Copyright notice

www.si6networks.com

Contents

This slideware contains **part** of the materials used for the training
"**Hacking IPv6 Networks**"
taught during the DEEPSEC 2011 Conference.

More inforamation available at:
www.hackingipv6networks.com

www.si6networks.com

Hacking IPv6 Networks

Fernando Gont

DEEPSEC 2011

Vienna, Austria. November 15-16, 2011

About

- I have worked in security assessment of communication protocols for:
  - UK NISCC (National Infrastructure Security Co-ordination Centre)
  - UK CPNI (Centre for the Protection of National Infrastructure)
- Currently working for SI6 Networks (http://www.si6networks.com)
- Member of R+D group CEDI at UTN/FRH
- Involved in the Internet Engineering Task Force (IETF)
- More information at: http://www.gont.com.ar

Agenda (I)

- Objectives of this training
- Motivation for IPv6, and current state of affairs
- Brief comparision between IPv6 and IPv4
- IPv6 Addressing Architecture
- IPv6 Header Fields
- IPv6 Extension Headers
- IPv6 Options
- Internet Control Message Protocol version 6 (ICMPv6)
- Neighbor Discovery for IPv6
- IPv6 Address Resolution
- Stateless Address Auto-configuration (SLAAC)

Agenda (II)

- IPsec
- Multicast Listener Discovery
- Dynamic Host Configuration Protocol version 6 (DHCPv6)
- DNS support for IPv6
- IPv6 firewalls
- Transition/co-existence technologies (6to4, Teredo, ISATAP, etc.)
- Network reconnaissance in IPv6
- Security Implications of IPv6 on IPv4-only networks
- IPv6 deployment considerations
- Key areas in which further work is needed
- Some conclusions

Brief introduction to IPv6

So... what is this “IPv6” thing about?

- IPv6 was developed to address the exhaustion of IPv4 addresses
- IPv6 has not yet seen broad/global deployment (current estimations are that IPv6 traffic is less than 1% of total traffic)
- However, general-purpose OSes have shipped with IPv6 support for a long time – hence part of your network is already running IPv6!
- Additionaly, ISPs and other organizations have started to take IPv6 more seriosly, partly as a result of:
  - Exhaustion of the IANA IPv4 free pool
  - Awareness activities such as the “World IPv6 Day”
  - Imminent exhaustion of the free pool of IPv4 addresses at the different RIRs
- It looks like IPv6 is finally starting to take off...

Motivation for this training

- A lot of myths have been created around IPv6 security:
  - Security as a key component of the protocol
  - Change from network-centric to host-centric paradigm
  - Increased use of IPsec
  - etc.
- They hace lead to a general misunderstanding of the security properties of IPv6, thus negatively affecting the emerging (or existing) IPv6 networks.
- This training separates fudge from fact, and offers a more realistic view of “IPv6 security”
  - At a conceptual level, it is meant to influence the way in which you think about IPv6 security (and IPv6 in general)
  - We will also reproduce some attacks and play with configuration information, to keep it real (“walk the talk”)

Some general considerations about IPv6 security

Some interesting aspects about IPv6

- We have much less experience with IPv6 than with IPv4
- IPv6 implementations are much less mature than their IPv4 counterparts.
- Security products (firewalls, NIDS, etc.) have less support for IPv6 than for IPv4
- The complexity of the resulting network will greatly increase during the transition/co-existence period:
  - Two internetworking protocols (IPv4 and IPv6)
  - Increased use of NATs
  - Increased use of tunnels
  - Use of a plethora of transition/co-existence mechanisms
- Lack of trained human resources

…and even then, IPv6 will be in many cases the only option on the table to remain in this business

Brief comparision between IPv6 and IPv4

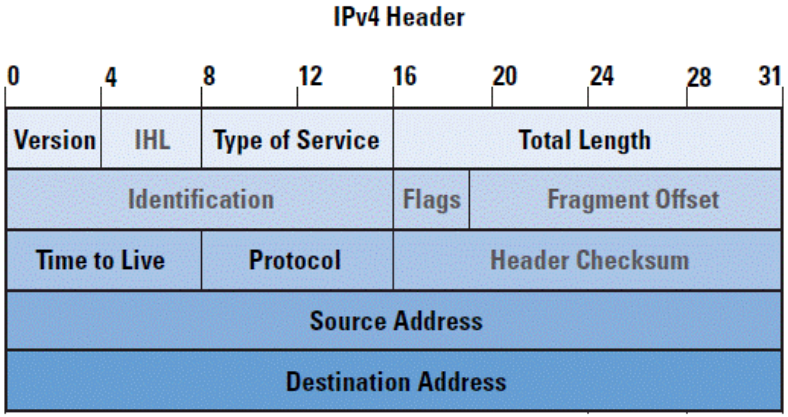
Brief comparision between IPv6 and IPv4

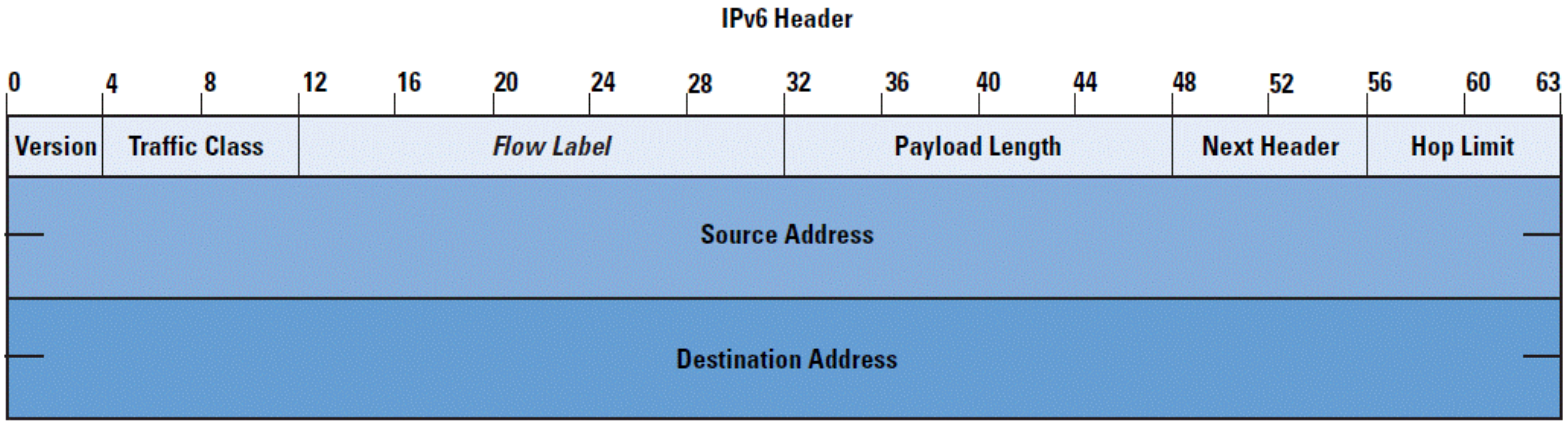
- IPv6 and IPv4 are very similar in terms of ***functionality*** (but not in terms of ***mechanisms***)

	IPv4	IPv6
Addressing	32 bits	128 bits
Address Resolution	ARP	ICMPv6 NS/NA (+ MLD)
Auto-configuration	DHCP & ICMP RS/RA	ICMPv6 RS/RA & DHCPv6 (recommended) (+ MLD)
Fault Isolation	ICMP	ICMPv6
IPsec support	Optional	Recommended (not mandatory)
Fragmentation	Both in hosts and routers	Only in hosts

Brief comparision of IPv4 and IPv6 (II)

- Header formats:



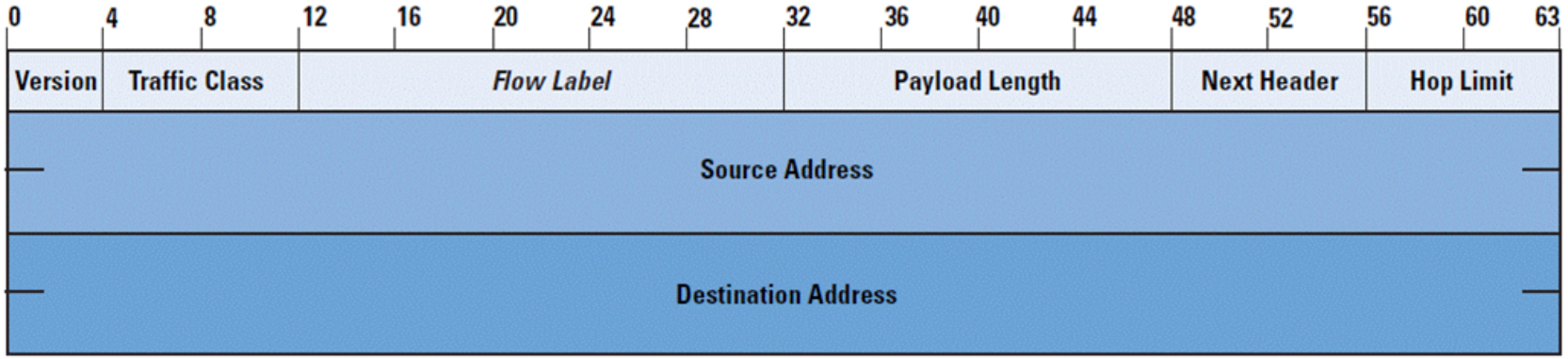


IPv6 header fields

Basic header fields

IPv6 header

- **Fixed-length** (40-bytes) header



Version

- Identifies the Internet Protocol version number (“6” for IPv6)
- It should match the “Protocol” specified by the underlying link-layer protocol
  - If not, link-layer access controls could be bypassed
- All implementations tested so far properly validate this field
  - Must admit I've learned it the hard way :-)

Traffic Class

- Same as IPv4's "Differentiated Services"
- No additional "Quality of Service" (QoS) feature in IPv6 (sorry)
- "Traffic Class" could be leveraged to receive differentiated service
- The Traffic Class should be policed at the network edge
- In summary, no differences with respect to IPv4 QoS

Flow Label

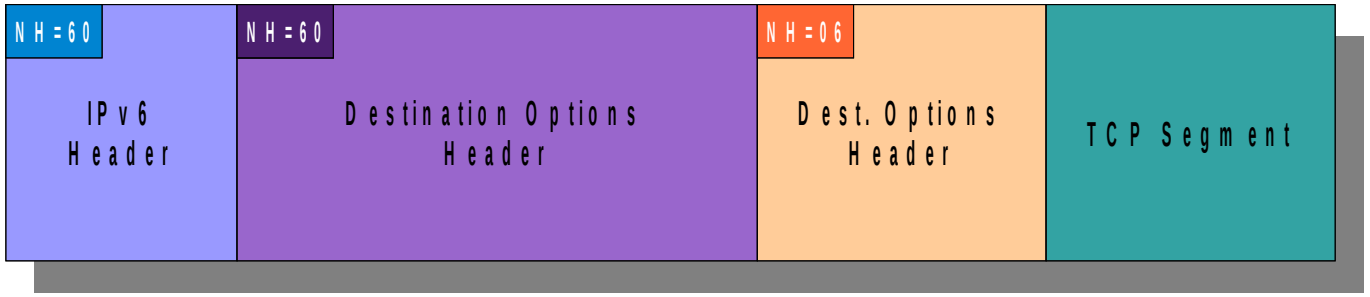
- Finding the transport-protocol port-numbers con probe to be difficult in IPv6
- The Flow Label is thus meant help with load sharing
- The three-tuple {Source Address, Destination Address, Flow Label} identifies a communication flow
- Currently unused by many stacks
  - Some stacks simply set it to 0 for all packets
  - Other stacks set it improperly
- Specification of this header field has just been published:
- Potential vulnerabilities depend on predictable Flow:
  - Might be leveraged to perform “dumb” (stealth) address scans
  - Might be leveraged to perform Denial of Service attacks

Payload Length

- Specifies the length of the IPv6 **payload** (not of the entire packet)
- Maximum IPv6 packet is 65855 bytes. However, IPv6 "Jumbograms" can be specified.
- A number of sanity checks need to be performed. e.g.:
  - The IPv6 Payload Length cannot be larger than the "payload size" reported by the link-layer protocol
- All stacks seem to properly validate this field

Next Header

- Identifies the header/protocol type following this header.
- IPv6 options are included in “extension headers”
  - These headers sit between the IPv6 header and the upper-layer protocol
  - There may be multiple instances of multiple extension headers
- Hence, IPv6 follow a “header chain” type structure. e.g.,



Hop Limit

- Analogous to IPv4's "Time to Live" (TTL)
- Identifies the number of network links that a packet may traverse
- Packets are discarded when the Hop Limit is decremented to 0.
- Different OSes use different defaults for the "Hop Limit" (typically a power of two: 64, 128, etc.)
- Could (in theory) be leveraged for:
  - Detecting the Operating System of a remote node
  - Fingerprinting a remote physical device
  - Locating a node in the network topology
  - Evading Network Intrusion Detection Systems (NIDS)
  - Reducing the attack exposure of some hosts/applications

Hop Limit: Fingerprinting the remote OS Devices

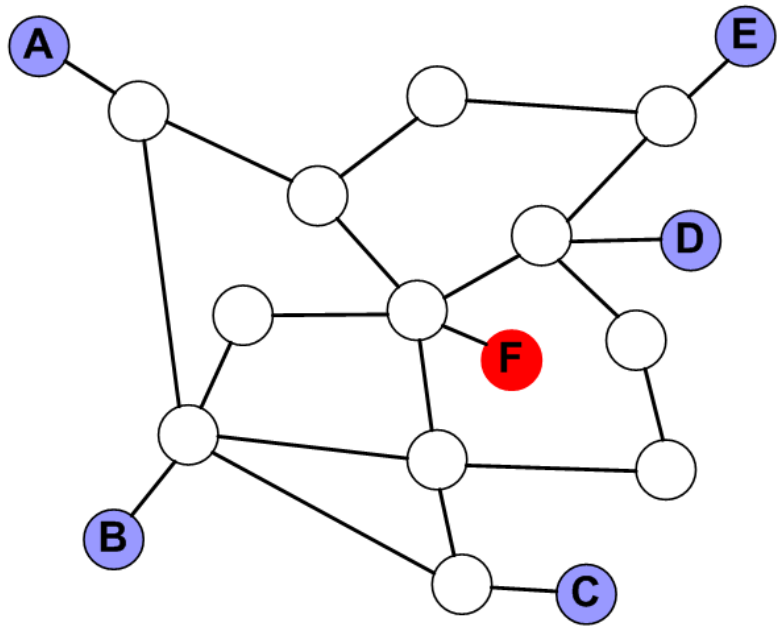
- There are a few default values for the Hop Limit in different OSes
- Based on the received Hop Limit, the original Hop Limit can be inferred
- Example:
  - We receive packets with a Hop Limit of 60
  - We can infer the original Hop Limit was 64
  - We can determine a set of possible remote OSes
- Note: mostly **useless**, since:
  - There is only a reduced number of default “Hop Limit” values
  - Fingerprinting graularity is too coarse

Hop Limit: Fingerprinting Physical Devices

- If packets originating from the same IPv6 addresses contain very different “Hop Limits”, they might be originated by different devices.
- Example:
  - We see this traffic:
    - Packets from FTP server 2001:db8::1 arrive with a “Hop Limit” of 60
    - Packets from web server 2001:db8:::2 arrive with a “Hop Limit” of 124
  - We infer:
    - FTP server sets the Hop Limit to 64, and is 4 “routers” away
    - Web server sets the Hop Limit to 128, and is 4 “routers” away
- Note: mostly **useless**, since:
  - It requires different OSes or different locations behind the “middle-box”
  - There is only a reduced number of default “Hop Limit” values

Hop Limit: Locating a Node

- Basic idea: if we are receiving packets from a node and assume that it is using the default “Hop Limit”, we can infer the orginal “Hop Limit”
- If we have multple “sensors”, we can “triangulate” the position of the node



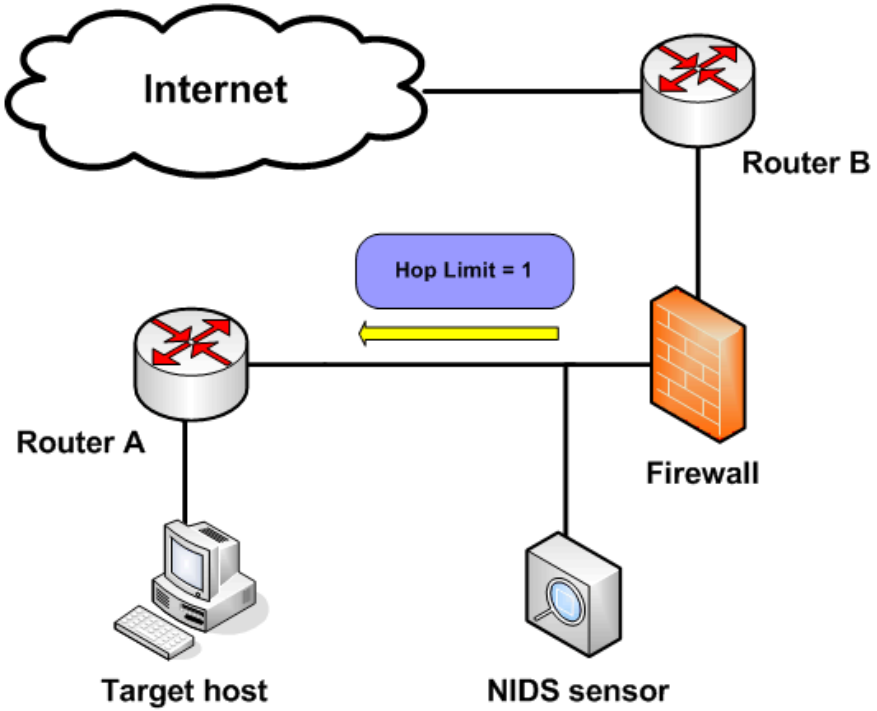
Source	Hop Limit
A	61
B	61
C	61
D	62

F is the only node that is:

- 3 “routers” from A
- 3 “routers” from B
- 3 “routers” from C
- 2 “routers” from D

Hop Limit: Evading NIDS

- Basic idea: the attacker sets the Hop Limit to a value such that the NIDS sensor receives the packet, but the target host does not.
- Counter-measure: Normalize the “Hop Limit” at the network edge (to 64) or block incomming packets with very small “Hop Limits” (e.g., smaller than 10)



Hop Limit: Improving Security (GTSM)

- GTSM: Generalized TTL Security Mechanism
  - Named after the IPv4 “TTL” field, but same concept applies to IPv6
  - It reduces the host/application exposure to attacks
- The Hop Limit is set to 255 by the source host
  - The receiving host requires the Hop Limit of incoming packets to be of a minimum value (255 for link-local applications)
  - Packets that do not pass this check are silently dropped
- This mechanism is employed by e.g., BGP and IPv6 Neighbor Discovery
- Example:

12:12:42.086657 2004::20c:29ff:fe49:ebdd > ff02::1:ff00:1: icmp6: neighbor sol: who has 2004::1(src lladdr: 00:0c:29:49:eb:dd) (len 32, **hlim 255**)
12:12:42.087654 2004::1 > 2004::20c:29ff:fe49:ebdd: icmp6: neighbor adv: tgt is 2004::1(RSO)(tgt lladdr: 00:0c:29:c0:97:ae) (len 32, **hlim 255**)

IPv6 Addressing Architecture

Brief Overview

- The main driver for IPv6 is its increased address space
- IPv6 uses 128-bit address (vs. IPv4's 32-bit addresses)
- Similarly to IPv4,
  - Addresses are aggregated into “prefixes” (for routing purposes)
  - There are different address types (unicast, anycast, and multicast)
  - There are different address scopes (link-local, global, etc.)
- However, at any given time, several IPv6 addresses, of multiple types and scopes are used. For example,
  - One or more unicast link-local address
  - One or more global unicast address
  - One or more link-local address

IPv6 Address Types

- The address type can be identified as follows:

Address Type	IPv6 prefix
Unspecified	::/128
Loopback	::1/128
Multicast	FF00::/8
Link-local unicast	FE80::/10
Unique Local Unicast	FC00::/7
Global Unicast	(everything else)

IPv6 Address Types

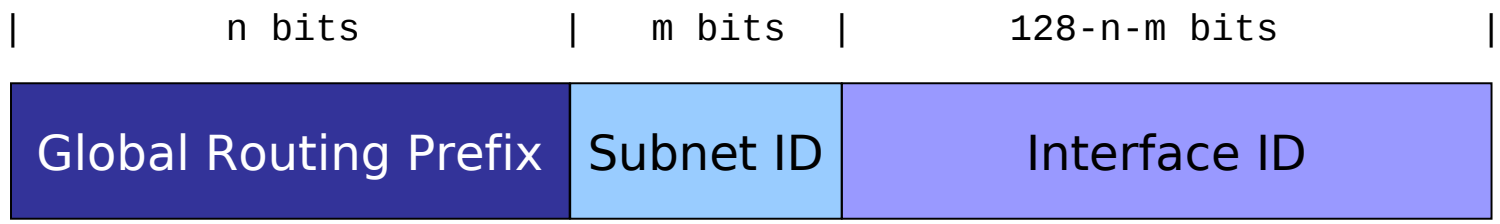
Unicast Addresses

Unicast Addresses

- Global unicast
  - Meant for communication on the public Internet
- Link-local unicast
  - Meant for communication within a network link/segment
- Site-local unicast
  - Deprecated (were meant to be valid only within a site)
- Unique Local unicast
  - Are expected to be globally unique, but not routable on the public Internet

Global Unicast Addresses

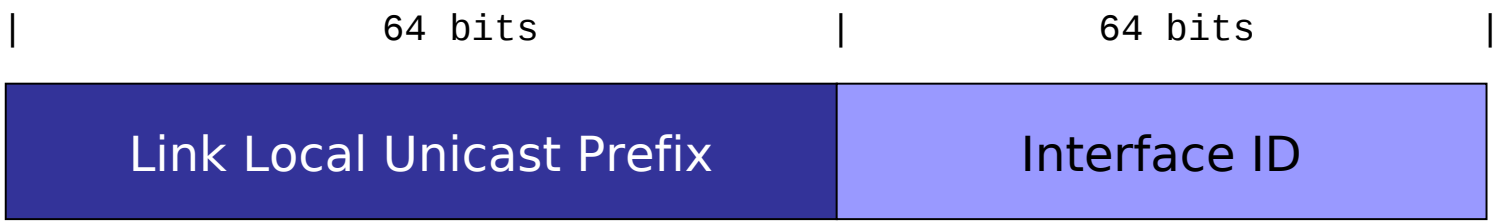
- Syntax of the global unicast addresses:



- The interface ID is typically 64-bis
- The Interface-ID can be selected with different criteria:
  - Use modified EUI-64 format identifiers (embed the MAC address)
  - “Privacy Addresses” (or some of their variants)
  - Manually-configured (e.g., 2001:db8::1)
  - As specified by some specific transition-co-existence technology

Link-local Unicast Addresses

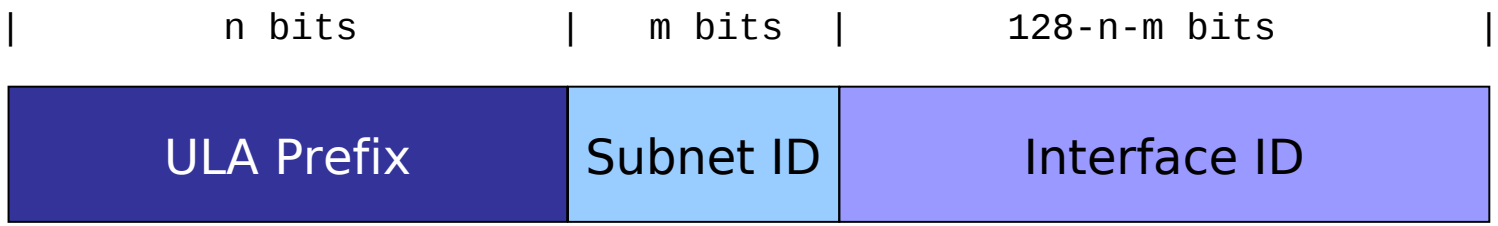
- Syntax of the link-local unicast addresses:



- The Link-Local Unicast Prefix is fe80::/64
- The interface ID is typically set to the modified EUI-64 format identifiers (embed the MAC address)

Unique-local Unicast Addresses

- Syntax of the unique-local unicast addresses:



- The interface ID is typically 64-bis
- The Interface-ID can be selected with different criteria:
  - Use modified EUI-64 format identifiers (embed the MAC address)
  - “Privacy Addresses” (or some of their variants)
  - Manually-configured (e.g., fc00::1, fc00::2, etc.)
  - As specified by some specific transition-co-existence technology

Modified EUI-64 Identifiers

- They are constructed from e.g. Ethernet addresses.
- The word “fffe” is inserted between the OUI and the rest of the Ethernet
- They are constructed from e.g. Ethernet addresses:
  - The “universal” (bit 6, left to right) is set to 1
  - The word 0xfeff is inserted between the OUI and the rest of the address

Example:

- Ethernet address: 00:1b:38:83:d8:3c
- We set bit 6 to 1, and get: 02:1b:38:83:d8:3c
- We insert the word 0xfffe and get: 021b 38ff fe83 d83c
- This would lead to e.g. the IPv6 address: fe80::21b:38ff:fe83:d83c

IPv6 Address Types

Multicast Addresses

Multicast Addresses

- Identify a set of nodes
- Can be of different scopes (interface local, link-local, global, etc.)
- Some examples:

Multicast address	Use
FF01:0:0:0:0:0:0:1	All nodes (interface-local)
FF01:0:0:0:0:0:0:2	All routers (interface-local)
FF02:0:0:0:0:0:0:1	All nodes (link-local)
FF02:0:0:0:0:0:0:2	All routers (link-local)
FF05:0:0:0:0:0:0:2	All routers (site-local)
FF02:0:0:0:0:1:FF00::/104	Solicited-Node

Solicited-node multicast addresses

- Used for address resolution (Neighbor Discovery)
- They avoid the use of broadcasts, which degrade network performance
- They are constructed from the prefix ff02:0:0:0:0:1:ff00::/104
- The least-significant 24 bits are copied from the original address
- Example:
  - We have the IPv6 address fc00::1::21b:38ff:fe83:d83c
  - The resulting solicited-node multicast address is: ff02::1:ff83:d83c

Mapping IPv6 multicast to Ethernet

- The mapping of IPv6 multicast addresses to Ethernet addresses is straightforward (no protocol is needed)
- The first two bytes of the Ethernet address are set to “33:33”
- The address is completed with the four least-significant bytes of the IPv6 address
- Example:
  - We have the IPv6 multicast address ff02::1:ff83:d83c
  - The resulting multicast Ethernet address is: 33:33:ff:83:d8:3c

IPv6 Address Types

Anycast Addresses

Anycast Addresses

- Identify a node belonging to a set of nodes (e.g., some DNS server, some DHCP server, etc.)
- Packets sent to an anycast address are sent only to one of those nodes (the nearest one, as from the point of view of the routing protocols).
- Only a few anycast addresses have been specified:
  - Subnet-router

IPv6 Addressing

Implications on End-to-End Conectivity

Brief overview

- The IPv4 Internet was based on the so-called “End to End” principle:
  - Dumb network, smart hosts
  - Any node can establish a communication instance with any other node in the network
  - The network does not care about what is inside internet-layer packets
- It is usually argued that the “end-to-end principle” allows for “innovation”
- Deployment of some devices (mostly NATs) have basically elimintated the “end-to-end” property of the Internet
- With the increased IPv6 address space, it is expected that each device will have a globally-unique address, and NATs will be no longer needed.

Some considerations

Myth: "IPv6 will return the End-to-End principle to the Internet"

- It is assumed that the possibility of glbal-addresses for every host will return the "End-to-End" principle to the Internet.
- However,
  - Global-addressability does not necessarily imply "end-to-end" connectivity.
  - Most production networks don't really care about innovation, but rather about getting work done.
  - Users expect to use in IPv6 the same services currently available for IPv4 without "end-to-end" connectivity (web, email, social networks, etc.)
- Thus,
  - End-to-end connectivity is not necessarily a desired property in a production network (e.g., may increase host exposure unnecessarily)
  - A typical IPv6 subnet will be protected by a stateful firewall that only allows "return traffic"

IPv6 Addressing

Implications on Network Reconnaissance

Implications on “brute-force scanning”

- If we assume that host addresses are uniformly distributed over the subnet address space (/64), IPv6 brute force scans would be virtually impossible.
- However, experiments (*) have shown that this is not necessarily the case
- IPv6 addresses are usually follow some of the following patterns:
  - SLAAC (Interface-ID based on the MAC address)
  - IPv4-based (e.g., 2001:db8::192.168.10.1)
  - “Low byte” (e.g., 2001:db8::1, 2001:db8::2, etc.)
  - Privacy Addresses (Random Interface-IDs)
  - “Wordy” (e.g., 2001:db8::dead:beef)
  - Related to specific transition-co-existence technologies(e.g., Teredo)

(*) Malone, D. 2008. *Observations of IPv6 Addresses*. Passive and Active Measurement Conference (PAM 2008, LNCS 4979), 29–30 April 2008.

Some real-world data….

- [Malone, 2008] (*) measures how IPv6 addreses are assigned to hosts and routers:

Hosts

Address Type	Percentage
SLAAC	50%
IPv4-based	20%
Teredo	10%
Low-byte	8%
Privacy	6%
Wordy	<1%
Other	<1%

Routers

Address Type	Percentage
Low-byte	70%
IPv4-based	5%
SLAAC	1%
Wordy	<1%
Privacy	<1%
Teredo	<1%
Other	<1%

(*) Malone, D. 2008. *Observations of IPv6 Addresses*. Passive and Active Measurement Conference (PAM 2008, LNCS 4979), 29–30 April 2008.

What about virtualization? (bonus track)

- Virtual machines get virtual network interfaces
- VirtualBox selects MAC addresses from the following OUI:
  - 08:00:27
- Automatically-generated addresses in VMWare ESX Server:
  - Use the OUI: 00:05:69
  - Two bytes of the addresses are taken from the IPv4 address of the host
  - Least significant byte taken from a hash of the VM's configuration file name
- Manually-generated addresses in VMWare ESX Server:
  - Use the OUI: 00:50:56

Some Conclusions and Advice

- IPv6 addresses can be very predictable
- In general, a node does not need to be “publicly reachable” (e.g., servers), unpredictable addresses are desirable
- For servers, security-wise the policy of selection of IPv6 addresses is irrelevant
- For clients, in most scenarios the use of “privacy extensions” (or some variant of it) is generally desirable
- In any case, always consider whether it would be applicable to enforce a packet filtering policy (i.e., if possible, do not rely on “security through obscurity)

Network reconnaissance with multicast

- Multicast addresses can be leveraged for reconnaissance
- Unfortunately (or not) these addresses can only be used locally
- Example with the all-nodes link-local multicast address:
  - ping6 ff02::1%eth0
- Example with the all-routers link-local multicast address:
  - ping6 ff02::2%em0

IPv6 Extension Headers

IPv6 Extension Headers

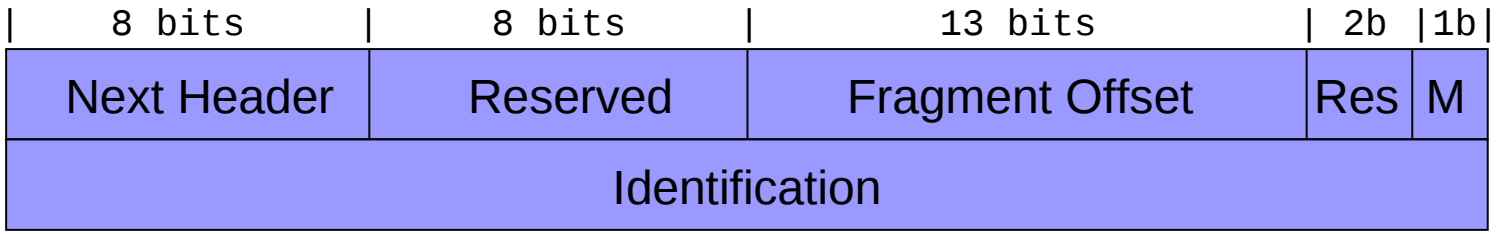
- IPv6 has a fixed header – any options must be included in “extension headers”
- So far, the following Extension Headers have been standardized:
  - Hop-byHop Options
  - Routing
  - Fragment
  - Encapsulating Security Payload (ESP)
  - Authentication
  - Destination Options
- By separating the options into different header, each node processes only the options meant for them (e.g. hosts vs. routers)

IPv6 Extension Headers

Fragment Header

Fragmentation Header

- The fixed IPv6 header does not include support for fragmentation/reassembly
- If needed, such support is added by an Extension Header (Fragmentation Header, NH=44)



  - Fragment Offset: offset of the data following this header, relative to the start of the fragmentable part of the original packet
  - M: “More Fragments” bit, as in the IPv4 header
  - Identification: together with the Source Address and Destination Address identifies fragments that correspond to the same packet

Fragmentation Example (legitimate)

- ping6 output

```
% ping6 –s 1800 2004::1
PING 2004::1(2004::1) 1800 data bytes
1808 bytes from 2004::1: icmp_seq=1 ttl=64 time=0.973 ms

--- 2004::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.973/0.973/0.973/0.000 ms
```

- tcpdump output

```
20:35:27.232273 IP6 2004::5e26:aff:fe33:7063 > 2004::1: frag (0|1448)
ICMP6, echo request, seq 1, length 1448
20:35:27.232314 IP6 2004::5e26:aff:fe33:7063 > 2004::1: frag (1448|360)
20:35:27.233133 IP6 2004::1 > 2004::5e26:aff:fe33:7063: frag (0|1232)
ICMP6, echo reply, seq 1, length 1232
20:35:27.233187 IP6 2004::1 > 2004::5e26:aff:fe33:7063: frag (1232|576)
```

Security Implications

- Some are the same as for IPv4 fragmentation:
  - Stateful operation for a stateless protocol: risk of exhausting kernel memory if the fragment reassembly buffer is not flushed properly

Predictable Identification values (CVE-2011-2699) allow for:

- "stealth" port scanning technique
- DoS attacks (IPv6 ID collisions)

- Others are different:
  - The Identification field is much larger: chances of "IP ID collisions" are reduced
  - Note: Overlapping fragments have been recently forbidden (RFC 5722) – but they are still allowed by many OSes

Fragment Header

IPv6 idle scan

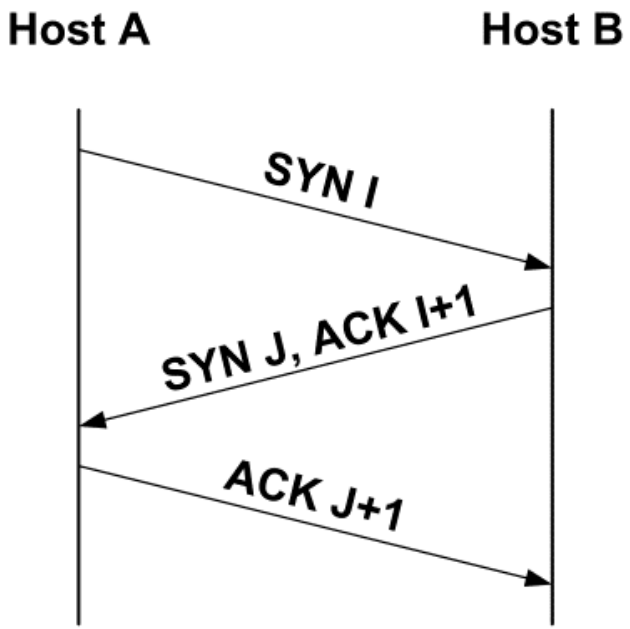
Example of Predictable Identification values

- tcpdump output (`% ping6 -s 1800 2004::1`)

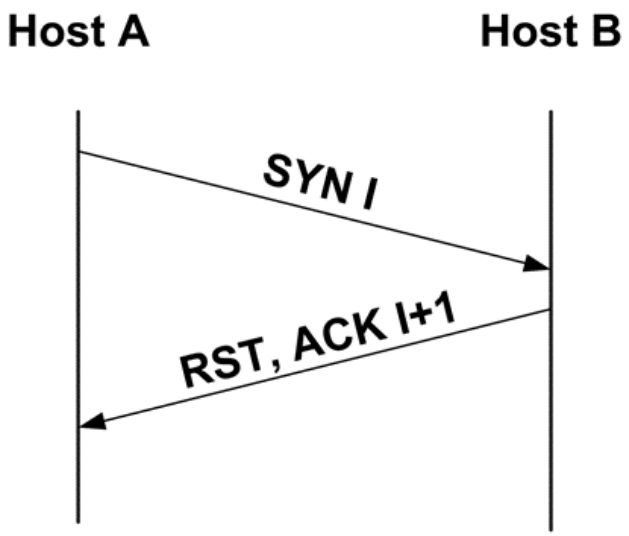
```
1. IP6 (hlim 64, next-header Fragment (44) payload length: 1456)
   2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007a:0|1448) ICMP6, echo
   request, length 1448, seq 1
2. IP6 (hlim 64, next-header Fragment (44) payload length: 368)
   2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007a:1448|360)
3. IP6 (hlim 64, next-header Fragment (44) payload length: 1240) 2004::1 >
   2004::5e26:aff:fe33:7063: frag (0x4973fb3d:0|1232) ICMP6, echo reply,
   length 1232, seq 1
4. IP6 (hlim 64, next-header Fragment (44) payload length: 584) 2004::1 >
   2004::5e26:aff:fe33:7063: frag (0x4973fb3d:1232|576)
5. IP6 (hlim 64, next-header Fragment (44) payload length: 1456)
   2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007b:0|1448) ICMP6, echo
   request, length 1448, seq 2
6. IP6 (hlim 64, next-header Fragment (44) payload length: 368)
   2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007b:1448|360)
7. IP6 (hlim 64, next-header Fragment (44) payload length: 1240) 2004::1 >
   2004::5e26:aff:fe33:7063: frag (0x2b4d7741:0|1232) ICMP6, echo reply,
   length 1232, seq 2
8. IP6 (hlim 64, next-header Fragment (44) payload length: 584) 2004::1 >
   2004::5e26:aff:fe33:7063: frag (0x2b4d7741:1232|576)
```

Revision TCP Connection-Establishment

Connection-established

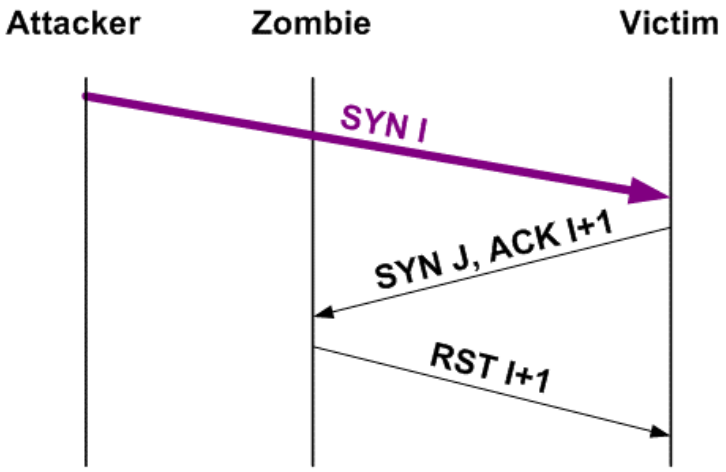


Connection-rejected

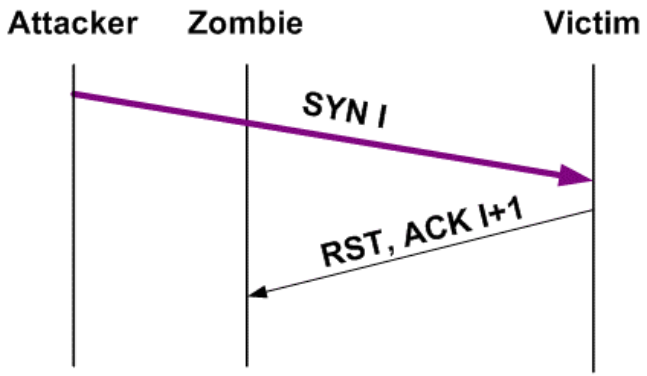


Forged TCP Connection-Establishment

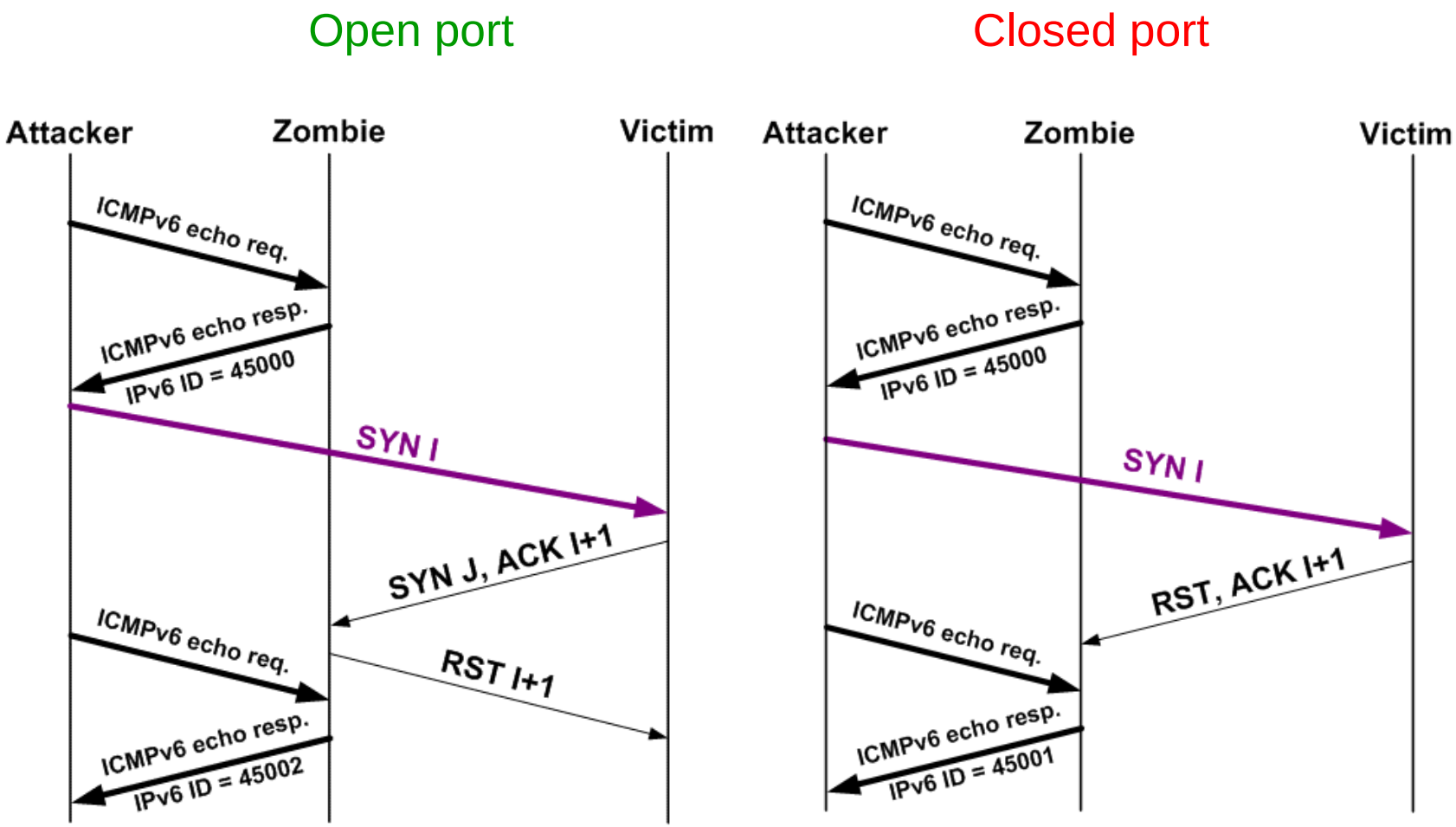
Open port



Closed port



IPv6 Idle Scan



IPv6 Idle Scan

- This “dumb scan” technique allows for a very stealthy port scan
- It only requires an “inactive” host to be used as “zombie”
- Clearly, we didn’t learn the lesson from IPv4

sysctl’s for frag/reassembly

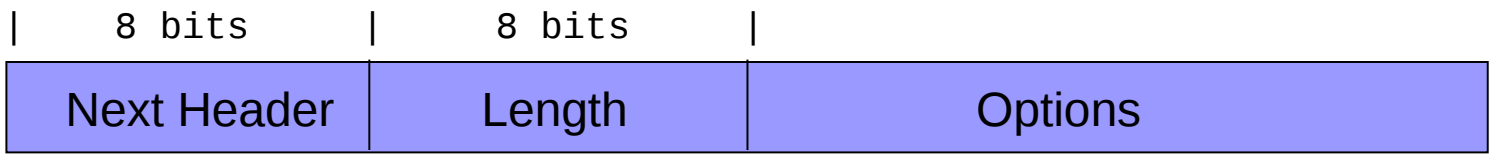
- `net.inet6.ip6.maxfragpackets`: maximum number of fragmented packets the node will accept (defaults to 200 in OpenBSD and 2160 in FreeBSD)
  - 0: the node does not accept fragmented traffic
  - -1: there’s no limit on the number of fragmented packets
- `net.inet6.ip6.maxfrags`: maximum number of fragments the node will accept (defaults to 200 in OpenBSD and 2160 in FreeBSD)
  - 0: the node will not accept any fragments
  - -1: there is no limit on the number of fragments

IPv6 Extension Headers

Hop-by-Hop Options

Hop-by-Hop Options Header

- Identified by a Next Header of 0.
- Carries options meant for routers
  - So far, only “Router Alert” option has been specified
- This header may lead to a DoS at the intervening routers
- Should be policed at the network edge

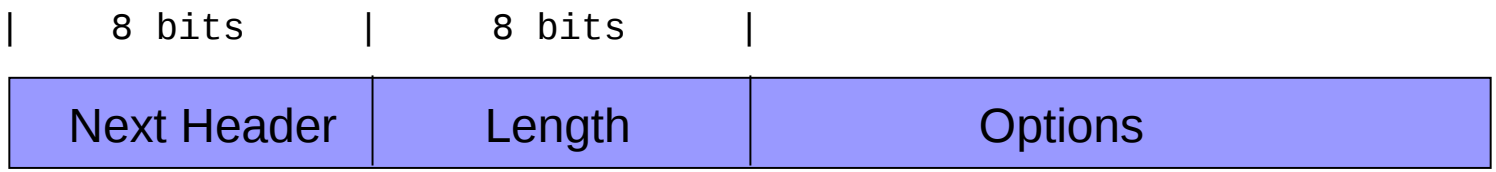


IPv6 Extension Headers

Destination Options

Destination options Header

- Identified by a Next Header of 60.
- Carries options meant for the destination nodes
  - Only some experimental options have been specified
- Should probably be policed at the network edge

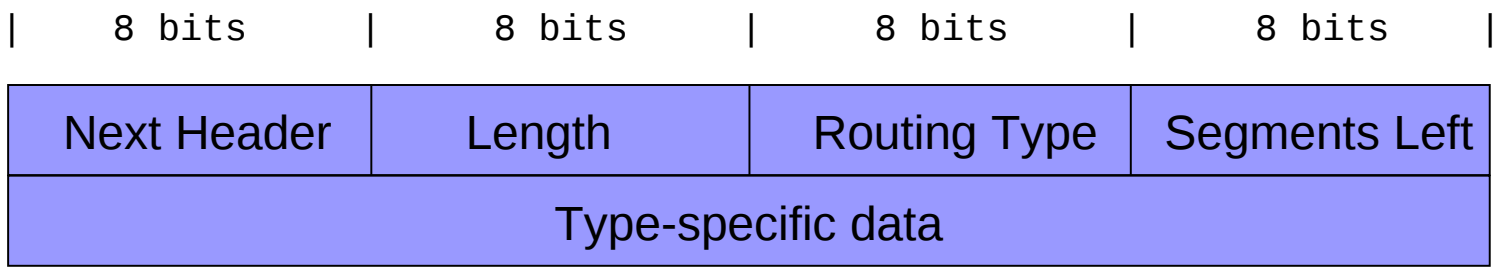


IPv6 Extension Headers

Routing Header

Routing Header

- Identified by a Next Header of 43
- Meant to lists nodes that must be visited on the way to the packet's destination



Routing Type 0

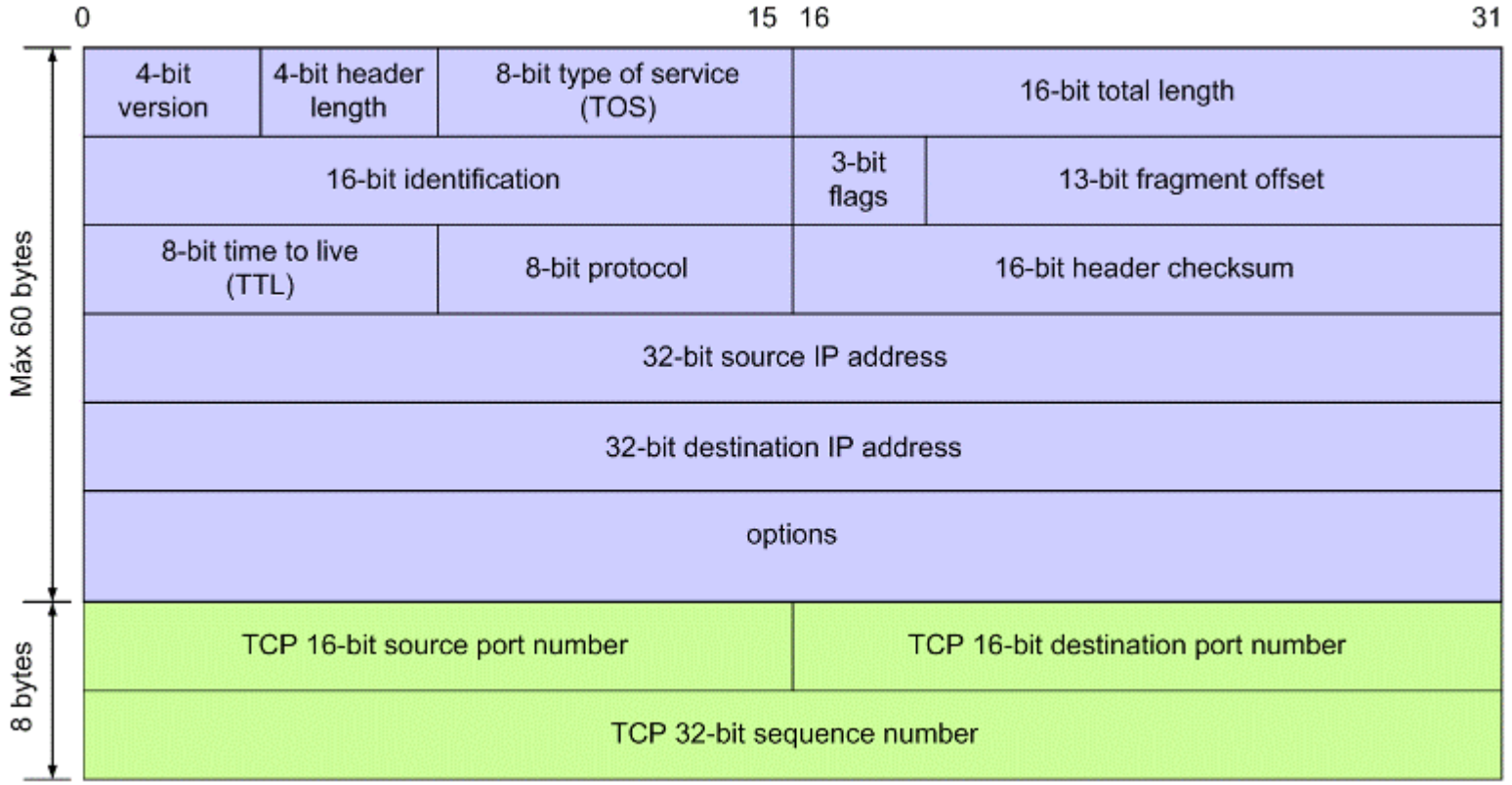
- IPv6 version of IPv4 Source Routing
- Can be far more damaging (many more addresses can be specified)
- Deprecated for current implementations

IPv6 Extension Headers

Implications on Firewalls

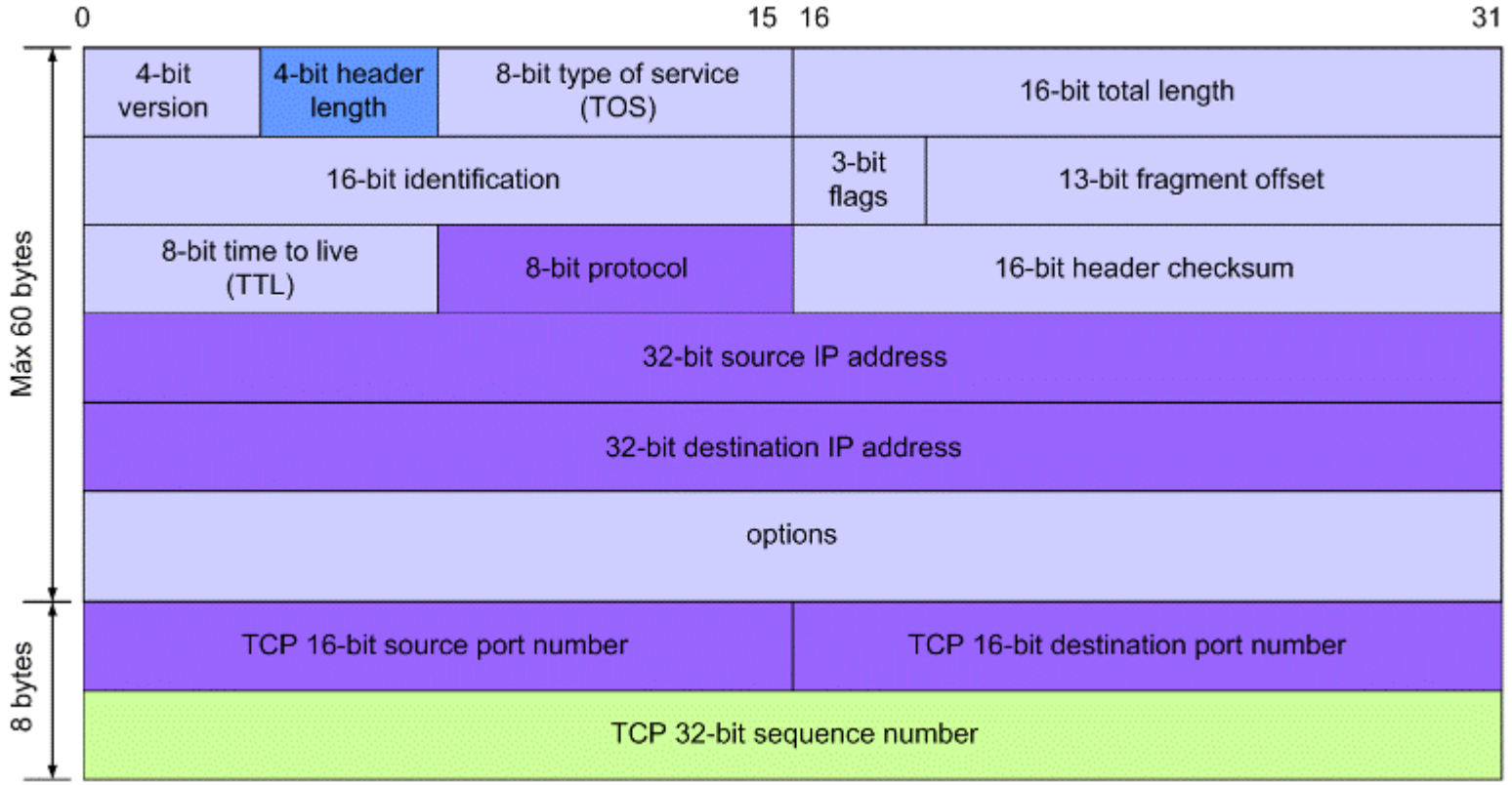
Brief Overview of the IPv4 Situation

- IPv4 has a variable-length (20-60 bytes) header, and a minimum MTU of 68 bytes.



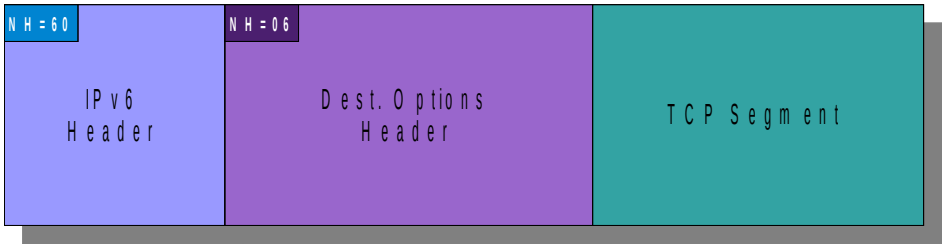
Brief Overview of the IPv4 Situation

- IPv4 has a variable-length (20-60 bytes) header, and a minimum MTU of 68 bytes. The following information can be assumed to be present on every packet:



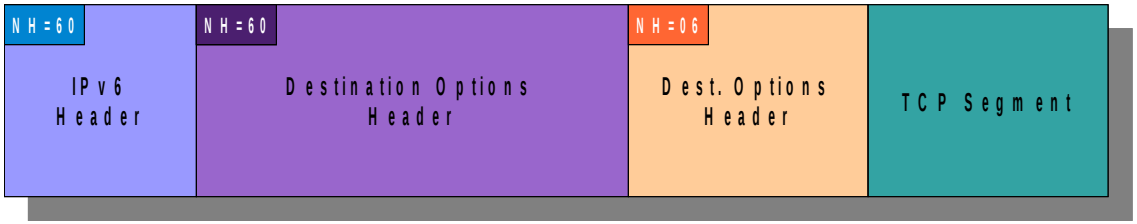
Brief Overview of the IPv6 Situation

- The variable length-header has been replaced by a fixed-length (40 bytes) header
- Any IPv6 options are included in “extension headers” that form a “header chain”
- For example,



Problem Statement

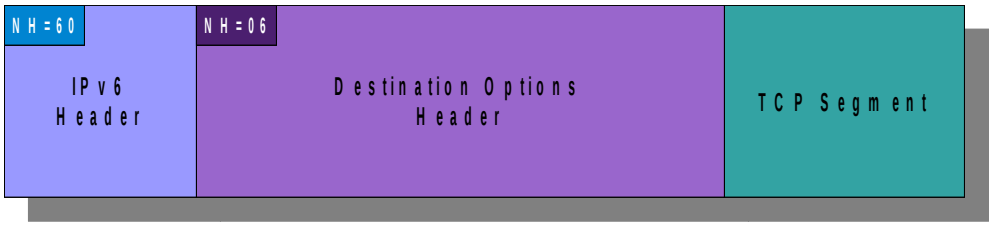
- The specifications allow for the use of multiple extension headers, even of the same type – and implementations support this.
- Thus, the structure of the resulting packet becomes increasingly complex, and packet filtering becomes virtually impossible.
- For example:



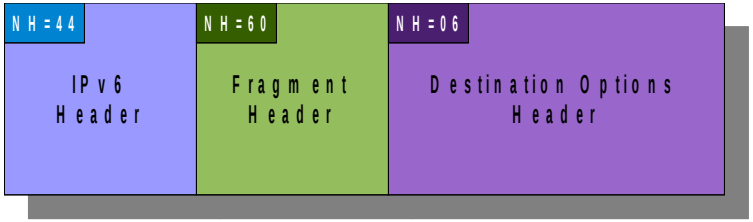
Problem Statement (II)

- Example of Destination Options and Fragmentation:

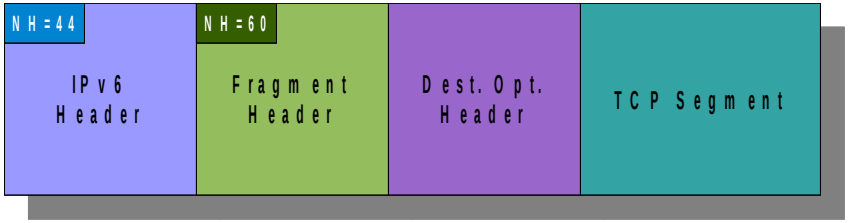
Original Packet



First Fragment

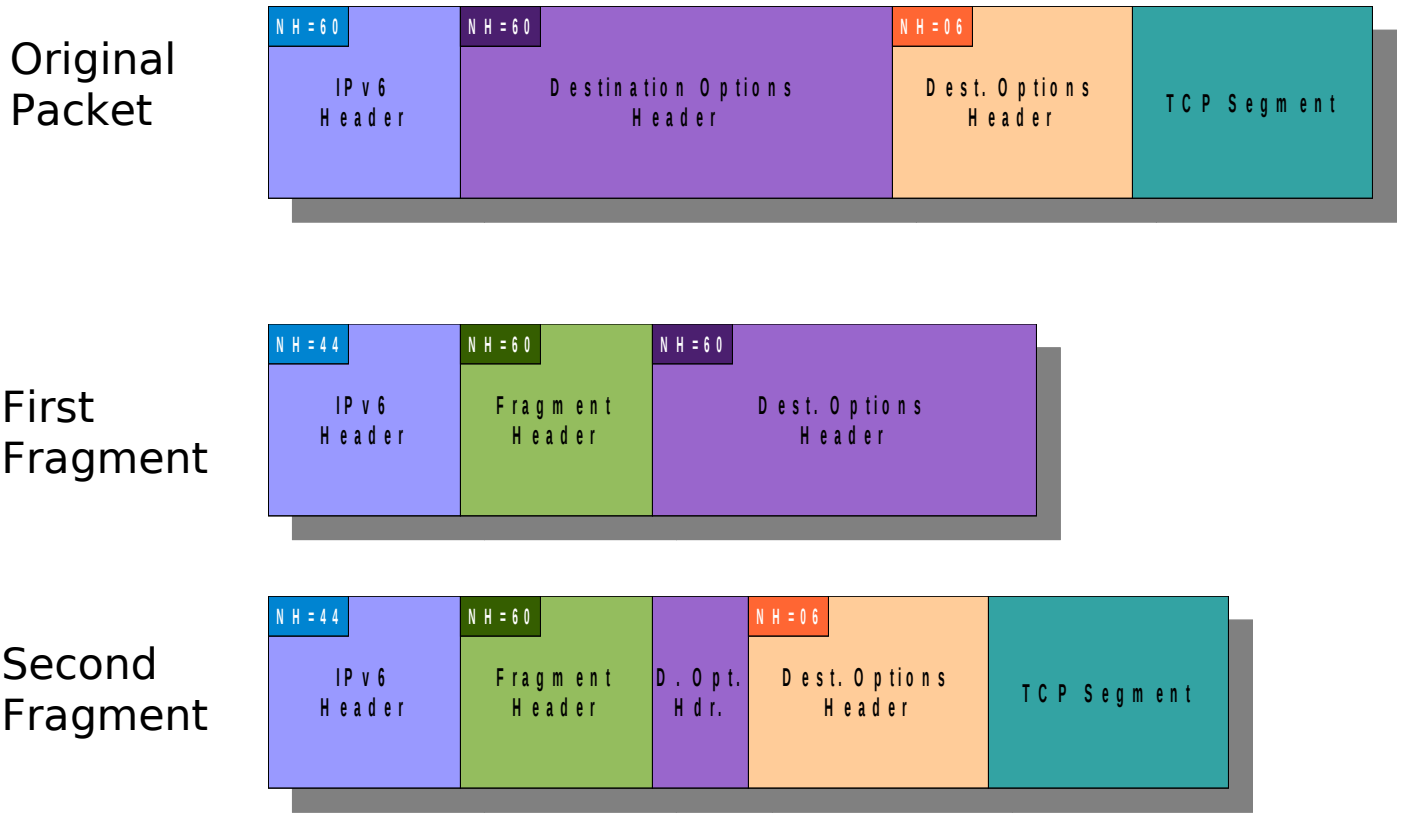


Second Fragment



Problem Statement (III)

- Two Destination Options headers, and a Fragment Header:



Possible Countermeasures

- Use a stateful firewall that reassembles the fragments, and then applies the packet filtering rules
- Filter (in firewalls and/or hosts) packets with specific combinations of extension headers:
  - Packets with multiple extension headers (e.g., more than 5)
  - Packets that combine fragmentation and other extension headers
  - Packets which are fragmented and do not contain the upper-layer header in the first fragment.
- If filtering is to be performed in layer-2 devices (e.g., RA-Guard), the possible counter-measures are reduced
  - e.g., it's not possible to do fragment reasembly at layer-2!

Some Conclusions

- With the current state of affairs, it ay be easy to circumvent IPv6 firewalls.
- We expect firewalls will block (at the very least) packets with specific combinations of extension headers.
- The result will be: less flexibility, possibly preventing any use of IPv6 exntesion headers

Internet Control Message Protocol version 6 (ICMPv6)

Internet Control Message Protocol version 6

- ICMP is a core protocol of the IPv6 suite, and is used for:
- Fault isolation (ICMPv6 errors)
  - Troubleshooting (ICMPv6 echo request/response)
  - Address Resolution
  - Stateless address autoconfiguration
- ICMPv6 is mandatory for IPv6 operation

ICMPv6

Error Messages

Fault Isolation (ICMPv6 error messages)

- A number of ICMPv6 error messages are specified in RFC 4443:
  - Destination Unreachable
    - No route to destination
    - Beyond scope of source address
    - Port Unreachable, etc.
  - Packet Too Big
  - Time Exceeded
    - Hop Limit Exceeded in Transit
    - Fragment reassembly time exceeded
  - Parameter Problem
    - Erroneous header field encountered
    - Unrecognized Nect Header type encountered
    - Unrecognized IPv6 option encountered
  - ICMP Redirect
- Clearly, most of them parallel their ICMP counter-parts

Hop Limit Exceeded in Transit

- Are generated when the Hop Limit of a packet is decremented to 0.
- Typically leveraged by traceroute tool
- Example:

```
% traceroute 2004:1::30c:29ff:feaf:1958
traceroute to 2004:1::30c:29ff:feaf:1958 (2004:1::30c:29ff:feaf:1958) from
2004::5e26:aff:fe33:7063, port 33434, from port 60132, 30 hops max, 60 byte
packets
 1  2004::1  0.558 ms  0.439 ms  0.500 ms
 2  2004::1  2994.875 ms !H  3000.375 ms !H  2997.784 ms !H
```

Hop Limit Exceeded in Transit (II)

- Tcpdump trace:

1. IP6 (**hlim 1**, next-header UDP (17) payload length: 20) 2004::5e26:aff:fe33:7063.60132 > 2004:1::30c:29ff:feaf:1958.33435: [udp sum ok] UDP, length 12
2. IP6 (hlim 64, next-header ICMPv6 (58) payload length: 68) 2004::1 > 2004::5e26:aff:fe33:7063: [icmp6 sum ok] **ICMP6, time exceeded in-transit**, length 68 for 2004:1::30c:29ff:feaf:1958
3. IP6 (**hlim 2**, next-header UDP (17) payload length: 20) 2004::5e26:aff:fe33:7063.60132 > 2004:1::30c:29ff:feaf:1958.33436: [udp sum ok] UDP, length 12
4. IP6 (hlim 64, next-header ICMPv6 (58) payload length: 68) 2004::1 > 2004::5e26:aff:fe33:7063: [icmp6 sum ok] **ICMP6, destination unreachable**, length 68, unreachable address 2004:1::30c:29ff:feaf:1958

Hop Limit Exceeded in Transit (III)

- Use of traceroute6 for network reconnaissance could be mitigated by:
  - filtering outgoing “Hop Limit Exceeded in transit” at the network perimeter, or,
  - by normalizing the “Hop Limit” of incoming packets at the network perimeter
- Note: NEVER normalize the “Hop Limit” to 255 (or other large value) –use “64” instead

ICMPv6 Connection-Reset Attacks

- Some ICMPv6 messages are assumed to indicate “hard errors”
- Some stacks used to abort TCP connections when hard errors were received
- No stacks were found vulnerable to these attacks
- We learned the lesson from IPv4 – good!

ICMPv6 PMTUD Attacks

- ICMPv6 PTB messages are used for Path-MTU discovery
- The security implications of these messages are well-known (remember “ICMP attacks against TCP” back in 2004?)
- The mitigations are straightforward:
  - Validate the received ICMPv6 messages (TCP SEQ #, etc.)
- Many implementations fail to properly validate ICMPv6 messages
  - The Path-MTU is never reduced to less than 1280 bytes
  - But a Fragment Header will be included in all further packets
    - This can be leveraged for exploiting fragmentation-relaetd attacks

ICMPv6 Redirects

- ICMP redirects are very similar to the ICMP counterpart, except for:
  - The Hop Limit is required to be 255 – this reduces exposure.
  - There are no “network redirects”
- ICMPv6 redirects are an optimization – hence they can be disabled with no interoperability implications
- Most stacks enable them by default
- In *BSDs, ICMPv6 Redirect processing is controlled with the sysctl `net.inet6.icmp6.rediraccept`.

ICMPv6
Informational Messages

ICMPv6 Informational

- Echo Request/Echo response:
  - Used to test node reachability (“ping6”)
  - Widely supported, although disabled by default in some OSes
- Node Information Query/Response
  - Specified by RFC 4620 as “Experimental”, but supported (and enabled by default) in KAME.
  - Not supported in other stacks
  - Used to obtain node names or addresses.

ICMPv6 Echo Request/Echo response

- Used for the “ping6” tool, for troubleshooting
- Also usually exploited for network reconnaissance
- Some implementations ignore incoming ICMPv6 “echo requests”
- Example:

```
% ping6 2004::1
PING 2004::1(2004::1) 56 data bytes
64 bytes from 2004::1: icmp_seq=1 ttl=64 time=28.4 ms

--- 2004::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 28.460/28.460/28.460/0.000 ms
```

tcpdump output

```
1. IP6 2004::5e26:aff:fe33:7063 > 2004::1: ICMP6, echo request, seq 1,
   length 64
2. IP6 2004::1 > 2004::5e26:aff:fe33:7063: ICMP6, echo reply, seq 1,
   length 64
```

sysctl’s for ICMPv6 Echo Request

- No sysctl’s in BSD’s or Linux
- ICMPv6 Echo requests can nevertheless be filtered in firewalls
- Might want to filter ICMPv6 Echo Requests in hosts (but not in routers)

Node Information Query/Response

- Specified in RFC 4620 as “Experimental”, but included (and enabled by default) in KAME
- Allows nodes to request certain network information about a node in a server-less environment
  - Queries are sent with a target name or address (IPv4 or IPv6)
  - Queried information may include: node name, IPv4 addresses, or IPv6 addresses
- Node Information Queries can be sent with the ping6 command (“-w” and “-b” options)

Node Information Query/Response (II)

- Response to Node Information Queries is controlled by the sysctl `net.inet6.icmp6.nodeinfo`:
  - 0: Do not respond to Node Information queries
  - 1: Respond to FQDN queries (e.g., “ping6 –w”)
  - 2: Respond to node addresses queries (e.g., “ping6 –a”)
  - 3: Respond to all queries
- `net.inet6.icmp6.nodeinfo` defaults to 1 in OpenBSD, and to 3 in FreeBSD.
- My take: unless you really need your nodes to support Node Information messages, disable it (i.e., “sysctl –w net.inet6.icmp6.nodeinfo=0).

Some examples with ICMPv6 NI (I)

- Query node names

```
$ ping6 -w ff02::1%vic0

PING6(72=40+8+24 bytes) fe80::20c:29ff:feaf:194e%vic0 --> ff02::1%vic0
41 bytes from fe80::20c:29ff:feaf:194e%vic0: openbsd46.my.domain.
30 bytes from fe80::20c:29ff:fe49:ebdd%vic0: freebsd
41 bytes from fe80::20c:29ff:feaf:194e%vic0: openbsd46.my.domain.
30 bytes from fe80::20c:29ff:fe49:ebdd%vic0: freebsd
41 bytes from fe80::20c:29ff:feaf:194e%vic0: openbsd46.my.domain.
30 bytes from fe80::20c:29ff:fe49:ebdd%vic0: freebsd
--- ff02::1%vic0 ping6 statistics ---
3 packets transmitted, 3 packets received, +3 duplicates, 0.0% packet loss
```

Some examples with ICMPv6 NI (II)

- Query addresses

```
$ ping6 -a Aacgls ff02::1%vic0

PING6(72=40+8+24 bytes) fe80::20c:29ff:feaf:194e%vic0 --> ff02::1%vic0
76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)  fe80::1(TTL=infty)

76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)  fe80::1(TTL=infty)

76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)
  fe80::1(TTL=infty)

--- ff02::1%vic0 ping6 statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
```

Some examples with ICMPv6 NI (III)

- Use the NI multicast group

```
$ ping6 -I vic0 -a Aacgls -N freebsd

PING6(72=40+8+24 bytes) fe80::20c:29ff:feaf:194e%vic0 --> ff02::1%vic0
76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)  fe80::1(TTL=infty)

76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)  fe80::1(TTL=infty)

76 bytes from fe80::20c:29ff:fe49:ebdd%vic0:
  fe80::20c:29ff:fe49:ebdd(TTL=infty)
  ::1(TTL=infty)
  fe80::1(TTL=infty)

--- ff02::1%vic0 ping6 statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
```

Neighbor Discovery for IPv6

Neighbor Discovery for IPv6

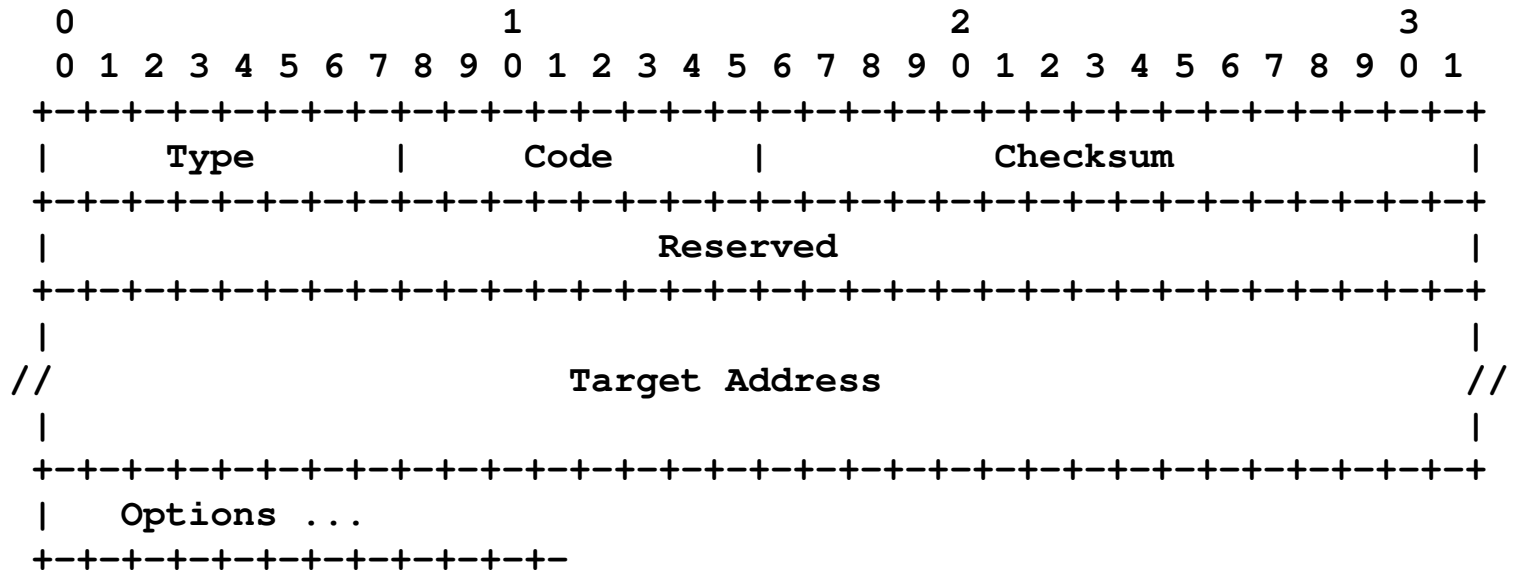
Address Resolution

Address Resolution in IPv6

- Employs ICMPv6 Neighbor Solicitation and Neighbor Advertisement
- It (roughly) works as follows:
  1. Host A sends a NS: Who has IPv6 address fc01::1?
  2. Host B responds with a NA: I have IPv6 address, and the corresponding MAC address is 06:09:12:cf:db:55.
  3. Host A caches the received information in a “Neighbor Cache” for some period of time (this is similar to IPv4’s ARP cache)
  4. Host A can now send packets to Host B

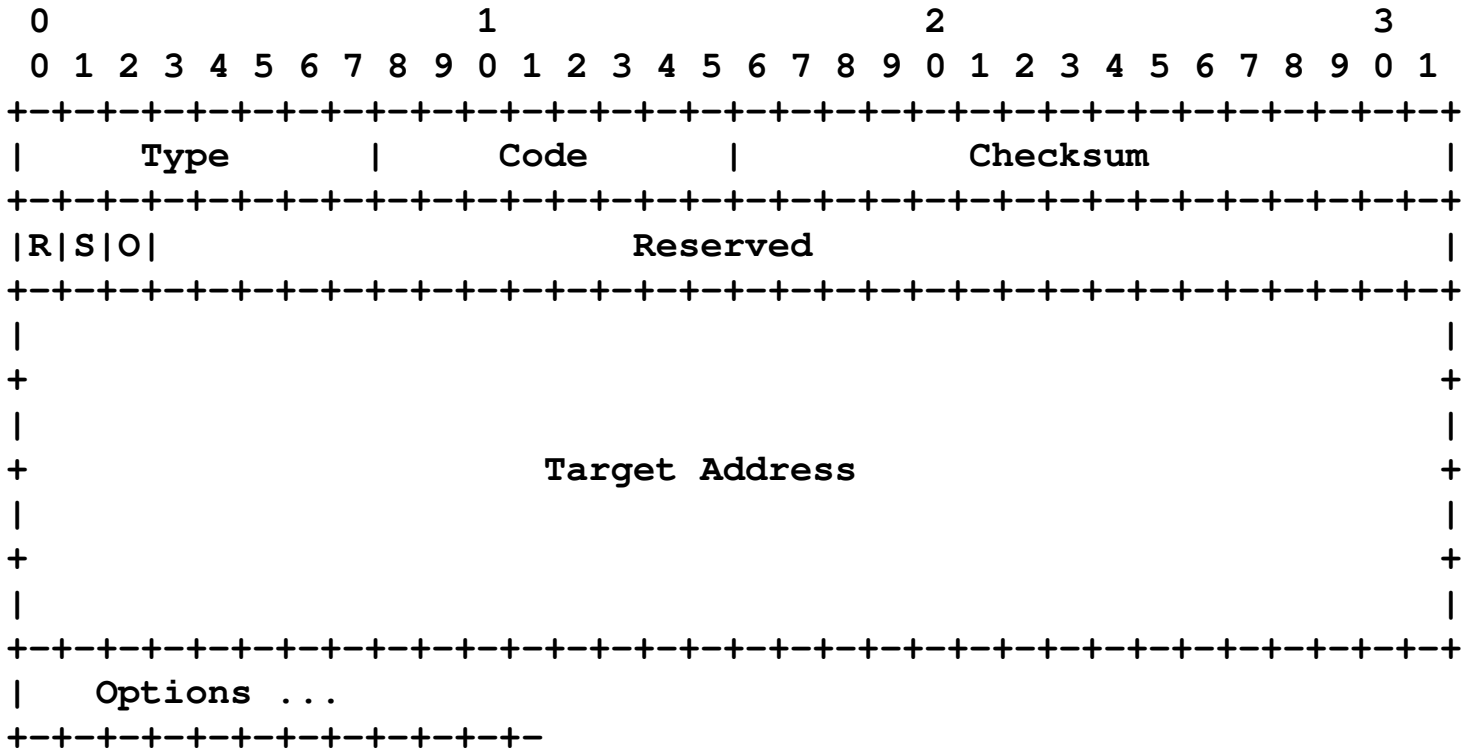
Neighbor Solicitation Messages

- ICMPv6 messages of Type 135, Code 0
- Used to solicit the mapping of an IPv6 address to a link-layer address
- Only allowed option so far: “Source Link-layer address”



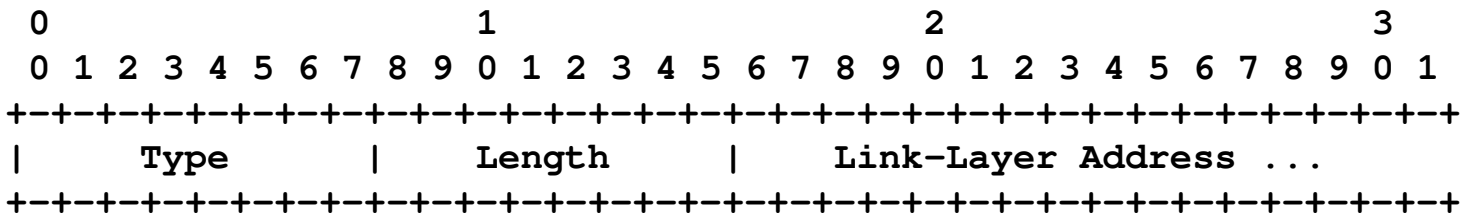
Neighbor Advertisement Messages

- ICMPv6 messages of Typo 136, Code 0
- Use to informa the maping of a IPv6 address to a link-layer address
- Only allowed option so far: “Target Link-layer address”



Source/Target Link-layer Address Options

- The Source Link-layer Address contains the link-layer address corresponding to the “Source Address” of the packet
- The Target Link-layer address contains the link-layer address correspondign to the “Target Address” of the Neighbor Solicitation message.



Type: 1 for Source Link-layer Address
2 for Target Link-layer Address

Sample Address Resolution Traffic

```
% ping6 2004::1

12:12:42.086657 2004::20c:29ff:fe49:ebdd > ff02::1:ff00:1: icmp6: neighbor
sol: who has 2004::1(src lladdr: 00:0c:29:49:eb:dd) (len 32, hlim 255)
12:12:42.087654 2004::1 > 2004::20c:29ff:fe49:ebdd: icmp6: neighbor adv:
tgt is 2004::1(RSO)(tgt lladdr: 00:0c:29:c0:97:ae) (len 32, hlim 255)
12:12:42.089147 2004::20c:29ff:fe49:ebdd > 2004::1: icmp6: echo request
(len 16, hlim 64)
12:12:42.089415 2004::1 > 2004::20c:29ff:fe49:ebdd: icmp6: echo reply (len
16, hlim 64)
```

ndisc6: Neighbor Discovery diagnostic tool

- Can be used to send NS for a particular address
- Example:

```
$ /bin/rdisc6 vboxnet0
Soliciting ff02::2 (ff02::2) on vboxnet0...

Hop limit                 :           64 (      0x40)
Stateful address conf.    :           No
Stateful other conf.      :           No
Router preference         :       medium
Router lifetime           :         1800 (0x00000708) seconds
Reachable time            :  unspecified (0x00000000)
Retransmit time           :  unspecified (0x00000000)
 Source link-layer address: 08:00:27:F9:73:04
 Prefix                   : 2000:1::/64
  Valid time              :      2592000 (0x00278d00) seconds
  Pref. time              :       604800 (0x00093a80) seconds
 from fe80::a00:27ff:fef9:7304
```

Neighbor Cache

- Stores information learned from the Address Resolution mechanism
- Each entry (IPv6 address, link-layer address) can be in one of the following states:

NC entry state	Semantics
INCOMPLETE	Add. Res. Is in progress (not yet determined)
REACHABLE	Neighbor is reachable
STALE	Not known to be reachable
DELAY	Not known to be reachable (wait for indication)
PROBE	Not known to be reachble (probes being sent)

Neighbor Cache (contents in *BSD)

- Sample output of “`ndp –a`” (BSDs):

```
% ndp -a
Neighbor                        Linklayer Address  Netif Expire    S Flags
2004:1::f8dd:347d:8fd8:1d2c     0:c:29:49:eb:e7      em1 permanent R
fe80::20c:29ff:fec0:97b8%em1    0:c:29:c0:97:b8      em1 23h48m16s S R
2004:1::20c:29ff:fe49:ebe7      0:c:29:49:eb:e7      em1 permanent R
fe80::20c:29ff:fe49:ebe7%em1    0:c:29:49:eb:e7      em1 permanent R
2004::1                         0:c:29:c0:97:ae      em0 23h49m27s S R
2004::20c:29ff:fe49:ebdd        0:c:29:49:eb:dd      em0 permanent R
fe80::20c:29ff:fe49:ebdd%em0    0:c:29:49:eb:dd      em0 permanent R
fe80::20c:29ff:fec0:97ae%em0    0:c:29:c0:97:ae      em0 23h48m16s S R
2004::d13e:2428:bae7:5605       0:c:29:49:eb:dd      em0 permanent R
```

Neighbor Cache (contents in Linux)

- Sample output of “ip -6 neigh show” (Linux):

```
$ ip -6 neigh show
fe80::a00:27ff:fef9:7304 dev vboxnet0 lladdr 08:00:27:f9:73:04 router STALE
2000::4000 dev vboxnet0 lladdr 11:22:33:44:55:66 PERMANENT
2000:1::1 dev vboxnet0 lladdr 08:00:27:f9:73:04 router REACHABLE
fe80::fc8d:15ed:7f43:68ea dev wlan0 lladdr 00:21:5c:0b:5d:61 router STALE
```

Address Resolution

some attacks...

“Man in the Middle” or Denial of Service

- They are the IPv6 version of IPv4’s ARP cache poisoning
- Without proper authentication mechanisms in place, its trivial for an attacker to forge Neighbor Discovery messages
- Attack:
  - “Listen” to incoming Neighbor Solicitation messages, with the victim’s IPv6 address in the “Target Address” field
  - When a NS is received, respond with a forged Neighbor Advertisement
- If the “Target Link-layer address” corresponds to a non-existing node, traffic is dropped, resulting in a DoS.
- If the “Target Link-layer address” is that of the attacker, he can perform a “man in the middle” attack.

Sniffing in a switched network

- Rather than trying to overflow the switch table, a more elegant attack can be performed-
- Map the target addresses to either:
  - The broadcast Ethernet address (ff:ff:ff:ff:ff:ff)
  - Multicast Ethernet addresses (e.g., 33:33:00:00:01)
- This will cause traffic to be sent to all nodes (including the attacker and the legitimate recipient)
- All BSD variants tested don’t check for these special addresses!

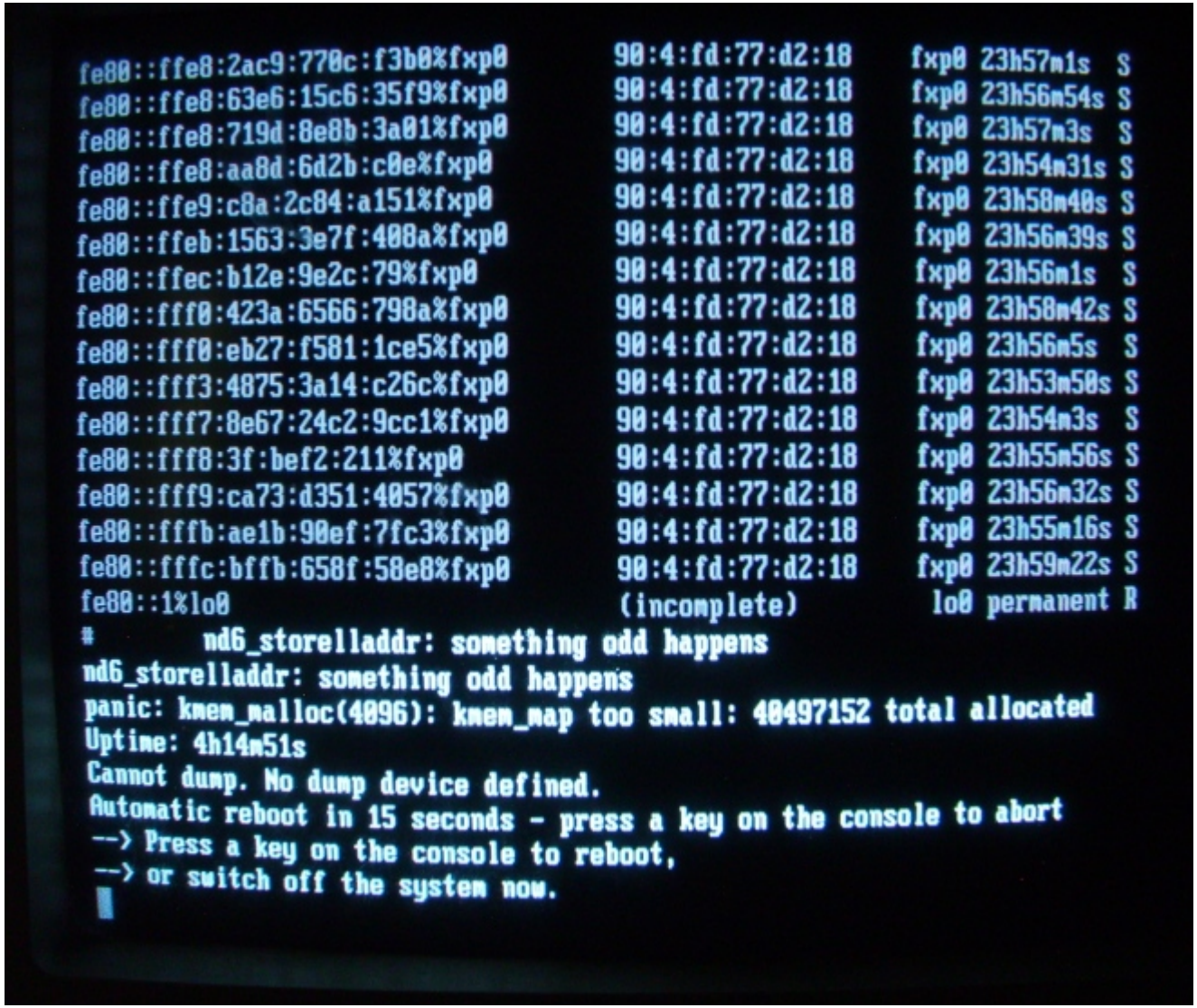
Introduce a forwarding loop at a router

- Respond the Neighbor solicitation sent by a router
- The router will receive a copy of the packet it sends (assuming the NIC allows this)
- The Hop Limit of the packet will be decremented, and the packet will be resent
- The process will be repeated until the the Hop Limit is decremented to 0.

Overflowing the Neighbor Cache

- Some implementations (e.g., FreeBSD and NetBSD) don’t enforce limits on the number of entries that can be created in the Neighbor Cache
- All kernel memory can be tied for the Neighbor Cache, leading to a system panic.
- Attack:
  - Send a large number of Neighbor Solicitation messages with a Source Link-layer address
  - For each received packet, the victim host creates an entry in the neighbor Cache
  - And if entries are added at a faster rate than “old entries” are pruned from the Neighbor Cache....

Overflowing the Neighbor Cache (II)



Some sysctl's for Neighbor Discovery (OpenBSD)

- `net.inet6.ip6.neighborgcthresh` (defaults to 2048): Maximum number of entries in the Neighbor Cache
- `net.inet6.icmp6.nd6_prune` (defaults to 1): Interval between Neighbor Cache babysitting (in seconds).
- `net.inet6.icmp6.nd6_delay` (defaults to 5): specifies the DELAY_FIRST_PROBE_TIME constant from RFC 4861.
- `net.inet6.icmp6.nd6_umaxtries` (defaults to 3): specifies the MAX_UNICAST_SOLICIT constant from RFC 4861
- `net.inet6.icmp6.nd6_mmaxtries` (defaults to 3): specifies the MAX_MULTICAST_SOLICIT constant from RFC 4861.
- `net.inet6.icmp6.nd6_useloopback` (defaults to 1): If non-zero, uses the loopback interface for local traffic.
- `net.inet6.icmp6.nd6_maxnudhint` (defaults to 0): Maximum number of upper-layer reachability hints before normal ND is performed.

Address Resolution

countermeasures

Secure Neighbor Discovery (SeND)

- SeND a cryptographic approach to the problem of forged Neighbor Solicitation messages
  - Certification paths certify the authority of routers
  - Cryptographically-Generated Addresses (CGA) bind IPv6 addresses to an assymetric key pair
  - RSA signatures protect all Neighbor Discovery messages
- However, SeND is hard to deploy:
  - Not widely supported
  - The requirement of a PKI is a key obstacle for its deployment

Neighbor Discovery traffic monitoring

- Some tools keep (e.g., NDPMon) record of the legitimate mappings (IPv6 -> Ethernet), and sound an alarm if the mapping changes
- This is similar to arpwatch in IPv4
- However, these tools can be trivially evaded:
  - ND runs on top of IPv6
  - Packets may contain IPv6 Extension Headers
  - Packets may be fragmented
  - And since traffic occurs in the local network, there is no "man in the middle" to reassemble the packets or "normalize" them

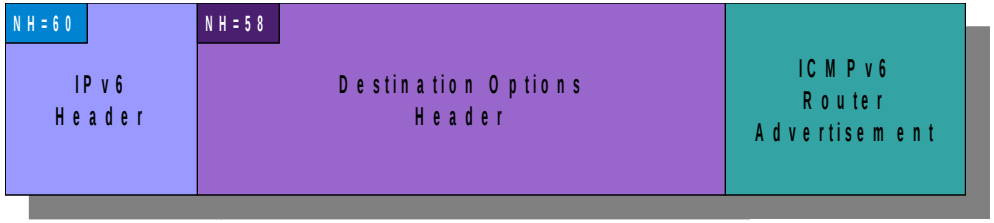
Neighbor Discovery traffic monitoring (II)

- An arbitrary number of Extension headers can be inserted to make traffic monitoring harder
- The monitor tool would need to follow the entire header chain to "spot" the Neighbor Discovery messages.

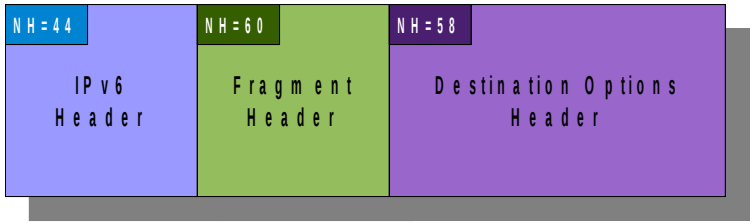
Neighbor Discovery traffic monitoring (III)

- Combination of a Destination Options Header and fragmentation:

Original Packet

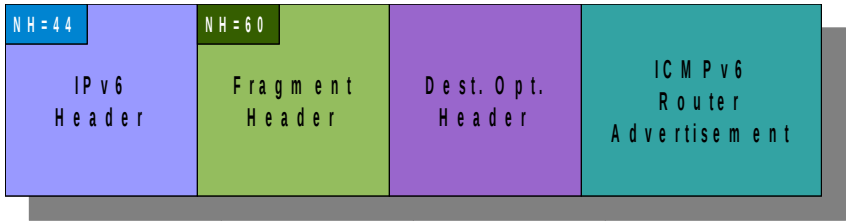


First Fragment



Can only tell there's ICMv6 inside

Second Fragment

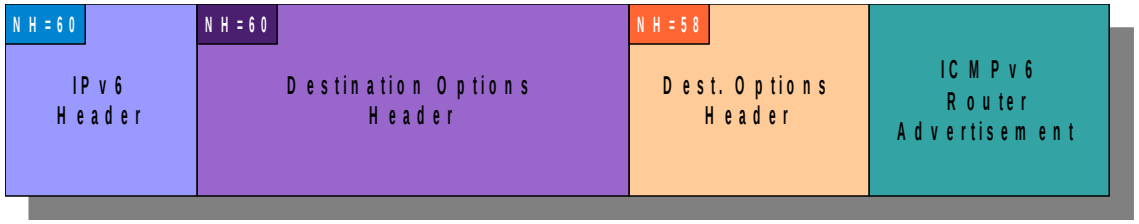


Can only tell there's Dest. Opt. Hdr inside!

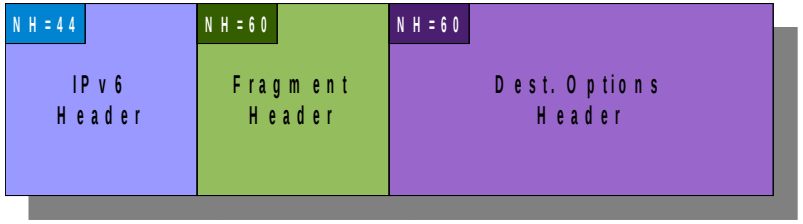
Neighbor Discovery traffic monitoring (IV)

- Two Destination Options headers, and fragmentation:

Original Packet

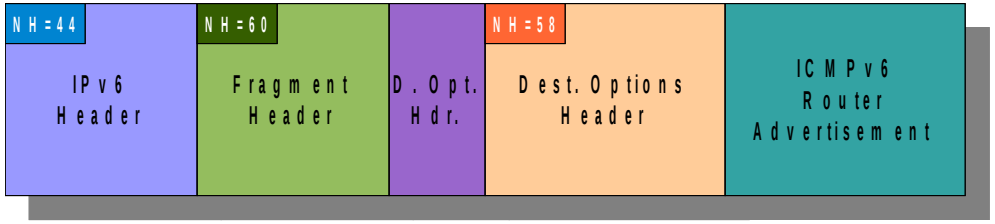


First Fragment



Can only tell there's Dest. Opt. Hdr inside!

Second Fragment



Can only tell there's Dest. Opt. Hdr inside!

Restricting access to the local network

- Neighbor Discovery traffic is limited to the local network
- Separation of systems in different networks limits the damage an attacker can cause
- This is not always possible, but still desirable

Static Neighbor Cache entries

- Static entries can be including in the Neighbor Cache
- This is similar to static entries in the ARP Cache en IPv4
- If a static NC entry is present for an IPv6, the host need not employ Neighbor Discovery
  - Beware that some implementations used to remain vulnerable to ND attacks anyway!

Static Neighbor Cache entries in BSDs

- The Neighbor Cache is manipulated with the "ndp" command
- Static entries are added as follows:

  # ndp –s IPV6ADDR MACADDR

- If IPV6ADDR is a link-local address, an interface index is specified as follows:

  # ndp –s IPV6ADDR%IFACE MACADDR

Static Neighbor Cache entries in Linux

- The Neighbor Cache is manipulated with the "ip" command.
- Static entries are added as follows:

sudo ip neigh add to IPV6ADDR lladdr MACADDR dev IFACE nud permanent

- Verify the results with:

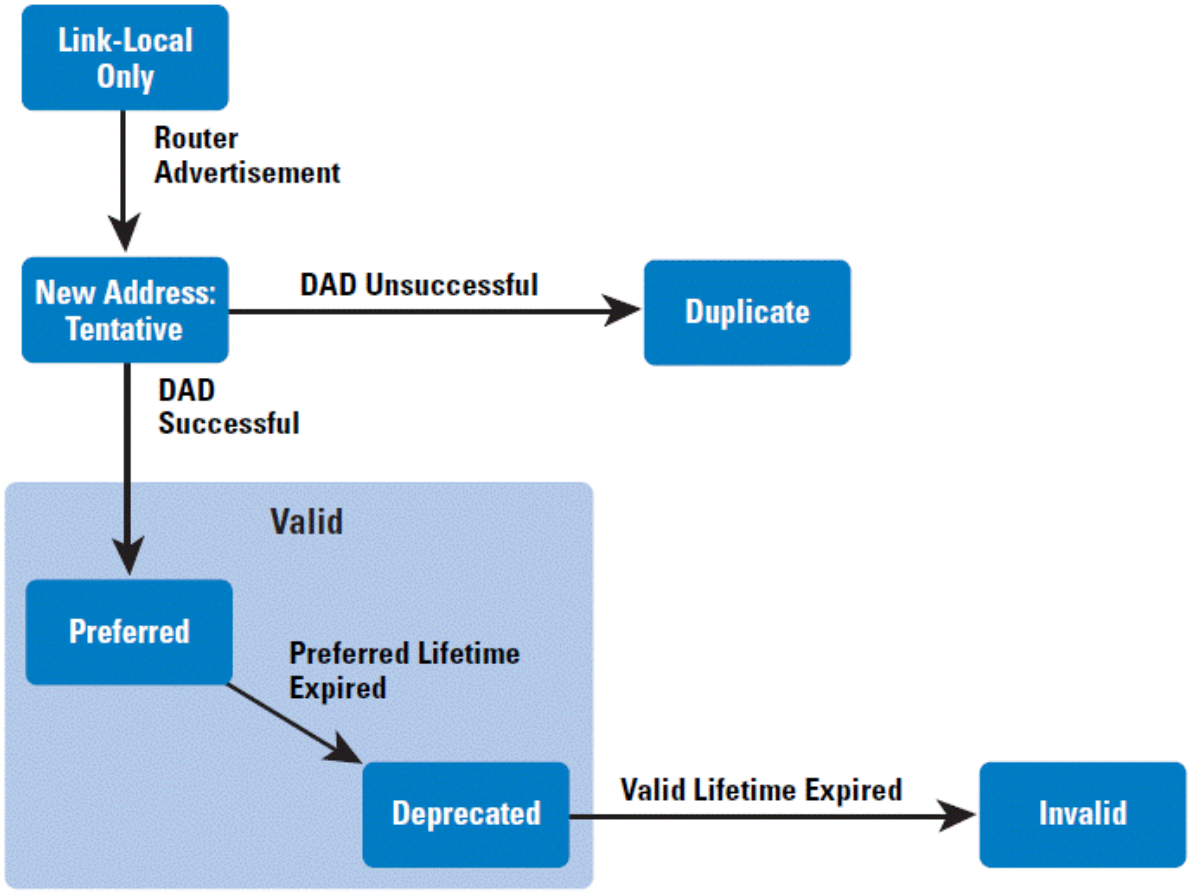
ip -6 neigh show

IPv6 Stateless Address Autoconfiguration (SLAAC)

Stateless Address Autoconfiguration

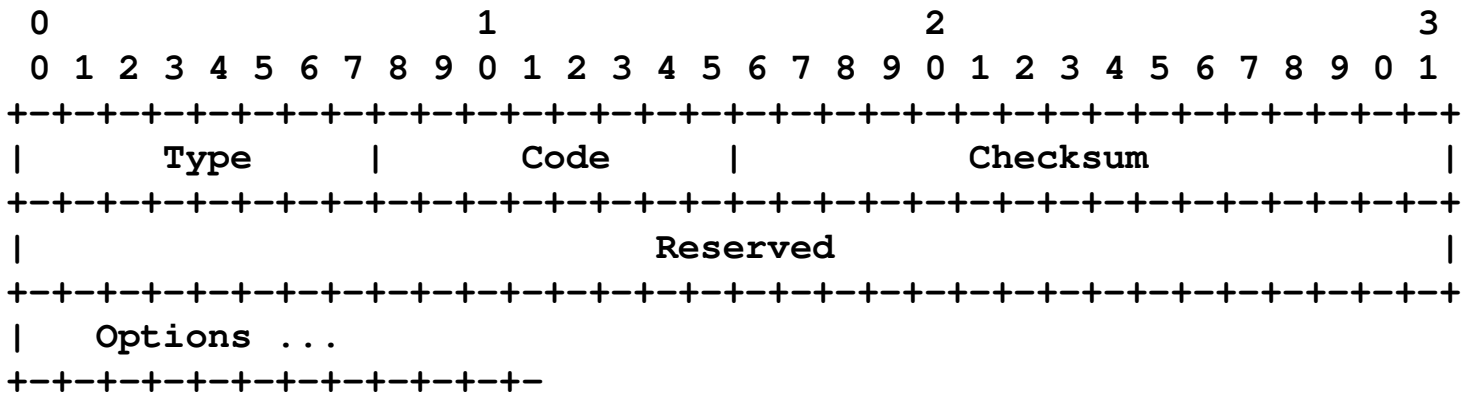
- It works (roughly) as follows:
  1. The host configures a link-local address
  2. It checks that the address is unique – i.e., it performs Duplicate Address Detection (DAD) for that address
     - Sends a NS, and waits for any answers
  1. The host sends a Router Solicitation message
  2. When a Router Advertisement is received, it configures a “tentative” IPv6 address
  3. It checks that the address is unique – i.e., it performs Duplicate Address Detection (DAD) for that address
     - Sends a NS, and waits for any answers
  1. If the address is unique, it typically becomes a “preferred” address

Address Autoconfiguration flowchart



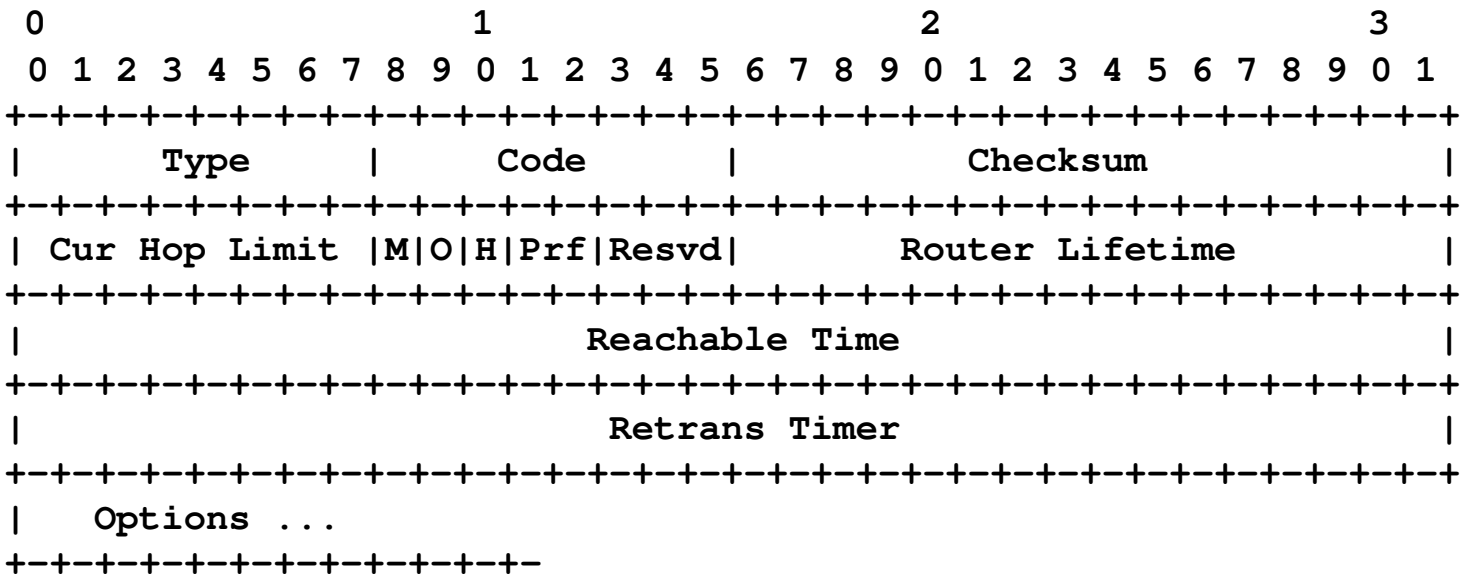
Router Solicitation Messages

- ICMPv6 messages of Type 133, Code 0
- Used to solicit network configuration information to local routers
- Only allowed option so far: Source Link-layer Address



Router Advertisement Messages

- ICMPv6 messages of Type 134, Code 0
- Used to announce network configuration information to local hosts

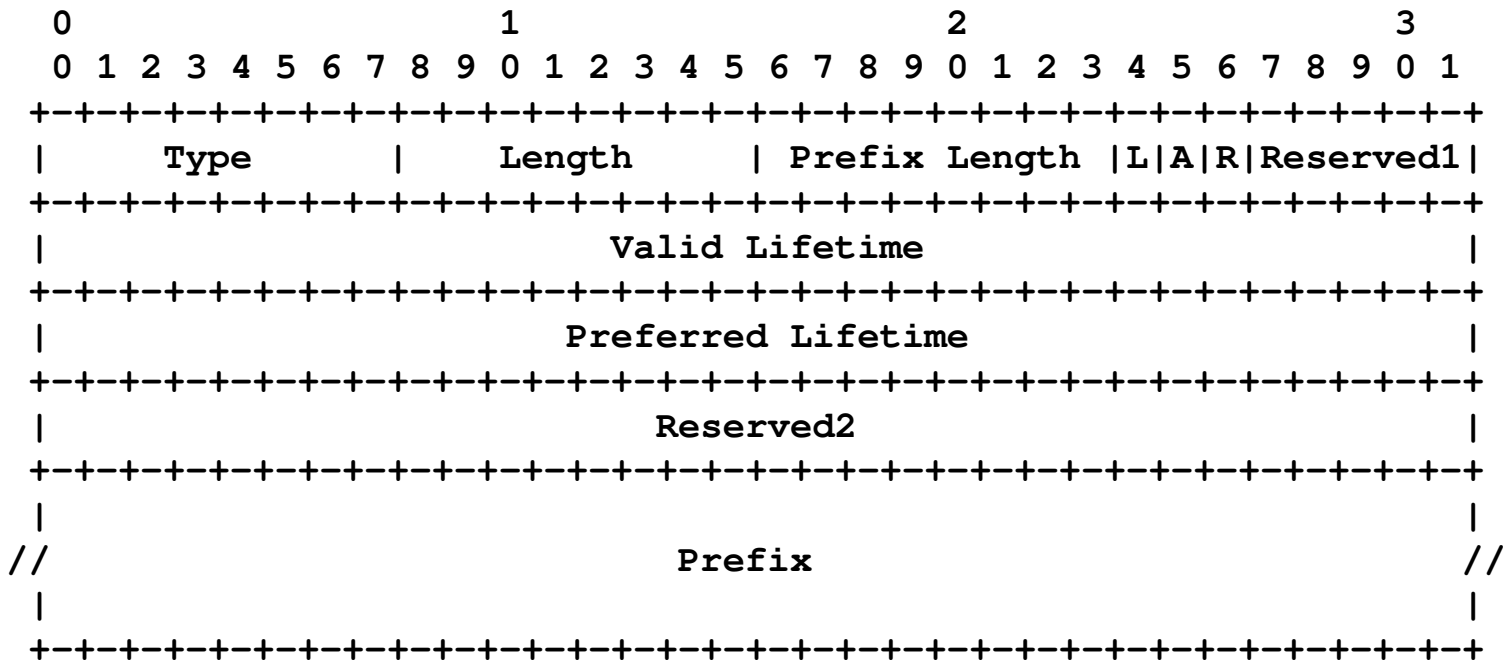


Possible Options in RA messages

- ICMPv6 Router Advertisements may contain the following options:
  - Source Link-layer address
  - Prefix Information
  - MTU
  - Route Information
  - Recursive DNS Server
- Usually, they include many of them

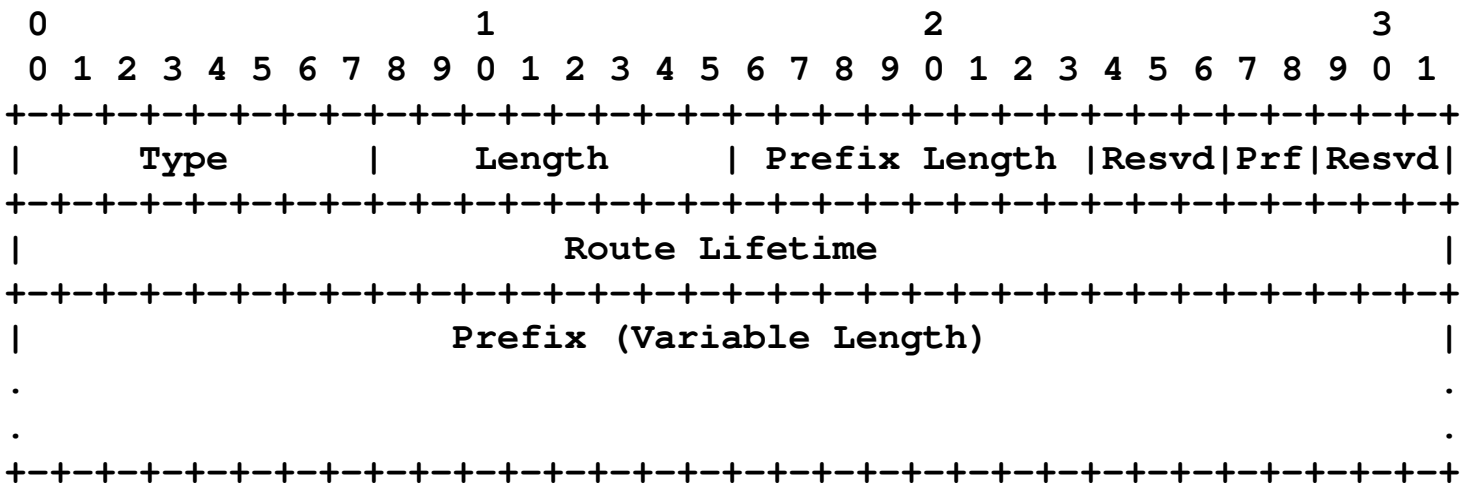
Prefix Information Option

- Identified by a Type of 3
- Specifies “on-link” and “auto-configuration” prefixes



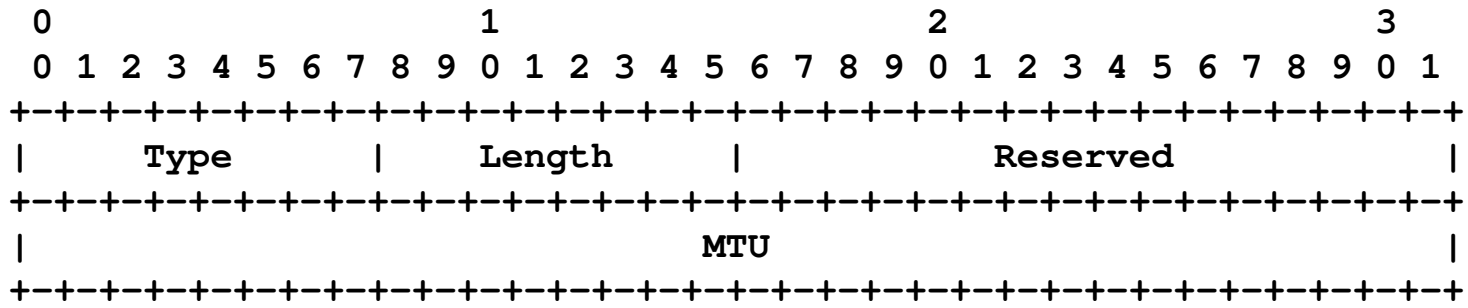
Router Information Option

- Identified by a Type of 24
- Advertises specific routes, with different priorities



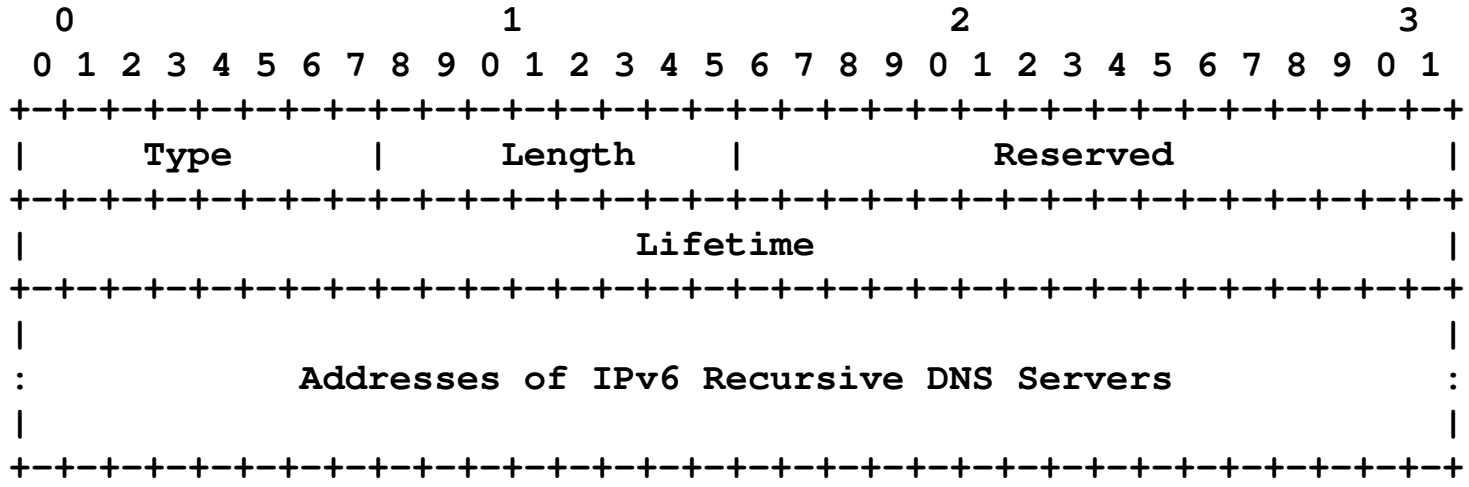
MTU Option

- Identified by a Type of 5
- Specifies the MTU to be used for this link



RDNSS Option

- Identified by a Type of 24
- Used to advertise recursive DNS servers



Sample Configuration

- Sample output of "`ifconfig –a`" (BSDs):

```
# ifconfig -a
em0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    options=9b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM>
    ether 00:0c:29:49:eb:dd
    inet 10.0.0.42 netmask 0xffffff00 broadcast 10.0.0.255
    inet6 fe80::20c:29ff:fe49:ebdd%em0 prefixlen 64 scopeid 0x1
    inet6 2004::20c:29ff:fe49:ebdd prefixlen 64 autoconf
    inet6 2004::d13e:2428:bae7:5605 prefixlen 64 autoconf temporary
    nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
    media: Ethernet autoselect (1000baseT <full-duplex>)
    status: active
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
    options=3<RXCSUM,TXCSUM>
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
```

Sample Configuration

- Sample output of “netstat –r –p ip6” (BSDs):

```
# netstat –r –p ip6
Internet6:
Destination        Gateway            Flags      Netif Expire
::                 localhost          UGRS        lo0 =>
default            fe80::20c:29ff:fec UG          em1
localhost          localhost          UH          lo0
::ffff:0.0.0.0     localhost          UGRS        lo0
2004:1::           link#2             U           em1
2004:1::20c:29ff:f link#2             UHS         lo0
2004:1::f8dd:347d: link#2             UHS         lo0
fe80::             localhost          UGRS        lo0
fe80::%em1         link#2             U           em1
fe80::20c:29ff:fe4 link#2             UHS         lo0
fe80::%lo0         link#5             U           lo0
fe80::1%lo0        link#5             UHS         lo0
ff01:1::           fe80::20c:29ff:fe4 U           em0
ff01:2::           fe80::20c:29ff:fe4 U           em1
ff01:5::           localhost          U           lo0
ff02::             localhost          UGRS        lo0
ff02::%em1         fe80::20c:29ff:fe4 U           em1
ff02::%lo0         localhost          U           lo0
```

Neighbor Cache (prefixes in *BSD)

- Sample output of "`ndp -p`" (BSDs):

```
% ndp -p
2004::/64 if=em0
flags=LAO vltime=2592000, pltime=604800, expire=29d23h57m4s, ref=2
  advertised by
    fe80::20c:29ff:fec0:97ae%em0 (reachable)
2004:1::/64 if=em1
flags=LAO vltime=2592000, pltime=604800, expire=29d23h50m34s, ref=2
  advertised by
    fe80::20c:29ff:fec0:97b8%em1 (reachable)
fe80::%em1/64 if=em1
flags=LAO vltime=infinity, pltime=infinity, expire=Never, ref=0
  No advertising router
fe80::%em0/64 if=em0
flags=LAO vltime=infinity, pltime=infinity, expire=Never, ref=0
  No advertising router
fe80::%lo0/64 if=lo0
flags=LAO vltime=infinity, pltime=infinity, expire=Never, ref=0
  No advertising router
```

Neighbor Cache (default routers in *BSD)

- Sample output of “`ndp -r`” (BSDs):

```
% ndp -r
fe80::20c:29ff:fec0:97b8%em1 if=em1, flags=, pref=medium, expire=20m23s
fe80::20c:29ff:fec0:97ae%em0 if=em0, flags=, pref=medium, expire=26m53s
```

IPv6 SLAAC

some sample attacks…

Disable an Existing Router

- Forge a Router Advertisement message that impersontes the local router
- Set the “Router Lifetime” to 0 (or some other small value)
- As a result, the victim host will remove the router from the “default routers list”

Exploit DAD for Denial of Service

- Listen to Neighbor Solicitation messages with the Source Address set to the IPv6 “unspecified” address (::).
- When such a message is received, respond with a Neighbor Advertisement message
- As a resul, the address will be considered non-unique, and DAD will fail.
- The host will not be able to use that “tentative” address

Advertise Malicious Network Parameters

- An attacker could advertise malicious network parameters for the purpose of Denial of Service or performance-degrading.
- For example, it could advertise a very small Current Hop Limit such that packets be discarded by the intervenning routers

Possible countermeasures

- Deploy SeND (SEcure Neighbor Discovery)
- Monitor Neighbor Discovery traffic (e.g., with NDPMon)
- Restrict access to the local network
- Deploy Router Advertisement Guard (RA-Guard)

Router Advertisement Guard

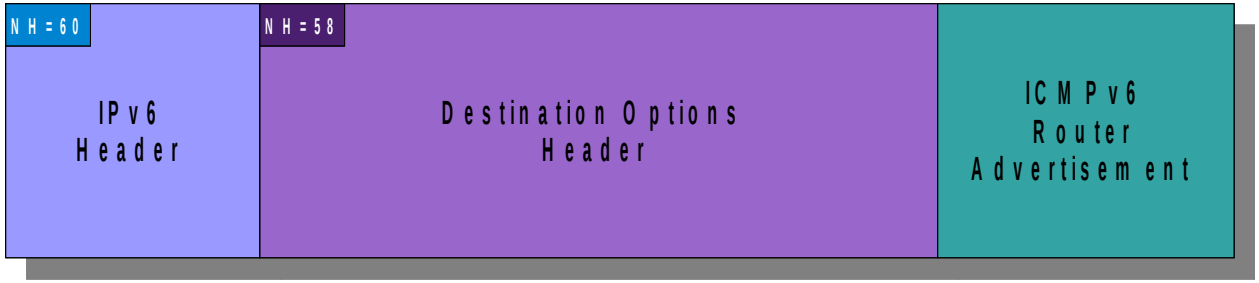
- Many organizations employ “RA-Guard” as the first line of defense against attacks based on forged Router-Advertisements
- RA-Guard works (roughly) as follows:
  - A layer-2 device is configured such that it accepts Router Advertisements on a specified port.
  - Router Advertisement messages received on other port are silently dropped (At layer-2)
- The RA-Guard mechanism relies on the device’s ability to identify Router Advertisement messages

Problem Statement

- The specifications allow for the use of multiple extension headers, even of the same type – and implementations support this.
- This is even allowed for Neighbor Discovery messages, that currently make no legitimate use of IPv6 Extension Headers.
- Thus, the structure of the resulting packet becomes increasingly complex, and packet filtering becomes virtually impossible.

RA-Guard: Evasion technique #1

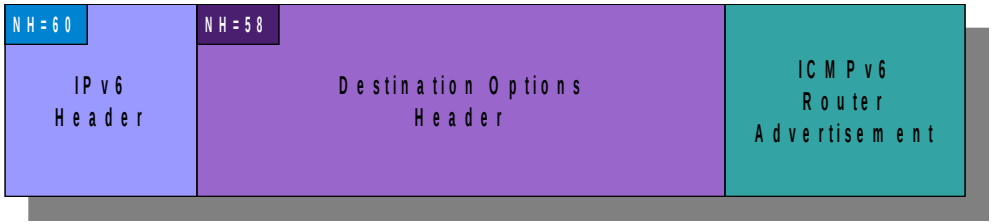
- RA-Guard implementations fail to process the entire IPv6 header chain



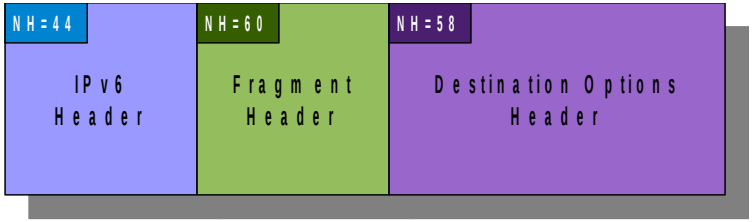
RA-Guard: Evasion technique #2

- Combination of a Destination Options Header and fragmentation:

Original Packet

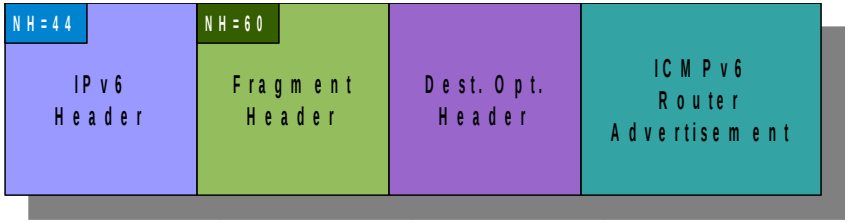


First Fragment



Can only tell there's ICMv6 inside

Second Fragment

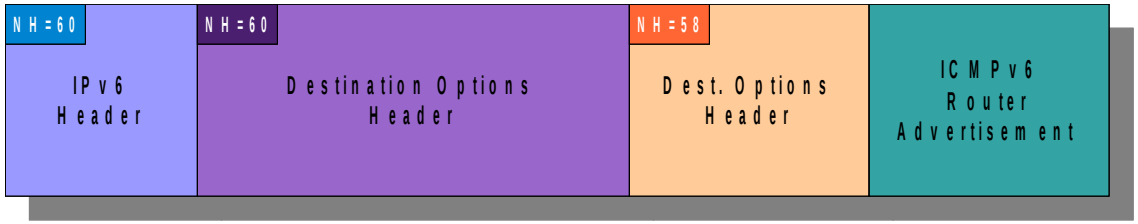


Can only tell there's Dest. Opt. Hdr inside!

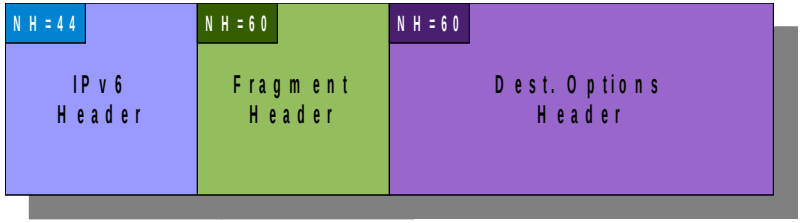
RA-Guard: Evasion technique #2(++)

- Two Destination Options headers, and fragmentation:

Original Packet

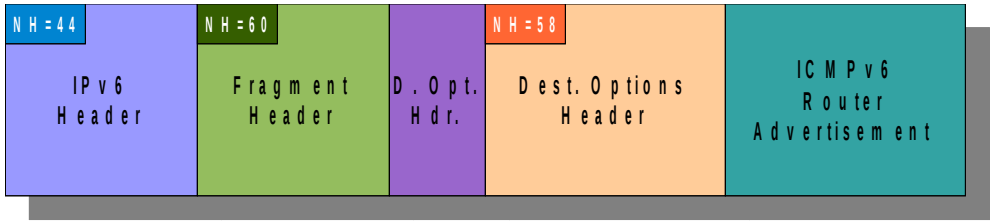


First Fragment



Can only tell there's Dest. Opt. Hdr inside!

Second Fragment



Can only tell there's Dest. Opt. Hdr inside!

Some comments about RA-Guard

- The use of a single “Destination Options” header is enough to evade most implementations of RA-Guard.
- If a Fragment Header is combined with two Destination Options headers, it becomes impossible for layer-2 device to filter forged Router Advertisements.
- This technique can also be exploited to circumvent Neighbor Discover monitoring tools such as NDPMon
- See my ongoing work on RA-Guard evasion:
  - http://tools.ietf.org/id/draft-gont-v6ops-ra-guard-evasion-01.txt
  - http://tools.ietf.org/id/draft-gont-6man-nd-extension-headers-01.txt
  - Or http://tools.ietf.org/id/gont

Some sysctl’s for autoconf (OpenBSD)

- `net.inet6.ip6.accept_rtadv` (defaults to 1): Controls whether Router Advertisements are accepted.
- `net.inet6.ip6.dad_count` (defaults to 1): Number of DAD probes sent when an interface is first brought up
- `net.inet6.ip6.maxifprefixes` (defaults to 16): Maximum number of prefixes per interface.
- `net.inet6.ip6.maxifdefrouters` (defaults to 16): maximum number fo default routers per interface.

IPv6 super-cookies

- When SLAAC is employed, the Interface ID is set to a Modified EUI-64 Identifier (based on the MAC address)
- Since MAC addresses are globally-unique, this results in a “super-cookie” (no, I didn't coin the term myself :-))
- Hosts can be traced as they move from one network to another
  - The prefix will change, but the globally-unique Interface Identifier will remain the same

IPv6 Privacy Extensions

- To MITIGate this privacy issue, “Privacy Extensions for SLAAC” were standardized (RFC 4941)
- Basically, the MAC-derived ID is replaced with a randomly-generated ID, and addresses are regenerated over time
  - This may be undesirable in some scenarios, since it makes logging harder
- Some OSes use (?) an alternative scheme:
  - The Interface ID is selected from a result of a hash function over the prefix and some secret value
  - Addresses are “constant” for any given prefix
  - But the Interface-ID changes as the host moves
  - This approach has the best of the “two worlds”

sysctl's for Privacy Addresses

- Privacy extensions for autoconf is implemented in FreeBSD (but not in, e.g., OpenBSD)
- These sysctl's control their operation:
  - `net.inet6.ip6.use_tempaddr` (defaults to 0)
    - Controls whether Privacy addresses are configured
  - `net.inet6.ip6.temppltime` (defaults to 86400)
    - Specifies the "preferred lifetime" for privacy addresses
  - `net.inet6.ip6.tempvltime` (defaults to 604800)
    - Specifies the "valid lifetime" for privacy addresses
  - `net.inet6.ip6.prefer_tempaddr` (defaults to 0)
    - Controls whether privacy addresses are "preferred" (i.e., whether outgoing "conections" should use privacy addresses)

Dynamic Host Configuration Protocol version 6 (DHCPv6)

Brief Overview

- IPv6 version of DHCPv4: mechanism for stateful configuration
- It implements “prefix delegation”, such that a DHCPv6 server can assign not only an IPv6 address, but also an IPv6 prefix.
- It is an optional mechanism which is invoked only if specified by Router Advertisement messages.
- It used to be the only mechanism available to advertise recursive DNS servers
- It can be exploited in a similar way to Router Advertisement messages.
- It suffers the same problems as IPv6 SLAAC:
  - If no authentication is enforced, it is trivial for an attacker to forge DHCPv6 packets
  - Layer2- mitigations can be easily circumvented with the same techniques as for RA-Guard

Multicast Listener Discovery

Brief Overview

- A generic protocol that allows hosts to inform local routers which multicast groups they are interested in.
- Routers use the infomation to decide which packets must be forwarded to the local segment.
- Since Neighbor Discovery uses multicast addresses (the solicited-node multicast addresse), MLD is used by all IPv6 nodes
- In practice, they only use for MLD with Neighbor Discovery is MLD-snooping switches – switches that interpret MLD packet to decide on which ports packets should be forwarded.
- Potential issues: If a MLD-snooping switch is employed, MLD could be exploited for Denial of Service attacks.
- MLDv2 implements per-source filtering capabilities, and greatly increases the complexity of MLD(v1).
- Security-wise, MLDv1 should be preferred.

IPsec Support

Brief overview and considerations

Myth: "*IPv6 is more secure than IPv4 because security was incorporated in the design of the protocol, rather than as an 'add-on'*"

- This myth originated from the fact that IPsec support is mandatory for IPv6, but optional for IPv4
- In practice, this is irrelevant:
  - What is mandatory is IPsec support, not IPsec usage.
  - And nevertheless, many IPv4 implementations support IPsec, while there exist IPv6 implementations that do not support IPsec.
  - Virtually all the same IPsec deployment obstacles present in IPv4 are also present in IPv6.
- The IETF has acknowledged this fact, and is currently changing IPsec support in IPv6 to "optional"
- Conclusion: there is no reason to expect increased use of IPsec as a result of IPv6 deployment

DNS support for IPv6

Brief Overview

- AAAA (Quad-A) records enable the mapping of domain names to IPv6 addresses
- The zone “ip6.arpa” is used for the reverse mapping (i.e., IPv6 addresses to domain names)
- DNS transport can be IPv4 and/or IPv6
- Troubleshooting tools such as “dig” already include support for IPv6 DNS features
- Security implications:
  - Increased size of DNS responses due to larger addresses might be exploited for DDos attacks

Looking for IPv6-enabled hosts

- The dig tool can be used to investigate IPv6-related DNS Reseource Records. Example:

```
$ dig www.si6networks.com aaaa
; <<>> DiG 9.7.3 <<>> www.si6networks.com aaaa
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12806
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.si6networks.com.       IN   AAAA

;; ANSWER SECTION:
www.si6networks.com.  12666     IN   AAAA 2a02:27f8:1025:18::232

;; Query time: 1 msec
;; SERVER: 172.31.252.1#53(172.31.252.1)
;; WHEN: Wed Nov 16 01:04:38 2011
;; MSG SIZE  rcvd: 65
```

IPv6 reverse mapping

- The dig tool can be also used to obtain the reverse mappings. Example:

```
$ dig -x 2a02:27f8:1025:18::232

; <<>> DiG 9.7.3 <<>> -x 2a02:27f8:1025:18::232
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34592
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;2.3.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.1.0.0.5.2.0.1.8.f.7.2.2.0.a.2.ip6.arpa. IN PTR

;; ANSWER SECTION:
2.3.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.1.0.0.5.2.0.1.8.f.7.2.2.0.a.2.ip6.arpa. 1000 IN PTR
srv01.bbserve.nl.

;; Query time: 269 msec
;; SERVER: 172.31.252.1#53(172.31.252.1)
;; WHEN: Wed Nov 16 01:12:48 2011
;; MSG SIZE  rcvd: 120
```

IPv6 Transition Co-Existence Technologies

IPv6 Transition/Co-existence Technologies

- Original transition plan: “deploy IPv6 before we ran out of IPv4 addresses, and eventually turn off IPv4 when no longer needed” – *it didn’t happen*
- *The c*urrent transition/co-existence plan is based on a toolbox:
  - Dual-stack
  - Tunnels
  - Translation
- Their use is intented for different networks setups
- Dual-stack is enabled by default in most general-purpose OSes
- Some transition mechanisms (e.g. Teredo an ISATAP) are enabled by default in smoe OSes (e.g. Windows Vista and Windows 7)

Transition Technologies

Dual Stack

Dual-stack

- Each node supports both IPv4 and IPv6
- Domain names include both A and AAAA (Quad A) records
- IPv4 or IPv6 are used as needed
- Dual-stack was the original transition co-existence plan, and still is the recommended strategy for servers
- Virtually all popular operating systems include native IPv6 support enabled by default

Exploiting Native IPv6 Support

- An attacker can connect to an IPv4-only network, and forge IPv6 Router Advertisement messages. (*)
- The IPv4-only hosts would configure IPv6 connectivity
- IPv6 could be leveraged to evade network security controls (if the network ignores IPv6)
- Possible counter-measures:
  - Implement IPv6 security controls, even on IPv4-only networks.
  - Disable IPv6 support in nodes that are not expected to use IPv6

(*) http://resources.infosecinstitute.com/slaac-attack/

Exploiting Native IPv6 Support (II)

- Some applications may be IPv6-enabled, but may have unexpected behaviors when IPv6 is employed.
- They may crash, fail to log users (*), etc.
- Example:
- Possible counter-measures:
  - Implement IPv6 security controls, even on IPv4-only networks.
  - Disable IPv6 support in nodes that are not expected to use IPv6

(*) Gmail complete anonymity possible with IPv6. Post to the full-disclosure mailing-list. Available at: http://lists.grok.org.uk/pipermail/full-disclosure/2010-August/075876.html

Transition Technologies

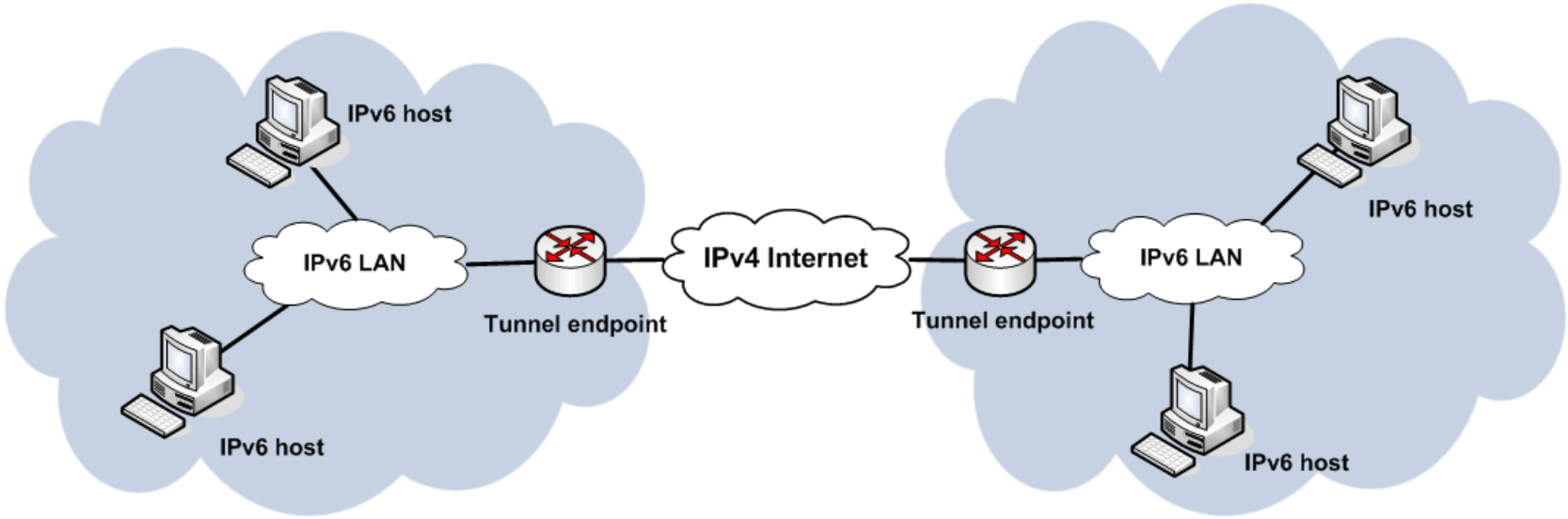
Tunnels

Tunnels

- Use the existing IPv4 Internet to transport IPv6 packets from/to IPv6 islands
- Tunnels can be:
  - configured: some sort of manual configuration is needed
  - automatic: the tunnel end-points are derived from the IPv6 addresses
- Configured tunnels:
  - 6in4
  - Tunnel broker
- Automatic tunnels:
  - ISATAP
  - 6to4
  - 6rd
  - Teredo

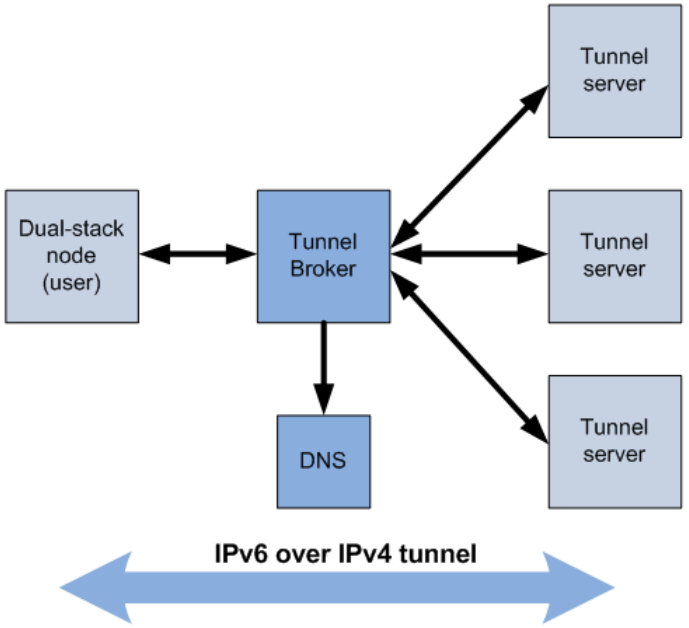
6in4

- The tunnel endpoints must be manually configured
- Management can be tedious
- Security may be used as needed (e.g., IPsec)
- May operate across NATs (e.g. IPsec UDP encapsulation, or if the DMZ function is employed)



Tunnel broker

- The Tunnel Broker is model to aid the dynamic establishment of tunnels (i.e., relieve the administrator from manual configuration)
- The TB is used to manage the creation, modification or deletion of a tunnel
- Example: “Tunnel Broker with the Tunnel Setup Protocol (TSP)

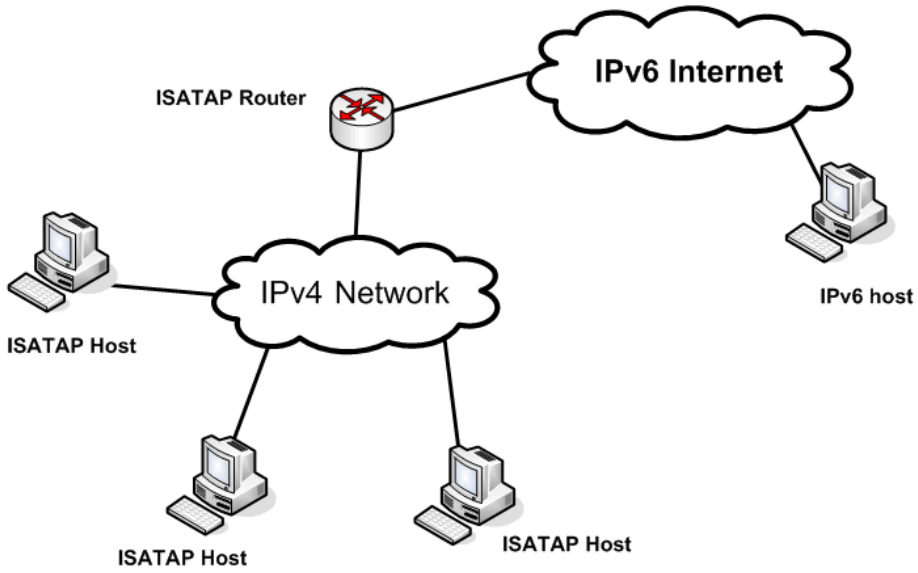


Tunnel Broker: Sample Implementation

- Gogoc is a tunnel broker implementation
- It even allows “anonymous” tunnel establishment (no account needed)
- Install it, and welcome to the IPv6 Internet!
- Privacy concerns: Beware that all your traffic will most likely follow a completely different path from your normal IPv4 traffic.

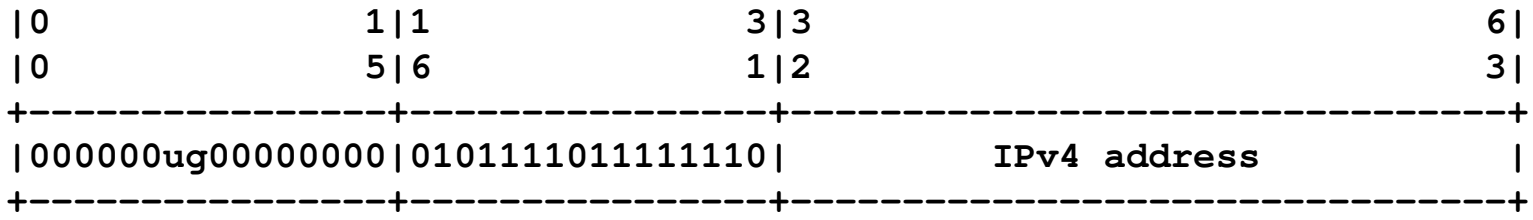
ISATAP: Brief Overview

- Intra-Site Automatic Tunnel and Addressing Protocol
- Aims at enabling IPv6 deployment withing a site with no IPv6 infrastructure -- does not work across NATs



ISATAP: Address format

- ISATAP uses normal global prefixes
- However, a special format is specified for the Interface ID, such that it encodes the IPv4 address of the ISATAP host.



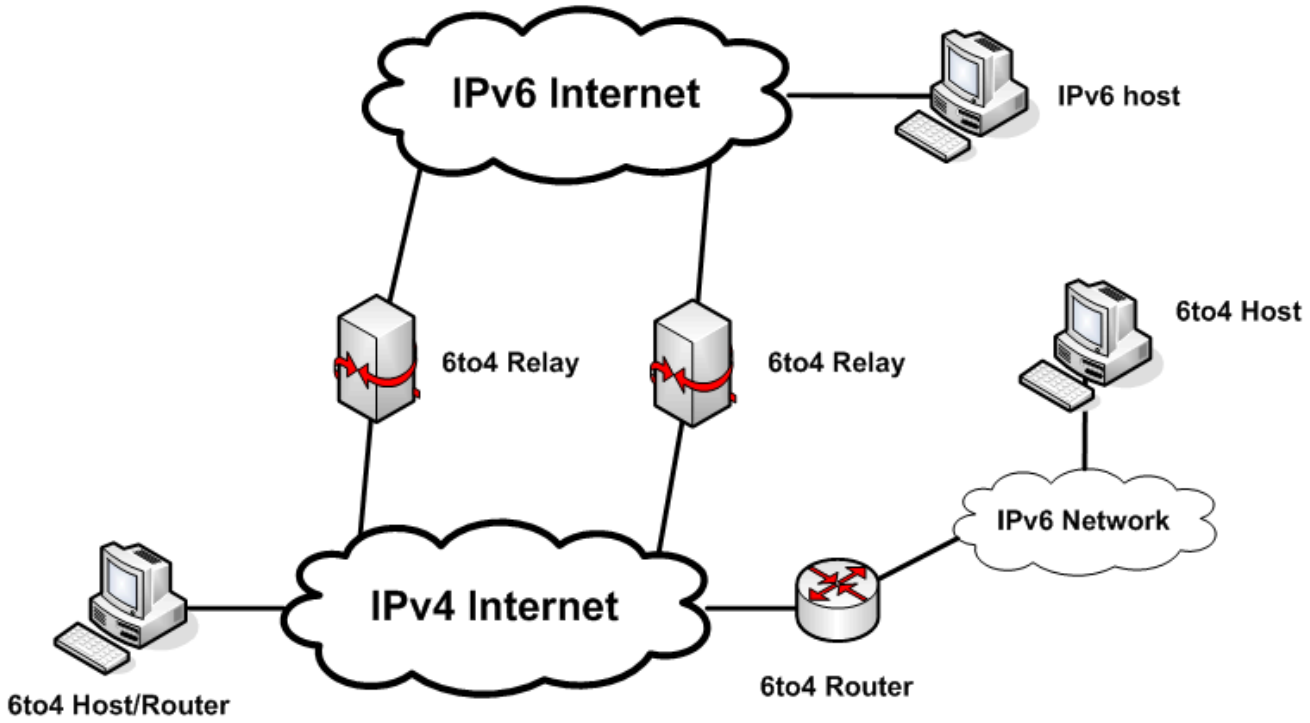
- ISATAP hosts learn the IPv4 address of the ISATAP router by resolving the name “isatap”.
- On the other hand, when an ISATAP router receives a native IPv6 packet destined to one of its ISATAP hosts, it learns the hosts' IPv4 address from the Interface ID

Exploting ISATAP

- Microsoft implementations “learn” the IPv4 address of the ISATAP router by resolving the name “isatap” (via DNS and others)
- An attacker could forge name resolution responses to:
  - Impersonate a legitimate ISATAP router
  - Enable IPv6 connectivity in an otherwise IPv4-only network
- This could be used in conjunction with other attacks (e.g. forging DNS responses such that they contain AAAA records)

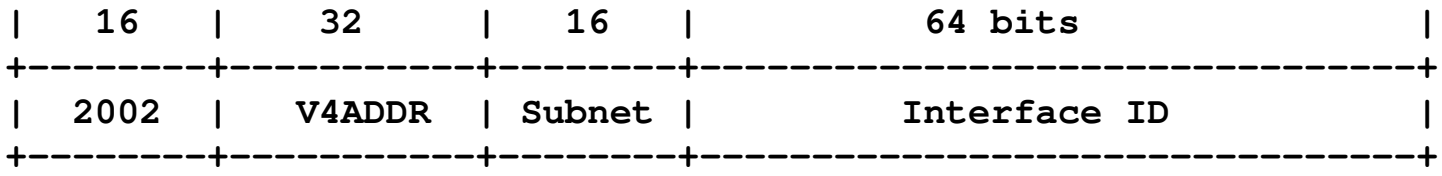
6to4: Brief overview

- Enables IPv6 deployment in sites with no global IPv6 connectivity - does not work across NATs (unless the DMZ function is used)
- 6to4 architecture:



6to4: Address format

- 6to4 addresses use the special prefix 2002::/16
- They encode the tunnel endpoint in part of the network prefix
- On the IPv6 world, they are treated as normal addresses
  - The prefix can be used for autoconfiguration
  - Packets are router towards ASes advertising reachability to the 2002::/16



6to4: Packets originating at 6to4 hosts

- Packets originate at a 6to4 host as native IPv6 packets
- A 6to4 router encapsulates the packet in IPv4, and sets the IPv4 Destination Address to:
  - that of a 6to4 relay (if the IPv6 destination is a native IPv6 host)
  - That of the corresponding 6to4 router (if the IPv6 Destination is a 6to4 host)
- The receiving 6to4 relay decapsulates the packets and forwards them on the native IPv6 network
- The receiving 6to4 router decapsulates the packets, and forwards them on the 6to4-powered IPv6 network.

6to4: Packets originating from IPv6 hosts

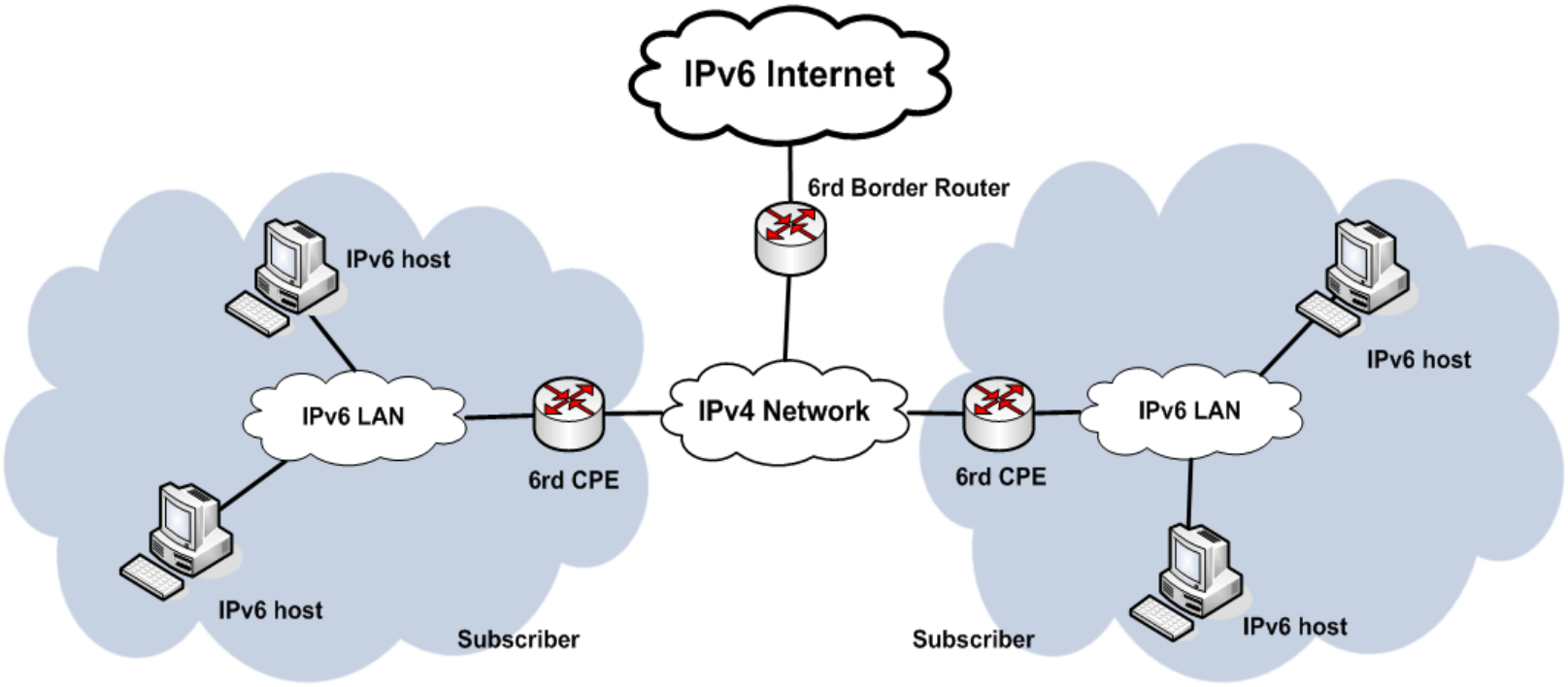
- Packets are routed in the native IPv6 Internet towards ASes announcing reachability to the 2002::/16 prefix
- Those ASes have deployed 6to4 relays, which help “bridge” the IPv4 and the IP6 Internets
- They encapsulate the aforementioned packets in IPv4, and set the IPv4 Destination Address to the one encoded in the 6to4 address
- The 6to4 router decapsulates the IPv6 packets, and forwards it to the “local” IPv6 network
- The packet travels over the IPv4 Internet to the 6to4 router
- The IPv4 router decapsultes the IPv6 packet, and forwards it to the 6to4-powered IPv6 network

Problems with 6to4

- Lots of poorly-managed 6to4 relays have been deployed
- In most cases they introduce PMTUD black-holes (e.g. as a result of ICMPv6 rate-limiting)
- Lack of control of which 6to4 relays are used make troubleshooting difficult
  - Use of the 6to4 anycast address makes it difficult to identify a poorly-managed relay in the 6to4 -> native IPv6 direction
  - It is always difficult to troubleshoot problems in the native IPv6 -> 6to4 direction (the user has no control over which relay is used)
- Privacy concerns:
  - 6to4 traffic might take a completely different path than IPv4 traffic

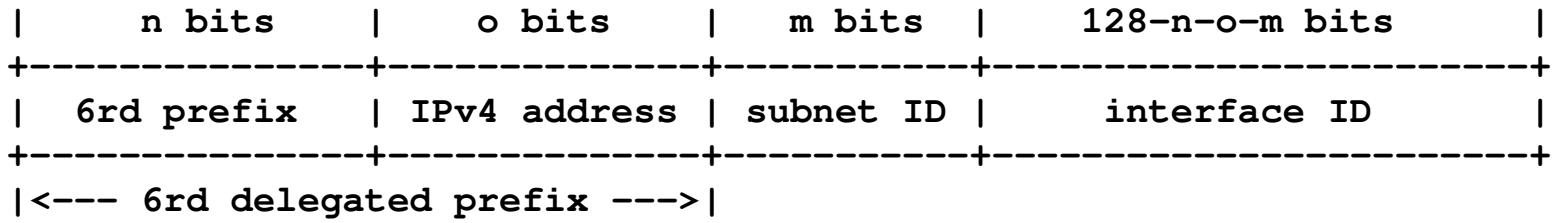
6rd: Brief overview

- 6rd stands for “IPv6 rapid deployment”
- Enables IPv6 deployment in a site with no IPv6 infrastructure
- Builds upon 6to4 – but the whole system is implemented within a site



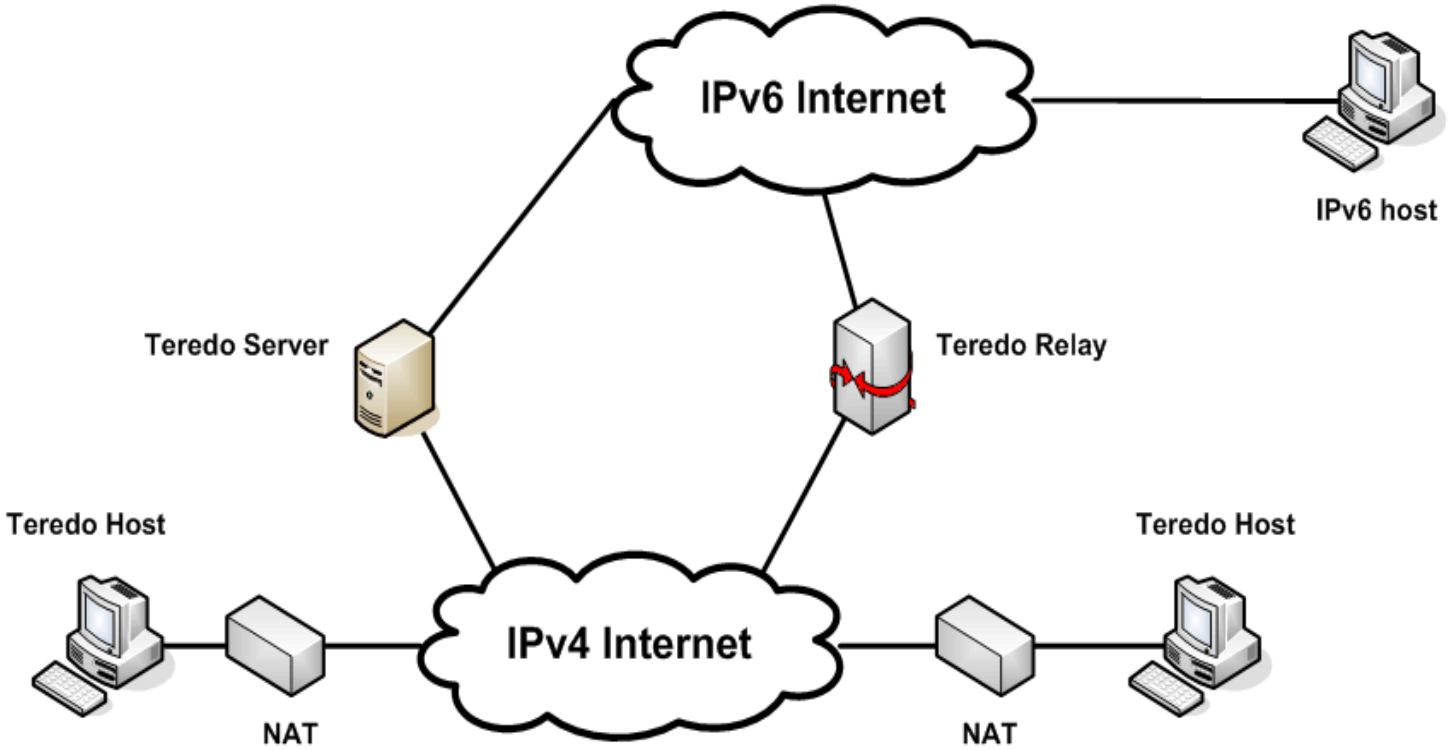
6rd: Address format

- 6rd uses no special prefixes – normal IPv6 Global Unicast addresses are employed
- But the addresses encode the tunnel endpoint in the prefix
  - This is only known to the 6rd routers
  - Is transparent to the rest of the world
- 6rd address format:



Teredo: Brief overview

- Aims at providing IPv6 connectivity to individual hosts behind one or more NATs -- “last resort” mechanism for IPv6 connectivity
- It tunnels IPv6 packets over UDP/IPv4

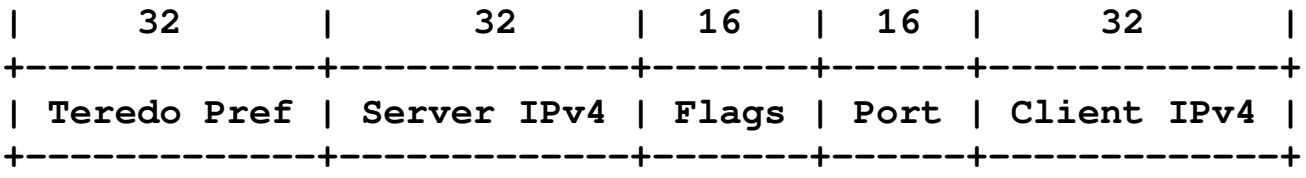


Teredo: Brief overview (II)

- Each Teredo client is associated with a Teredo server
- The Teredo acts as an agent to the client, such that the client is reachable from the public Internet
- Teredo systems (hosts or relays) willing to send packets to the Teredo client talk with the corresponding Teredo server
- “Holes” will be punched in the NAT as needed
- Teredo is a “smart” transition mechanism... but the resulting performance is usually as bas as it could possibly get.

Teredo: Address format

- Teredo uses a special prefix
- The Teredo address encodes:
  - The Teredo server's IPv4 address
  - The Teredo client's IPv4 address
  - The Teredo client's UDP port
- Teredo address format:



Securiy Implications of Teredo

- Teredo increases the host exposure to attack
- Hosts behind a NAT may become reachable from the public Internet
- Windows systems obtain the address of a Teredo serving by resolving “teredo.ipv6.microsoft.com”
- An attacker could impersonate a Teredo server if he can attack the DNS
- Privacy concerns:
  - Teredo traffic might take a completely different path than IPv4 traffic

Transition Technologies

Translation

Brief overview

- All of the previous transition/co-existence technologies require assignment of both IPv4 and IPv6 addresses – But ...what if there are no IPv4 addresses left?
- A number of technologies have been developed to share IPv4 addresses at a large scale:
  - CGN (Carrier-Grade NAT)
  - A+P
- Additionally, NAT64 has been developed, such that IPv6-only hosts can access IPv4-only hosts

The future doesn’t look like NAT-free…..

Security impications

- Translation introduces a “single point of failure” in the network
- They will be interesting targets for attackers
- Since the have been recently developed, they are likely to be buggy

Security Implications of IPv6 on IPv4 Networks

Security Implications on IPv4 Networks

Transition Technologies

Exploiting Transition Technologies

- Some systems (notably Windows) have support of trnasition technologies enabled by default.
- These technologies could be used to circumvent security controls.
- Technologies such as Teredo could increase the attack expoure of hosts
- Possible countermeasures:
  - Enforce IPv6 security controls on IPv4 networks.
  - Disable support of these technologies.
  - Deploy packet filtering policies, such that these technologies are blocked.

Filtering IPv6 Transition Technologies

Transition Technology	Filtering rule
Dual-Stack	Automatic (if network does not support IPv6)
IPv6-in-IPv4 tunnels	IPv4 Protocol == 41
6to4	IPv4.Protocol == 41 && IPv4.{src,dst} == 192.88.99.0/24
ISATAP	IPv4 Protocol == 41
Teredo	IPv4.dst == known_teredo_servers && UDP.DstPort == 3544
TSP	IPv4.dst == known_teredo_servers && {TCP,UDP}.dst == 3653

IPv6 Network Reconnaissance

IPv6 Network Reconnaisance

Host scanning

Leveraging IPv6 features

- ICMPv6 echo/request response
- Traceroute6 (based on ICMPv6 errors)
- ICMPv6 Node Information messages
- IPv6 options of type 10xxxxxx
- IPv6 multicast addresses
- Sniffing
- Special IPv4 addresses used for trasition technologies (e.g., Teredo)

Multicast addresses

- Multicast address (e.g. ff02::1) can be leveraged for host scanning
- However, some stacks (notably Windows Vista/7) do not respond to Echo Requests sent to multicast addresses
- Trick: send packets with unsupported options of type 10xxxxxx
  - Even Windows Vista/7 responds to these!
- Note: Hosts will tipically respond using a link-local unicast (fe80::/10) address – i.e., this technique does not discover global address
- Global addresses can be obtained, indirectly:
  - Learn link-local addresses of hosts
  - Learn global prefixes used in the subnet
  - Form global addresses with the global prefixes and the Interface ID of the local address
  - Check that the addresses actually exist

Application-layer protocols

- A number of applications may leak IPv6 addresses:
  - E-mail headers
  - P2P applications
- Together with maling-list archives and popular search engines, they may be an interesting vector for network reconnaisance

Example of application-layer leakeage

- Sample e-mail header:

```
X-ClientAddr: 46.21.160.232
Received: from srv01.bbserve.nl (srv01.bbserve.nl [46.21.160.232])
       by venus.xmundo.net (8.13.8/8.13.8) with ESMTP id p93Ar0E4003196
       for <fernando@gont.com.ar>; Mon, 3 Oct 2011 07:53:01 -0300
Received: from [2001:5c0:1000:a::943]
       by srv01.bbserve.nl with esmtpsa (TLSv1:AES256-SHA:256)
       (Exim 4.76)
       (envelope-from <fgont@si6networks.com>)
       id 1RAg8k-0000Qf-Hu; Mon, 03 Oct 2011 12:52:55 +0200
Message-ID: <4E8993FC.30600@si6networks.com>
Date: Mon, 03 Oct 2011 07:52:44 -0300
From: Fernando Gont <fgont@si6networks.com>
Organization: SI6 Networks
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.23)
Gecko/20110922 Thunderbird/3.1.15
MIME-Version: 1.0
To: Fernando Gont <fernando@gont.com.ar>
Subject: Prueba
```

DNS

- IPv6 addresses can be obtained by querying the DNS for AAAA records.
- Many sites currently use domain names such as “ipv6*”
- For example, you may google for “site:ipv6*” and “site:ip6*”

Network “Neighborhood” protocols

- mDNS is increasily used for discovering peers on the same network.
- Not IPv6-specific, but could be employed with IPv6, too.
- Hosts announce themselves on the network, for “ocassional” networking.
- This provides yet another vector for network reconnaissance

IPv6 Network Reconnaisance

Port scanning

IPv6 port-scanning

- IPv6 port scanning remains the same as in IPv4
- Nmap may be used for such purpose

```
# nmap -6 -p1-10000 -n 2000:db8::1
80/tcp open http
135/tcp open msrpc
445/tcp open microsoft-ds
554/tcp open rtsp
1025/tcp open NFS-or-IIS
1026/tcp open LSA-or-nterm
1027/tcp open IIS
1030/tcp open iad1
1032/tcp open iad3
1034/tcp open unknown
1035/tcp open unknown
1036/tcp open unknown
1755/tcp open wms
9464/tcp open unknown
```

Key areas in which further work is needed

Key areas in which further work is needed

IPv6 resiliency

- Implementations have not really been the target of attackers, yet
- Only a handful of publicly available attack tools
- Lots of vulnerabilities and bugs still to be discovered.

IPv6 support in security devices

- IPv6 transport is not broadly supported in security devices (firewalls, IDS/IPS, etc.)
- This is key to be able enforce security policies comparable with the IPv4 counterparts

Education/Training

- Pushing people to “Enable IPv6” point-and-click style is simply insane.
- Training is needed for engineers, technicians, security personnel, etc., before the IPv6 network is running.

20 million engineers need IPv6 training, says IPv6 Forum
The IPv6 Forum - a global consortium of vendors, ISPs and national research & Education networks - has launched an IPv6 education certification programme in a bid to address what it says is an IPv6 training infrastructure that is "way too embryonic to have any critical impact.“ (**http://www.itwire.com**)

Some Conclusions

Some conclusions…

- Beware of IPv6 marketing and mythology! – “assumption is the mother of all...err...problems” :-)
- While IPv6 provides similar features than IPv4, it uses different mechanisms – and the devil is in the small details
- The security implications of IPv6 should be considered before it is deployed (not after!)
- Most systems have IPv6 support enabled by default, and this has implications on “IPv4-only” networks!
- Even if you are not planning to deploy IPv6 in the short term, most likely you will eventually do it
- It is time to learn about and experiment with IPv6!

Questions?

Thank you!

Fernando Gont
fgont@si6networks.com

IPv6 Hackers mailing-list
http://www.si6networks.com/community/

www.si6networks.com