6 address. Teredo tunneling, which, unlike 6to4, works through IPv4 NAT, is enabled with a Windows system has a private IPv4 address and no global unicast IPv6 address. This means that 6to4 or Teredo may be used to bypass firewall restrictions—both from the inside to reach filtered outside destinations, and by systems on the outside trying to reach hosts inside the network.

For this reason, it is not uncommon for organizations that use public IPv4 address space for PCs/workstations to filter protocol number 41, which is used by 6to4 and several other tunnel mechanisms. In the past, this would be problematic, because the Windows systems would think they had IPv6 connectivity, but that connectivity didn't actually work. As a result, destinations with IPv6 addresses in the DNS would be hard to reach. However, because of this issue, 6to4 connectivity is no longer preferred over IPv4 connectivity, so filtering protocol 41 isn't as

problematic as it was before. Teredo connectivity is only used as a last resort, so filtering UDP port 3544 typically won't create any problems.

However, simply providing native (untunneled) IPv6 connectivity will achieve the same results. Another good solution would be a firewall and/or IDS that can look inside tunnel packets and filter the inner IPv6 packets.

See [RFC 3964](#) for 6to4 security considerations.

3.3. Translators (NAT64)

NAT64 translates between the IPv6 and IPv4 packet formats. With stateless NAT64 ([RFC 6145](#)), a single IPv4 address maps to a single IPv6 address. As such, stateless NAT64 doesn't address the current shortage of IPv4 addresses. There's also stateful NAT64 ([RFC 6146](#)), where multiple IPv6-only clients share a single IPv4 address in order to connect to IPv4-only servers. Stateful NAT64 works along with DNS64. A regular DNS server returns an empty result when an IPv6 host looks up the IPv6 address of a server that only has an IPv4 address. A DNS64 however, generates a synthetic IPv6 address from an IPv6 prefix that is routed towards the NAT64 translator and the destination's IPv4 address. When the IPv6 host then tries to connect to that address, its packets end up at the NAT64, which translates from IPv6 to IPv4 and applies IPv4 NAT. Return packets from the server are translated back to IPv6.

The main security implication of NAT64 is that if the NAT64 translator accepts packets coming from the outside for translation, malicious external users may use it as an open proxy. So it's important to limit access to the NAT64 to legitimate users, typically by not allowing packets from the outside towards the NAT64 prefix.

3.4. IPv4 addresses in IPv6 addresses

As noted in section 3.2 (and also see the [SURFnet IPv6 number plan document](#)), it is possible to make an IPv4 address part of an IPv6 address. But the `x::x::x::x::y.y.y.y` notation is merely cosmetic; the address remains a normal IPv6 address. However, IPv4 addresses may be embedded in IPv6 addresses meaningfully in several ways. The most visible is when using tunnels, which will be covered later.

IPv4 addresses are also embedded in an IPv6 address when using NAT64. A "well-known" prefix has been set aside for stateful NAT64: `64:ff9b::/96`. But network operators can also use a prefix of their own for this purpose. The IPv4 address usually goes in the bottom 32 bits. For instance, if the NAT64 uses the well-known prefix and the IPv6 host wants to connect to a server with IPv4 address `192.0.2.31`, the DNS64 will return the synthetic address `64:ff9b::192.0.2.31`, or `64:ff9b::c000:201`. Organizations that run a stateful NAT64 gateway should make sure that filters are in place to make sure that only legitimate users can send packets through the NAT64 gateway.


Last but not least, the prefix `::ffff:0:0:/96` has been set aside to represent the IPv4 address space so that applications can use just the IPv6 Socket API to interface with the IP networking stack, rather than having to use the original Socket API for IPv4 and the IPv6 Socket API for IPv6. This means that when an application creates a connection to, for instance, `::ffff:192.0.2.1`, the system will in fact send IPv4 packets towards `192.0.2.1`. As such, there can never be valid IPv6 packets with such addresses in them.

Commonly used tunnel mechanisms such as 6to4 and Teredo generate IPv6 addresses based on a fixed prefix (`2002::/16` for 6to4, `2001::/32` for Teredo) and the user's IPv4 address. They then automatically encapsulate packets sent to addresses within those prefixes in an IPv4 packet and copy the IPv4 destination bits from the IPv4 bits in the 6to4 or Teredo IPv6 address. Teredo also includes the UDP port used to reach the host that runs Teredo in the IPv6 address.

4. IPV6 THREATS

Before we look at specific new features, it's important to note that IPv6 implementations may have bugs simply because they're newer. As a whole, the IPv6 specifications have been around for some time now, so most of the bugs in the specifications **should** have been found and addressed by now. Implementations are also becoming more mature, but they certainly haven't seen as many hours in production as their IPv4 counterparts. It's entirely possible that bugs that were fixed in an IPv4 implementation years ago are still present in the corresponding IPv6 implementation.

One example of such a bug is the Routing Header type 0 (RH0), which was used for source routing in IPv6. This header could be used to bypass (some) firewalls as well as be used in amplification attacks, and was "deprecated" by the IETF in [RFC 5095](#) in 2007.

-  Make sure there are procedures for updating IPv6 routers and hosts (including embedded systems) to new software or firmware when security vulnerabilities are fixed.

4.1. Scanning

In 2003, the SQL Slammer worm propagated by sending out copies of the worm towards random addresses. Because the worm code was only a few hundred bytes long, and the vulnerable service was reachable over UDP, the worm could be propagated in a single packet. These packets were sent to random IPv4 addresses at a high rate: some systems sent out 15000 copies of the worm per second. As a result, 80% of all vulnerable systems connected to the internet were infected within ten minutes.

Sending packets to random addresses isn't the best way to scan the entire address space, but the distributed nature of the Slammer worm makes up for that. If we assume ten minutes to hit each IPv4 address, then scanning the entire IPv6 address space would take 1.5×10^{24} years. So if someone had started scanning at the time of the big bang and scanned IPv6 addresses for the entire lifetime of the universe, they'd be at around address 0:0:2:9422:415f:8800:0:0 by now.

In other words, scanning **every** IPv6 address is completely infeasible. However, often, many bits are known beforehand, and then scanning is still possible. For instance, scanning a single /64 subnet for systems using stateless autoconfiguration without privacy addresses still takes two years to scan at a million packets per second (1 Gbps with 125-byte packets). But MAC addresses are given out in blocks of 16.8 million to vendors. Scanning one of these blocks takes only 17 seconds at a rate of 1 MPPS, so scanning a subnet for (for instance) Apple devices that are using stateless autoconfig takes time, but is certainly doable.

One security researcher (Heuse, 2012) found that 79% of networks respond to scans on the ::0, ::1 or ::2 addresses in at least one subnet. This allows someone with malicious intentions to quickly determine which subnets are in use and focus on those subnets. It may be useful to avoid giving such obvious addresses to routers and other systems and/or block packets coming in from the outside to those addresses. However, be careful filtering outgoing packets; see section 5.8.

When explicitly assigning individual addresses to individual hosts (through manual configuration or DHCPv6), it is common to either number hosts sequentially, embed an IPv4 address in the bottom bits of the IPv6 address in some form, or use words in the IPv6 address. (For instance, the IPv6 address for www.facebook.com is 2a03:2880:f006:101:face:b00c::1.) In all of these cases, the addresses may be guessed. Addresses may also be discovered through the DNS. In general, attackers will attempt to guess your numbering plan.

Automatic tunneling mechanisms such as 6to4 and Teredo (discussed in section 3.2) generate IPv6 addresses from a system's IPv4 address. These addresses are relatively easy to guess/scan.

4.2. Network mapping using multicast packets

IPv6 doesn't use broadcasts, but the all-hosts multicast address `ff02::1` serves the same purpose as the IPv4 `255.255.255.255` broadcast address. Many non-Windows systems respond to pings to that address:

```
$ ping6 -c 2 -I en0 ff02::1
PING6(56=40+8+8 bytes) fe80::bae8:86ff:fe3a:69f2%en0 --> ff02::1
16 bytes from fe80::bae8:86ff:fe3a:69f2%en0, icmp_seq=0 hlim=64 time=0.130 ms
16 bytes from fe80::8a1f:a1ff:fec9:323c%en0, icmp_seq=0 hlim=64 time=1.334 ms
16 bytes from fe80::60c:ceff:fe93:1860%en0, icmp_seq=0 hlim=64 time=112.837 ms
16 bytes from fe80::10e9:bd67:blad:a78c%en0, icmp_seq=0 hlim=64 time=119.508 ms
16 bytes from fe80::bae8:86ff:fe3a:69f2%en0, icmp_seq=1 hlim=64 time=0.175 ms
```

Even more helpful to people attempting to debug the network (authorized or otherwise) are node information queries ([RFC 4620](#)). These are ICMPv6 packets that request that IPv6 nodes return name or address information. Certain implementations of the `ping6` command can send these queries and display the results:

```
$ ping6 -c 2 -I en0 -w ff02::1
PING6(72=40+8+24 bytes) fe80::bae8:86ff:fe3a:69f2%en0 --> ff02::1
36 bytes from fe80::bae8:86ff:fe3a:69f2%en0: macbookpro.local
28 bytes from fe80::8a1f:a1ff:fec9:323c%en0: basestation
40 bytes from fe80::10e9:bd67:blad:a78c%en0: Apple-TV
36 bytes from fe80::bae8:86ff:fe3a:69f2%en0: macbookpro.local
```

Apple products still respond to local node information queries, but most other systems no longer do. Some older BSD systems would even respond to non-local NIQs with a lot of information:

```
$ ping6 -c 1 -a acgslA 2001:db8:31:1af::1
PING6(72=40+8+24 bytes) 2001:470:1f0b:1289:a5ad:a91f:9f44:d0e9 -->
2001:db8:31:1af::1
116 bytes from 2001:db8:31:1af::1:
 fe80::212:3fff:feed:d76c (TTL=infty)
 2001:db8:31:1af::1 (TTL=infty)
 ::1 (TTL=infty)
 fe80::1 (TTL=infty)
 2002:c000:201::1 (TTL=infty)
```

4.3. Neighbor Discovery cache exhaustion

Suppose someone scans IPv6 subnet `2001:db8:dead:beef::/64`, starting at the `2001:db8:dead:beef:0:0:0:0` (`2001:db8:dead:beef::`) address. The router then has to send a Neighbor Solicitation message (similar to an IPv4 ARP request, as discussed later) for address `2001:db8:dead:beef::`. Then for `2001:db8:dead:beef::1`, `2001:db8:dead:beef::2` and so on. Routers have a limited Neighbor Discovery cache, which will eventually be filled with incomplete entries for the addresses being scanned. Depending on the implementation, this may cause problems for hosts and routers holding addresses that are actually used in that subnet.

The same problem may occur with IPv4, but there it's rare. In many cases, a NAT will be in place, so it's impossible for someone to scan addresses within a subnet from outside the organization. But even without NAT, IPv4 subnets are typically small enough that the ARP cache isn't exhausted. However, some routers share resources between the IPv6 ND cache and the IPv4 ARP cache, so an attack against one may impact the other.

Filters for common types of packets used to do scanning, such as ICMPv6 echo request (ping), will stop the cache exhaustion as a side effect from "normal" scanning. However, it's hard to completely mitigate the issue, as any type of packet that remains unfiltered may be used to launch a deliberate Neighbor Discovery cache exhaustion attack. Only a full stateful firewall that

blocks all unsolicited packets coming in from the outside completely solves this. However, such restrictive stateful firewalls may get in the way of peer-to-peer communication.

Ideally, routers should implement Neighbor Discovery in such a way that cache exhaustion doesn't lead to problems. You may want to ask your router vendors for how they address this issue. On subnets with only routers and/or servers, it's possible to use a smaller block of IPv6 addresses than a /64 to mitigate this issue. (See section 4.12 of the [SURFnet IPv6 addressing plan manual](#).) It may also be useful to reserve a /64 but only use a /120, that way, it's possible to switch to a regular /64 subnet size later without renumbering.

4.4. Rogue Router Advertisements and RA Guard

With IPv4, there is the possibility that someone runs an unauthorized "rogue" DHCP server and disrupts the network. Similarly, someone may send out IPv6 Router Advertisements by accident or maliciously.

In networks that don't run IPv6, rogue Router Advertisements may be used to trick hosts into sending the traffic for IPv6-enabled destinations through an attacker, who can then inspect and/or modify the packets. Unlike rogue IPv4 DHCP servers, this has no impact on the IPv4 network, so it may continue unnoticed for some time. However, in most cases this is caused by Windows systems set up for connection sharing with no malicious intent. In those cases, IPv6 destinations will be unreachable or much slower for hosts that have both IPv4 and IPv6 enabled.

[RFC 6105](#) specifies Router Advertisement Guard (RA Guard), a mechanism to filter out rogue RAs in the layer 2 infrastructure. RA Guard can operate in different ways, but the most obvious one is to allow RAs from the switch ports where IPv6 routers are connected, and filter out RAs that arrive on all other ports. RA Guard can also be used on wireless networks, as all packets, even between two wirelessly connected hosts, must pass through a Wi-Fi base station.

Where possible, RA Guard should be used alongside similar filtering of IPv4 DHCP and DHCPv6. Because this selectively filters some IPv6 (and IPv4) packets and not others, this functionality is usually limited to higher-end switches and Wi-Fi base stations.

See the note about fragmentation and filtering in section 5.2.


4.5. Rogue DHCPv6 servers

Just like with IPv4 DHCP, it is possible for malicious users to run a rogue DHCPv6 server. However, this is uncommon. The impact of a rogue DHCPv6 server varies depending on the status of the M (managed config) or O (other stateful config) flags in Router Advertisements as per table 1.

Table 1, M and O flags in Router Advertisements and DHCPv6 use

Type of network	M and O flags	Rogue DHCPv6 impact
DHCPv6 is used for address configuration	M = 1	Attacker can disrupt address configuration and insert addresses of malicious DNS servers
DHCPv6 is used for other configuration (DNS etc.)	M = 0, O = 1	Attacker can insert addresses of malicious DNS servers
DHCPv6 is not used	M = 0, O = 0	No impact, hosts don't send DHCPv6 requests

So in networks where DHCPv6 is used, the main risk is that an attacker gets hosts to use a malicious DNS server. When DHCPv6 is used for address configuration, a denial-of-service type attack is possible through disrupting address configuration, but because DHCPv6 doesn't carry default gateway/router information, a rogue DHCPv6 server can't reroute traffic.

-  However, there are many additional DHCPv6 options for many different purposes, depending on the ongoing implementation status of these, a rogue DHCPv6 server may be used to exploit these. As such, in networks that use DHCPv6, it's important to filter out UDP packets with destination port 546 unless they come from a legitimate DHCPv6 server or DHCPv6 relay.

In networks where DHCPv6 isn't used, an attacker would have to send rogue RAs with $M = 1$ or $O = 1$ in addition to running a rogue DHCPv6. So in these cases deploying RA Guard **should** be sufficient to protect against rogue DHCPv6 servers. However, there may be hosts that send DHCPv6 requests even if the M and O bits are zero, so where possible, filtering (rogue) DHCPv6 servers is recommended.

5. THREAT MITIGATION

The main way to mitigate IPv6-related threats is to filter out the packets that may be used to execute those threats.

5.1. Address configuration security tradeoffs

In general, there is a tradeoff between ease of autoconfiguration and security. The only way to avoid all possible security issues with access control and address configuration is to enforce authentication before address configuration occurs (through IEEE 802.1x or WPA2 Enterprise) and then statically configure IPv6 addresses as well as Neighbor Discovery MAC address entries and enforce the use of authorized MAC addresses in Wi-Fi base stations or switches. This way, only authorized users can connect to the network, and then they can't spoof their MAC or IPv6 address to assume another user's identity. Of course this requires a certain amount of setup to add a new user and/or new device to the network.

A more practical approach is to separate users and/or workstations and other devices based on security requirements and assign different groups of users or devices to different subnets. After all, servers with sensitive data on them have different security requirements than the mobile phones and tablets of visitors, and by separating them, it's possible to give each the proper tradeoff between ease of (auto)configuration and security: in a server subnet, manual configuration may be called for to avoid accidents, but logging of addresses is then not required. For visitors, access control may be unfeasible and/or undesirable, but logging of addresses could be necessary to identify misbehaving users.

SEND would be especially useful in networks that occupy the middle ground: the users and devices are too numerous and diverse for manual configuration, but the users are known and the type of communication is potentially sensitive. In these cases, the need to install and/or configure security certificates wouldn't be unreasonable. However, SEND is not available in the most widely used operating systems. An alternative could be the use of a VPN.


Table 2, address configuration methods with and without SEND and static ND/MAC tables

	On its own	With SEND	With static ND/MAC tables
Manual configuration	No autoconfig, IP/MAC hijacking and DoS possible	No autoconfig, no IP hijacking, MAC DoS possible	No autoconfig, no IP/MAC hijacking or DoS
Stateless autoconfig	Hard to log IP addresses, hijacking and DoS possible	Logging easier, no IP hijacking, MAC DoS possible	Static ND not possible, static MAC makes logging easier, DoS harder
DHCPv6	Easy to log IP addresses, hijacking and DoS possible	Easy to log IP addresses, no IP hijacking, MAC DoS possible	Per-host DHCP config required, IP/MAC hijacking or DoS impossible

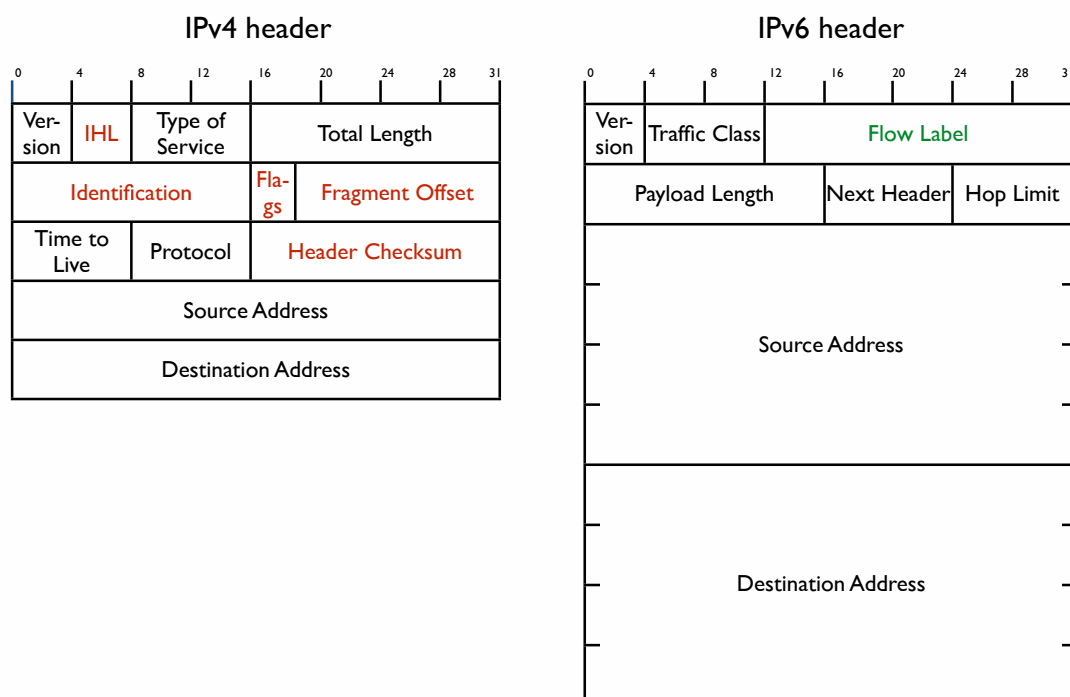
In this context denial-of-service (DoS) means that host A can't take over host B's IPv6 address or MAC address, but host B **can** disrupt host A's communication by derailing Neighbor Discovery or DHCPv6 with spoofed packets. For instance, SEND makes sure that only host A can use host A's IPv6 address, but host B can still claim to be the owner of host A's MAC address.

5.2. Filtering, extension headers and fragmentation

When firewalling/filtering IP packets, there is no one size that fits all. As such, we don't make specific recommendations, but rather, provide information that will be useful when creating filters.

 In a dual stack network, everything that is filtered in IPv4 must also be filtered in IPv6.

The IPv6 header is simplified compared to the IPv4 header. Apart from the (much) longer addresses, the most important differences are that the fields relevant to fragmentation are no longer present, and that the header is always 40 bytes long. If additional functionality is required, additional extension headers are placed between the IPv6 header and the packet payload.


Figure 1, the IPv4 en IPv6 headers

For instance, when an IPv6 packet is too large to be transmitted over an interface, it must be fragmented, the same as with IPv4. Because routers aren't allowed to fragment IPv6 packets, when hosts send IPv6 packets larger than 1280 bytes (the minimum IPv6 MTU), Path MTU Discovery must be used, so hosts can decrease the size of further packets. In the case of TCP, the packet size can be reduced without issue, but for UDP protocols, packet boundaries are fixed. So UDP packets must be fragmented. For this, the three fragmentation-related header fields that are now missing from the IPv6 header are needed: Identification, Flags and Fragment Offset. So a Fragment Header is added between the IPv6 header and the UDP header.

[RFC 6980](#) discusses issues that arise when Neighbor Discovery packets are fragmented; certain implementations silently drop such ND packets, but they can also be used to circumvent the RA Guard implementation in switches because the information required to perform RA Guard is split across multiple fragments. For these reasons, [RFC 6980](#) specifies that ND packets must not be fragmented.

5.3. Extension headers

Normally, the Next Header field in the IPv6 header functions the same way as the Protocol field in the IPv4 header, and usually contains 6 (TCP), 17 (UDP) or 58 (ICMPv6). However, when an extension header such as the Fragment Header is present, the Next Header field contains the number that indicates the type of the next header, in this case 44, for the IPv6 Fragment Header. The extension headers have a Next Header field of their own, which in this case is set to 17 (UDP).

 As a result, firewalls and IP filters must follow the chain of extension headers to determine if a packet is TCP, UDP or ICMP. In the past, there have been filters that solely look at the Next Header field in the IPv6 header, and thus failed to filter fragmented packets properly. When selecting a firewall or implementing a filter, check for this issue. A complicating factor is that although **most** extension headers follow a fixed layout, this is not mandatory, so an older firewall can't process extension headers that were defined after the firewall software was completed.


The Fragment Header is the most common IPv6 extension header, but others are also possible, including Shim6, the IPsec Authentication Header, the Routing Header, the Hop-by-Hop Options Header and the Destination Options Header.

The Routing Header is used for source routing; type 0 (RH0) has been deprecated and should no longer be used, but type 2 (RH2) may be used for Mobile IPv6 (MIPv6). The Hop-by-Hop Options Header is used for additional functionality that must be processed by every router along the path and the Destination Options Header for additional functionality that is only processed by the packet's destination. None of these is widely used; see the [IANA website](#) for an overview. See [RFC 4942](#) for detailed information about extension headers and fragmentation.

5.4. Hardware versus software filters/firewalls

As a general rule, hardware and software firewalls operate in a fundamentally different way that is very important to recognize. In general, hardware firewalls that don't support IPv6 will simply not "see" IPv6 packets and therefore not let them through. (Please make sure that this is true for your hardware firewall before deployment.) Software firewalls and filters that filter IPv4, on the other hand, simply don't apply to IPv6, and thus typically let **all** IPv6 packets through.

So when setting up a firewall or when creating filters on routers or servers, make sure that **both** IPv4 and IPv6 are filtered as intended. In many cases, separate filter rules need to be created for IPv4 and IPv6. However, some firewalls or filters (such as [PF](#)) allow filtering both protocols using a single filter rule.

-  Many high-end routers, and possibly also firewalls, use hardware-based packet filtering where the filtering hardware can only look at the first 64, 128 or 256 bytes of a packet. This means that such filters may be defeated by including large or many extension headers. See [IPv6 ACL bypass](#).

5.5. (Unique) Site Local addresses

Site-local IPv6 addresses were intended to be used for systems that don't connect to the global internet. In that sense, they're similar to [RFC 1918](#) IPv4 addresses, with the difference that [RFC 1918](#) addresses are most commonly used behind NATs. However, the IETF couldn't come up with a good way for routers to handle the situation where they are connected to two private networks that both use the same range of site-local addresses.

So the IETF created a different kind of site-local addresses: Unique Site Local Addresses (ULAs). The idea here is that each network uses its own range of site-local addresses. Two types were defined: one where a central registry would give out ULA prefixes and one where users generate their own ULA prefix. The first type was never put into use, so organizations that want to use site-local addresses should generate one (or more) /48 ULA prefixes based on a random number. Several websites will do this for you, or consult [RFC 4193](#). 40 bits in the ULA prefix are randomly generated, so it's very unlikely for two users of ULAs, and especially two users of ULAs who know each other, to generate the same ULA prefix, as long as both use the correct procedure for generating it.

ULAs are taken from fc00::/7 and "are not expected to be routable on the global internet".

5.6. Global unicast addresses

At this time, only addresses within 2000::/3 are defined as global unicast addresses. (In other words, regular public addresses.) So almost three quarters of the IPv6 address space is reserved for future use. However, this doesn't necessarily mean that filtering out all other addresses is a good idea. Many people installed filters to filter out unused address space in IPv4, but then failed to update these filters as additional address space was put into use. And the class E IPv4 address block (240.0.0.0/4) can't be used on many systems "because it's set aside for future use". With the result that now that we need the extra addresses, they're unusable.

So according to the IPv6 specifications, any IPv6 address space that's reserved for future use must be treated as global unicast address space. Should you choose to filter non-2000::/3 address space, make sure there are procedures in place to monitor the deployment of additional global unicast address space and then change the filters accordingly.

5.7. Stateful firewalling to replace NAT

Hosts that were protected against unsolicited incoming packets by an IPv4 NAT no longer have such protection when they're given global unicast IPv6 addresses. It's important to recognize that most NATs aren't airtight: there are several ways to get packets through, especially if the host behind the NAT actively attempts to accomplish this. Also, hosts on the same local network behind the same NAT aren't protected from each other. As such, hosts must be prepared to deal with unexpected incoming packets of various kinds.

That being said, it's often desirable to keep certain kinds of packets from reaching the local network. Without NAT, this can be done with a stateful firewall. A stateful firewall keeps track of outgoing packets and then allows incoming packets that look like responses to earlier outgoing packets. Incoming packets that don't match previously established TCP sessions or outgoing UDP packets are dropped.

In this regard, a stateful firewall purposely implements the functionality that a NAT achieves as a side effect. As such, stateful firewalls are typically much harder to bypass than NATs, with the result that many peer-to-peer applications, including SIP for VoIP and video chat, may have a harder time working over IPv6 in the presence of a stateful firewall than over IPv4 with NAT. However, the advantage of a firewall over NAT is that it can be selectively disabled.

When implementing packet filters and/or a stateful firewall, a good place to start is [RFC 6092](#), which recommends simple security capabilities for home routers/gateways (CPEs), which will often be unmanaged.

5.8. Address prefix and ICMPv6 filtering

Below is an overview of the IPv6 address space, along with the implication of filtering certain prefixes.

Table 3, filtering implications for IPv6 prefixes

Prefix	First bits	Purpose	Filtering implications
::3	000	Special purpose addresses	May be filtered at network border
::1/128		Localhost	In hosts: don't filter / be careful*
::ffff:0:0/96		IPv4-mapped API addresses	In hosts: don't filter / be careful*
64:ff9b::/96	0000 0000 0110 0100	NAT64 well-known prefix	Filter at network border
2000::/3	001	Global unicast	Don't filter
2001::/32		Teredo	Filtering may lead to rare timeouts
2002::/16	0010 0000 0000 0010	6to4	Filtering may lead to some timeouts
4000::/2 - f800::/6	01 - 1111 01	Reserved	Only filter with update procedure in place
fc00::/7	1111 110	Unique Site Local	Filter at network border
fe80::/10	1111 1110 01	Link-local	Don't filter / be careful**, won't go through routers
fec0::/10	1111 1110 11	Old site-local	Filter
ff00::/8	1111 1111	Multicast	Don't filter, but only enable multicast routing where needed

* Accidentally blocking packets to/from ::1/128 or ::ffff:0:0/96 inside a host will create big problems. However, these packets can safely be filtered in routers as part of a filter rule that blocks ::3.

** Although a case can be made for filtering packets to/from fe80::/10, the link-local prefix, in routers or firewalls, doing so serves no useful purpose. Routers won't forward these packets anyway, and such filters are dangerous because they can easily block legitimate link-local packets, including Neighbor Discovery and routing protocols.



Networks should only allow outgoing IPv6 packets if those packets have a source address that belongs to the network in question. See [RFC 2827](#) / Best Current Practice 38.

Table 4 contains an overview of ICMPv6 error messages (types below 128) and the implications of filtering them. Note that when a stateful firewall is used, explicitly filtering incoming ICMPv6 error messages should be unnecessary, the firewall should filter out these messages if they're not in response to earlier outgoing packets.


Table 4, filtering implications for ICMPv6 error messages

Type	Name	Purpose	Filtering implications
1	Destination Unreachable	Indicate reachability errors	Filtering will lead to timeouts
2	Packet Too Big	Path MTU Discovery	Filtering will make communication with certain destinations impossible
3	Time Exceeded	Traceroute	Filtering will block traceroute, RFC 4890 recommends allowing Code 0
4	Parameter problem	Indicate protocol errors	Hard to say, type is uncommon, RFC 4890 recommends allowing Code 1 + 2

Finally, in table 5 an overview of ICMPv6 informational messages (types above 127) and the implications of filtering them.

Table 5, filtering implications for ICMPv6 informational messages

Type	Name	Purpose	Filtering implications
128	Echo Request	Ping	Ping won't work, may impact Teredo
129	Echo Reply	Ping	Ping won't work, may impact Teredo
130 - 132, 143	MLD	Tell routers where to forward multicasts	Multicast applications likely won't work
133	Router Solicitation	Trigger Router Advertisement	Default gateway and address configuration will be much slower
134	Router Advertisement	Default gateway, address configuration	Hosts won't discover default gateway and won't have global unicast addresses
135	Neighbor Solicitation	Local communication, address configuration	Hosts will be unable to communicate over IPv6
136	Neighbor Advertisement	Local communication, address configuration	Hosts will be unable to communicate over IPv6
137	Redirect Message	Optimize packet flow with multiple routers	Packets may have to flow through an extra router
139	Node Information Query	Debugging	None
140	Node Information Response	Debugging	None
144 - 147	Mobile IPv6	Mobile IPv6	MIPv6 will not work (MIPv6 is not widely implemented)
148 - 149	SEND	SEcure Neighbor Discovery	SEND will not work (SEND is not widely implemented)
152 - 153	Multicast Router Discovery	Optimize multicast forwarding by switches (like IGMP snooping)	Switches may block multicasts or forward them unnecessarily

 Note the difference between IPv4, where ARP messages are never filtered because those are no IPv4 packets, and IPv6, where Neighbor Discovery packets may be filtered by accident. Cisco routers use an "implicit deny" where all packets that aren't explicitly allowed are filtered out, but this implicit deny doesn't apply to Neighbor Discovery packets. However, an explicit "deny all" clause **does** filter Neighbor Discovery packets, so when such a clause is used, ND packets must be explicitly allowed first.

Also note that it is unnecessary to filter Neighbor Discovery, Redirect and Multicast Listener/Router Discovery and SEcure Neighbor Discovery messages at the network border, as hosts will only process these messages if they're generated locally. (The Hop Limit = 255 security check.) Should you wish to filter them anyway to avoid possible implementation bugs in hosts, make sure that ND packets **to and from the router** are allowed by the filter.

For ICMPv6, a filter that allows (whitelists) necessary and desired ICMP types and blocks all others would be appropriate.

See [RFC 4890](#) for a full discussion on filtering ICMPv6 messages.

As noted in section 5.2, a Fragment Header may be used by attackers to get past filters. However, it's usually not possible to filter all packets with a Fragment Header, because legitimate packets may be fragmented. The most common example would be UDP DNS packets used for DNSSEC. These are often larger than the maximum Ethernet size of 1500 bytes and are thus fragmented. Neighbor Discovery packets with a Fragment Header may be filtered.

In general, there are two approaches to filtering: enumerating what must be blocked and allow everything else (default allow) or enumerating what is allowed and block everything else (default deny). Because it's hard to predict whether a given communication session will happen over IPv4 or IPv6 (if both are available), a consistent user experience is best served by using either default allow or default deny for both IPv4 and IPv6. As such, it's best to use default deny for IPv6 if default deny is used with IPv4 and default allow for IPv6 if default allow is used with IPv4.

5.9. Prefix length filtering in BGP

In general, IPv6 routing protocols function very similar to IPv4 routing protocols. A notable difference is that it's understood that only /24 and shorter prefixes are propagated in BGP. For IPv6, there is not a similar de facto minimum prefix length. The most common IPv6 prefix length seen in BGP is /32, which is the size of prefixes given to small-to-medium-sized ISPs. Large ISPs have shorter prefixes. In addition to this, end-user organizations may use /48 provider independent prefixes. So it's possible to filter out prefixes longer than /48 without issues.

With IPv4, it's not unheard of for an ISP with, for instance, a /16 to leak the 256 /24s that make up that /16. With IPv6, a similar occurrence would be quite devastating, as a /32 given to an ISP encompasses 65000 /48s, which is probably enough to make many BGP routers run out of memory. So in addition to using the appropriate inbound and outbound prefix filters and having MD5 passwords on BGP sessions, it's important to set maximum prefix limits on BGP sessions to avoid such a scenario.

6. DISABLING IPV6

Should you wish to prevent having IPv6 on your network, there are several ways to accomplish this. However, they're not equally effective.

An obvious start would be to simply not turn on IPv6 in routers and not let firewalls forward IPv6 packets. However, this still allows IPv6 packets to be exchanged between hosts connected

to the same subnet, and IPv6 packets may be tunneled to/from the internet. Some of these tunneled packets are easy to filter (protocol 41 for 6to4, UDP port 3544 for Teredo), but there are many tunneling mechanisms and users may try to encrypt their tunnel packets or use alternative port numbers.

Another way is to disable IPv6 in the network configuration of all hosts connected to the network. However, recent versions of Mac OS X no longer allow turning off IPv6 completely through the graphical user interface. Of course this approach is only possible when all devices that connect to the network can be controlled by network or system administrators.

Last but not least, switches and Wi-Fi base stations may be set up to filter the IPv6 ethertype, 0x86dd, making it impossible for IPv6 packets to be transmitted across the switches and base stations in question. In this case, also filter IPv6 tunnel protocols, such as protocol 41 (6to4, ISATAP) and UDP port 3544 (Teredo). See [RFC 7059](#) for more information.

 Some Windows components or applications need IPv6 to function. Therefore, Microsoft "recommends that you leave IPv6 enabled, even if you do not have an IPv6-enabled network". See the Technet [IPv6 for Windows FAQ](#).

7. ACKNOWLEDGMENTS

Rogier Spoor, SURFnet, commissioned this work.

Christiaan Ottow, Pine Digital Security, reviewed the document and provided many suggestions for improvements.

Roel Hoek, Universiteit Twente, reviewed and provided suggestions for improvements.

Ilijtsch van Beijnum wrote the text.

8. REFERENCES

SURFnet, "Een IPv6 nummerplan opstellen". <http://www.surf.nl/kennis-en-innovatie/kennisbank/2013/whitepaper-ipv6-nummerplan-opstellen.html>

Christiaan Ottow et al. "The Impact of IPv6 on Penetration Testing." Information and Communication Technologies. Springer Berlin Heidelberg, 2012.

Wikipedia, "PF (firewall)". [http://en.wikipedia.org/wiki/PF_\(firewall\)](http://en.wikipedia.org/wiki/PF_(firewall))

Microsoft Technet, "IPv6 for Microsoft Windows: Frequently Asked Questions". <http://technet.microsoft.com/en-us/network/cc987595.aspx>

Saku Ytti, "IPv6 ACL bypass". <http://blog.ip.fi/2011/08/ipv6-acl-bypass.html>

Marc Heuse, "IPv6 Insecurity Revolutions". Hack in the Box, Kuala Lumpur, 2012. <http://www.youtube.com/watch?v=t9d7p3zxoIM>, <http://conference.hitb.org/hitbsecconf2012kul/materials/>

RFC 1918 Address Allocation for Private Internets. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear. February 1996. (Format:TXT=22270 bytes) (Obsoletes RFC1627, RFC1597) (Updated by RFC6761) (Also BCP0005) (Status: BEST CURRENT PRACTICE)

RFC 2827 Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. P. Ferguson, D. Senie. May 2000. (Format:TXT=21258 bytes)

(Obsoletes RFC2267) (Updated by RFC3704) (Also BCP0038) (Status: BEST CURRENT PRACTICE)

RFC 3756 IPv6 Neighbor Discovery (ND) Trust Models and Threats. P. Nikander, Ed., J. Kempf, E. Nordmark. May 2004. (Format:TXT=56674 bytes) (Status: INFORMATIONAL)

RFC 3971 SEcure Neighbor Discovery (SEND). J. Arkko, Ed., J. Kempf, B. Zill, P. Nikander. March 2005. (Format:TXT=123372 bytes) (Updated by RFC6494, RFC6495, RFC6980) (Status: PROPOSED STANDARD)

RFC 3964 Security Considerations for 6to4. P. Savola, C. Patel. December 2004. (Format:TXT=83360 bytes) (Status: INFORMATIONAL)

RFC 4193 Unique Local IPv6 Unicast Addresses. R. Hinden, B. Haberman. October 2005. (Format:TXT=35908 bytes) (Status: PROPOSED STANDARD)

RFC 4294 IPv6 Node Requirements. J. Loughney, Ed.. April 2006. (Format:TXT=39125 bytes) (Obsoleted by RFC6434) (Updated by RFC5095) (Status: INFORMATIONAL)

RFC 4620 IPv6 Node Information Queries. M. Crawford, B. Haberman, Ed.. August 2006. (Format:TXT=31134 bytes) (Status: EXPERIMENTAL)

RFC 4861 Neighbor Discovery for IP version 6 (IPv6). T. Narten, E. Nordmark, W. Simpson, H. Soliman. September 2007. (Format:TXT=235106 bytes) (Obsoletes RFC2461) (Updated by RFC5942, RFC6980) (Status: DRAFT STANDARD)

RFC 4864 Local Network Protection for IPv6. G. Van de Velde, T. Hain, R. Droms, B. Carpenter, E. Klein. May 2007. (Format:TXT=95448 bytes) (Status: INFORMATIONAL)

RFC 4890 Recommendations for Filtering ICMPv6 Messages in Firewalls. E. Davies, J. Mohacsi. May 2007. (Format:TXT=83479 bytes) (Status: INFORMATIONAL)

RFC 4941 Privacy Extensions for Stateless Address Autoconfiguration in IPv6. T. Narten, R. Draves, S. Krishnan. September 2007. (Format:TXT=56699 bytes) (Obsoletes RFC3041) (Status: DRAFT STANDARD)

RFC 4942 IPv6 Transition/Co-existence Security Considerations. E. Davies, S. Krishnan, P. Savola. September 2007. (Format:TXT=102878 bytes) (Status: INFORMATIONAL)

RFC 5095 Deprecation of Type 0 Routing Headers in IPv6. J. Abley, P. Savola, G. Neville-Neil. December 2007. (Format:TXT=13423 bytes) (Updates RFC2460, RFC4294) (Status: PROPOSED STANDARD)

RFC 5952 A Recommendation for IPv6 Address Text Representation. S. Kawamura, M. Kawashima. August 2010. (Format:TXT=26570 bytes) (Updates RFC4291) (Status: PROPOSED STANDARD)

RFC 6092 Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service. J. Woodyatt, Ed.. January 2011. (Format:TXT=91729 bytes) (Status: INFORMATIONAL)

RFC 6105 IPv6 Router Advertisement Guard. E. Levy-Abegnoli, G. Van de Velde, C. Popoviciu, J. Mohacsi. February 2011. (Format:TXT=20817 bytes) (Status: INFORMATIONAL)

RFC 6145 IP/ICMP Translation Algorithm. X. Li, C. Bao, F. Baker. April 2011. (Format:TXT=76484 bytes) (Obsoletes RFC2765) (Updated by RFC6791) (Status: PROPOSED STANDARD)

RFC 6146 Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. M. Bagnulo, P. Matthews, I. van Beijnum. April 2011. (Format:TXT=107954 bytes) (Status: PROPOSED STANDARD)

RFC 6296 IPv6-to-IPv6 Network Prefix Translation. M. Wasserman, F. Baker. June 2011. (Format: TXT=73700 bytes) (Status: EXPERIMENTAL)

RFC 6434 IPv6 Node Requirements. E. Jankiewicz, J. Loughney, T. Narten. December 2011. (Format: TXT=64407 bytes) (Obsoletes RFC4294) (Status: INFORMATIONAL)

RFC 6724 Default Address Selection for Internet Protocol Version 6 (IPv6). D. Thaler, Ed., R. Draves, A. Matsumoto, T. Chown. September 2012. (Format: TXT=74407 bytes) (Obsoletes RFC3484) (Status: PROPOSED STANDARD)

RFC 6936 Client Link-Layer Address Option in DHCPv6 G. Halwasia, S. Bhandari, W. Dec. May 2013. (Format: TXT=14739 bytes) (Status: PROPOSED STANDARD)

RFC 6980 Security Implications of IPv6 Fragmentation with IPv6 Neighbor Discovery. F. Gont. August 2013. (Format: TXT=20850 bytes) (Updates RFC3971, RFC4861) (Status: PROPOSED STANDARD)

RFC 7059 A Comparison of IPv6-over-IPv4 Tunnel Mechanisms. S. Steffann, I. van Beijnum, R. van Rein. November 2013. (Format: TXT=98886 bytes) (Status: INFORMATIONAL)

SURFnet

Radboudkwartier 273
3511 CK Utrecht
The Netherlands

PO Box 19035
3501 DA Utrecht
The Netherlands

+31 (0)30 230 5305
www.surfnet.nl

