



Hacking IPv6 Networks

Fernando Gont

(UTN/FRH, Argentina)

Hack In Paris 2011

Paris, France. June 14-17, 2011



Agenda (I)

- Objectives of this training
- Motivation for IPv6
- Brief comparison 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
- 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



Objectives of this training

- Provide an Introduction to IPv6
- Provide an objective discussion of IPv6 security issues
- Identify and analyze a number of security aspects that must be considered before deploying IPv6
- Identify and analyze the security implications of IPv6 on IPv4 networks
- Identify areas in which further work is needed
- Draw some conclusions regarding IPv6 security



Some general considerations about IPv6 security

Some interesting aspects about IPv6 security

- 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 internetworkin 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 comparison between IPv6 and IPv4

Brief comparison 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	Opcional	Recommended (<u>not</u> mandatory)
Fragmentation	Both in hosts and routers	Only in hosts

Brief comparison of IPv4 and IPv6 (II)

- Header formats:

IPv4 Header

0	4	8	12	16	20	24	28	31
Version	IHL	Type of Service	Total Length					
Identification				Flags	Fragment Offset			
Time to Live		Protocol		Header Checksum				
Source Address								
Destination Address								

IPv6 Header

0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	63
Version	Traffic Class		<i>Flow Label</i>					Payload Length			Next Header	Hop Limit				
Source Address																
Destination Address																



IPv6 header fields

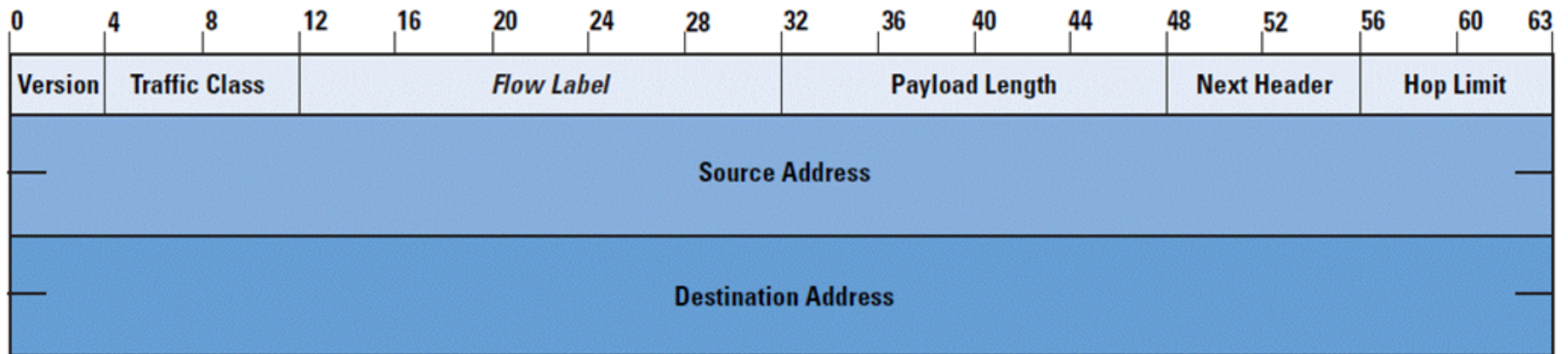


IPv6 header fields

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



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
- This field should be policed at the network edge

Flow Label

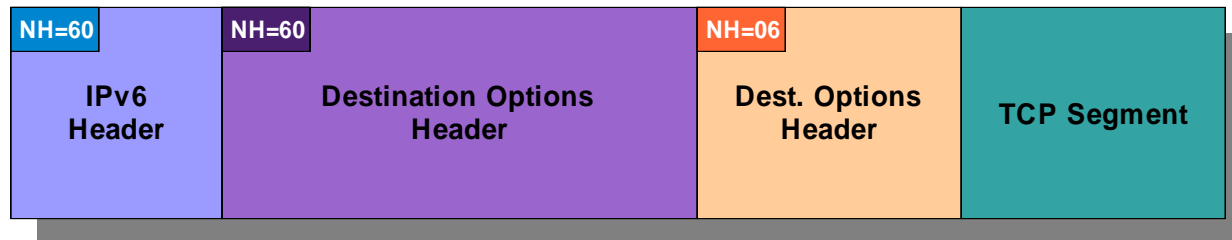
- The three-tuple {Source Address, Destination Address, Flow Label} was meant to identify a communication flow.
- Currently unused by many stacks – others use it improperly
- Specification of this header field, together with possible uses, is “work in progress” at the IETF.
- Potential vulnerabilities depend on the ongoing work at the IETF, but if the Flow Label is predictable:
 - 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 packet (without including the length of the fixed IPv6 header)
- Maximum IPv6 packet is 65855 bytes. However, IPv6 “Jumbograms” can be specified.
- Among the basic checks:
 - The IPv6 Payload Length cannot be larger than the “payload size” reported by the link-layer protocol

Next Header

- Identifies the header/protocol type following this header.
- Since IPv6 has a fixed-length header, options are included in “extension headers” (i.e., headers that sit between the IPv6 header and the upper-layer protocol)
- In IPv6, packets follow a “header chain” type structure. E.g.,



Hop Limit

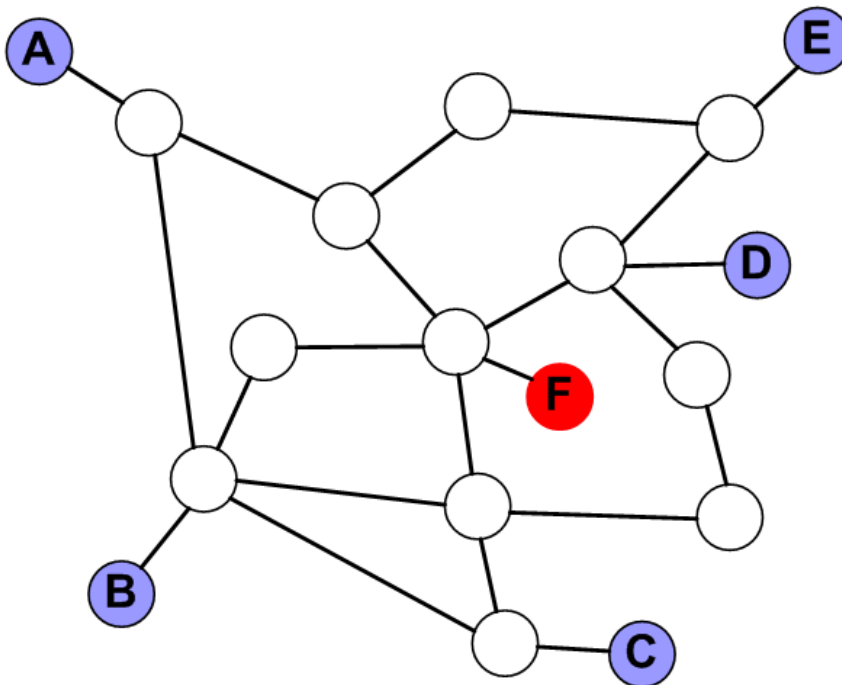
- Analogous to IPv4's "Time to Live" (TTL)
- Identifies the number of network links the packet may traverse
- Packets are discarded when the Hop Limit is decremented to 0.
- Could be leveraged for:
 - Detecting the Operating System of a remote node
 - Fingerprinting a remote physical device
 - Locating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

Hop Limit: Fingerprinting Devices or OSes

- Different Oses use different defaults for the “Hop Limit” (typically a power of two: 64, 128, etc.)
- If packets originating from the same IPv6 addresses contain very different “Hop Limits”, they might be originated by different devices. E.g.:
 - 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
 - Detecting the Operating System of a remote node
 - Note: mostly useless, since:
 - It requires different OSes behind the “middle-box”
 - There is only a reduced number of default “Hop Limit” values
- Depending on the inferred original “Hop Limit”, the possible OS could be guess (again, mostly useless)

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 original “Hop Limit”
- If we have multiple “sensors”, we can “triangulate” the position of the node



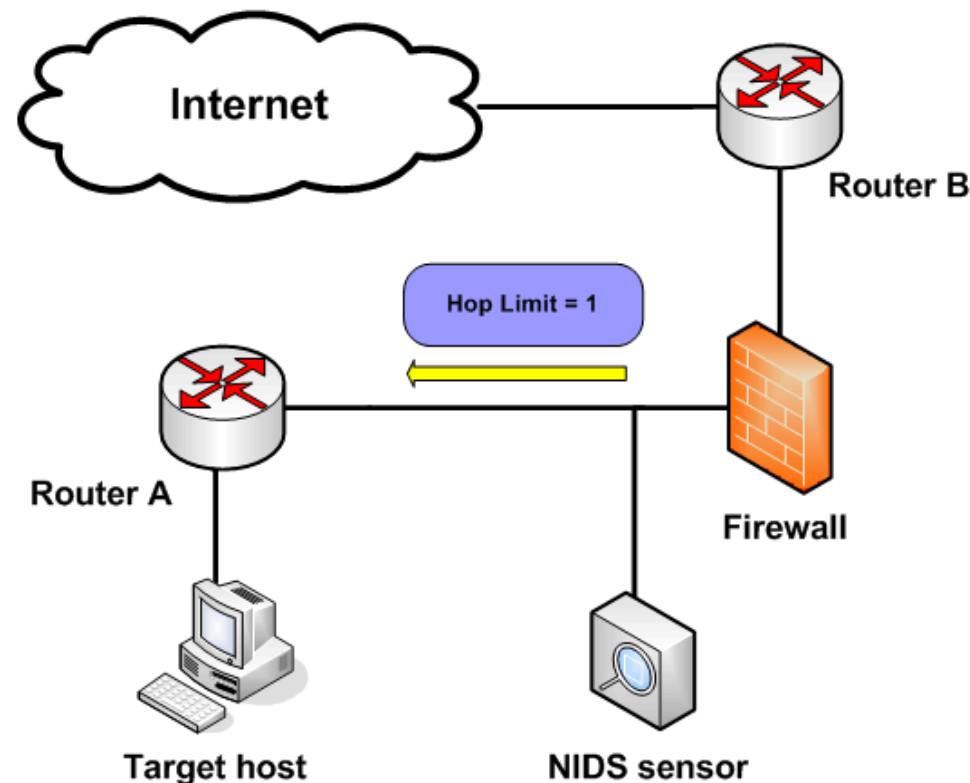
Source	Hop Limit
A	61
B	61
C	61
D	62

F is the only node that is:

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

Hop Limit: Evading NIDS

- Basic idea: set 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 incoming 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 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.)
- It’s common for a node to be using, at any given time, several addresses, of multiple types and scopes. For example,
 - One or more unicast link-local address
 - One or more global unicast address
 - One or more link-local address

Address Types

- 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

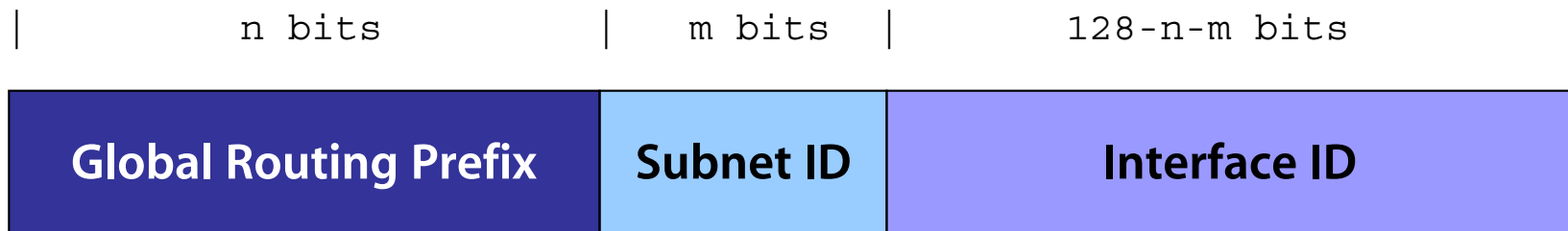


Unicast Addresses

- 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
- Global Unicast Addresses can be generated with multiple 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

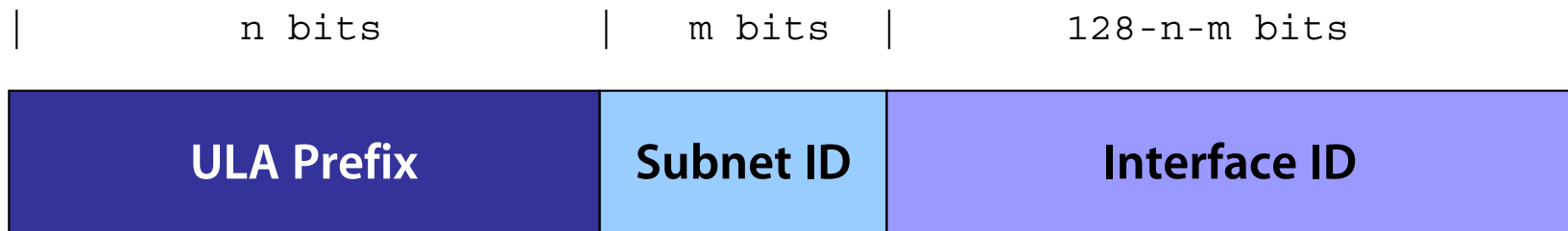
- Syntax of the link-local unicast addresses:



- 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
- Unique-local Unicast Addresses can be generated with multiple 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)



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




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 Connectivity

Brief Overview

- Because of the increased IPv6 address space, it is expected that each device connected to the Internet will have a unique address
- It is also assumed that this will “return” the “End-to-end Principle” to the Internet:
 - The network is transparent to the communication of any two nodes (e.g., intermediate nodes do not modify the TCP port numbers, etc.)
 - Any node can establish a communication node with any other node in the network (e.g., the network does not filter “incoming connections”)
 - It is usually argued that the “end-to-end principle” allows for Innovation

Some Considerations

- Even if each device has a unique address, that does not necessarily imply “end-to-end” connectivity
 - This is not necessarily a desired property in a production network
 - Thus, a typical IPv6 subnet will be protected by a stateful firewall that only allows “return traffic” (i.e., communications can only be initiated from the inside network)
- In practice, most production networks don’t really care about innovation, but rather about getting work done.
- And the users of these networks expect to use the same services currently available for IPv4 without “end-to-end” connectivity(web, email, social networks, etc.)



IPv6 Extension Headers

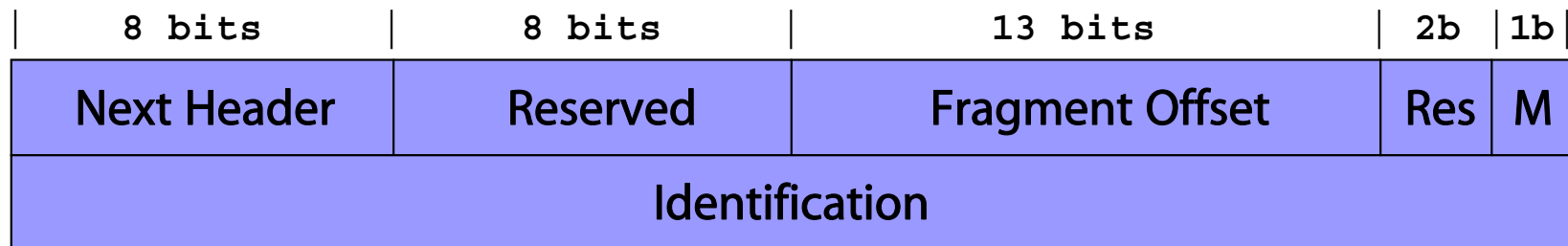


IPv6 Extension Headers

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 might allow “stealth” port scanning technique
- 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?

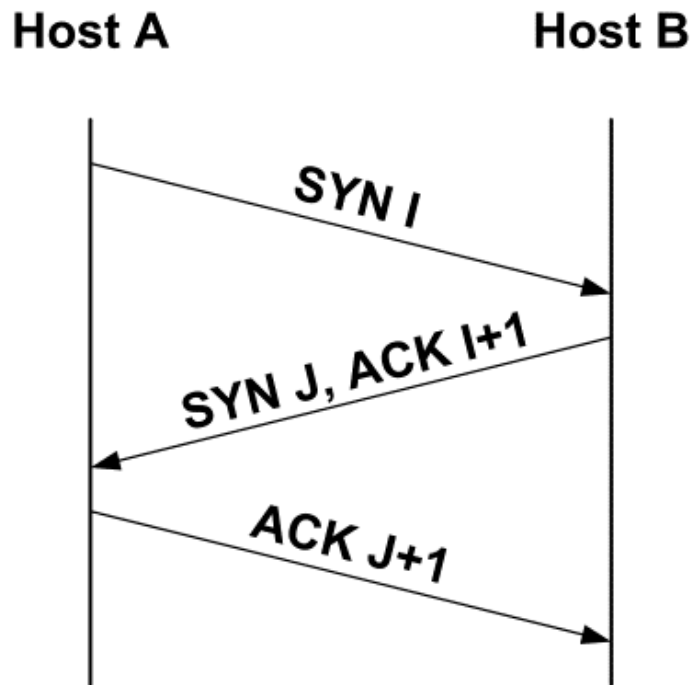
Example of Predictable Identification values

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

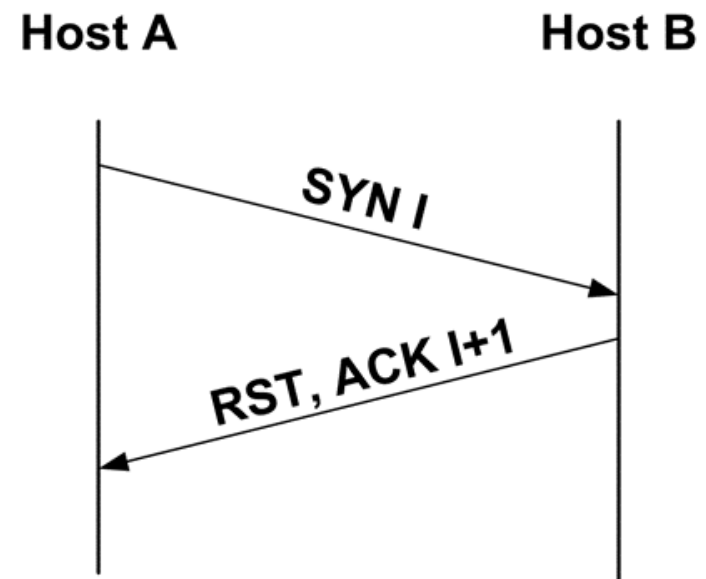
1. IP6 (hlim 64, next-header Fragment (44) payload length: 1456)
2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007a:0|1448) ICMP6, echo request, length 1448, seq 1
2. IP6 (hlim 64, next-header Fragment (44) payload length: 368)
2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007a:1448|360)
3. IP6 (hlim 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

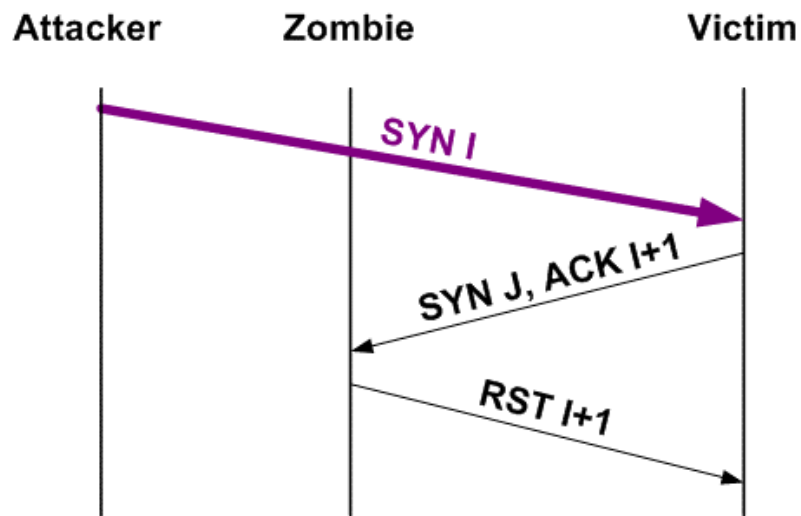


Connection-rejected

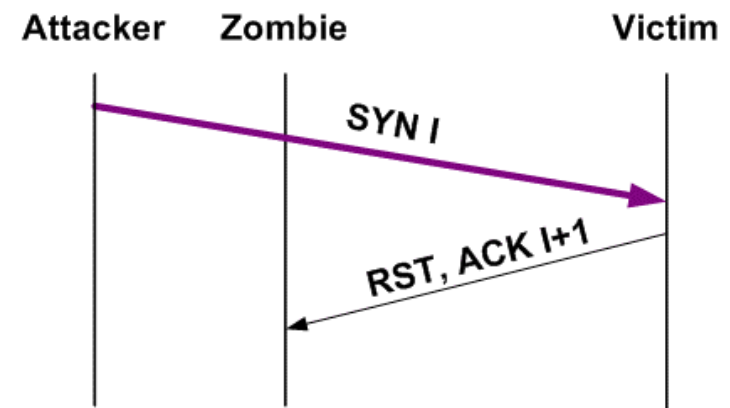


Forged TCP Connection-Establishment

Open port



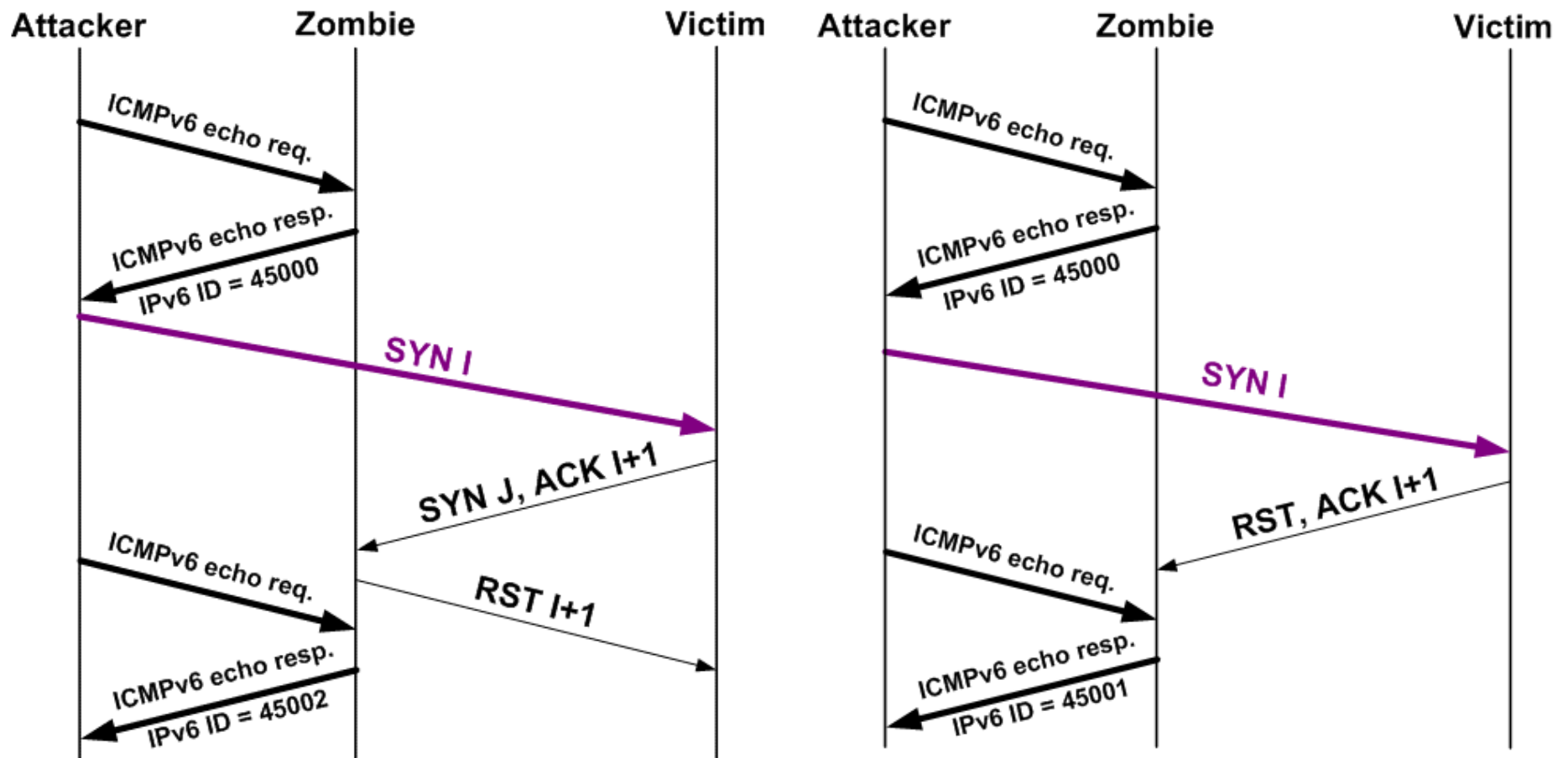
Closed port



IPv6 Idle Scan

Open port

Closed port



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
- Vulnerable implementations:
 - Linux
 - Possibly others
- Relevant vendors have been notified (today)

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

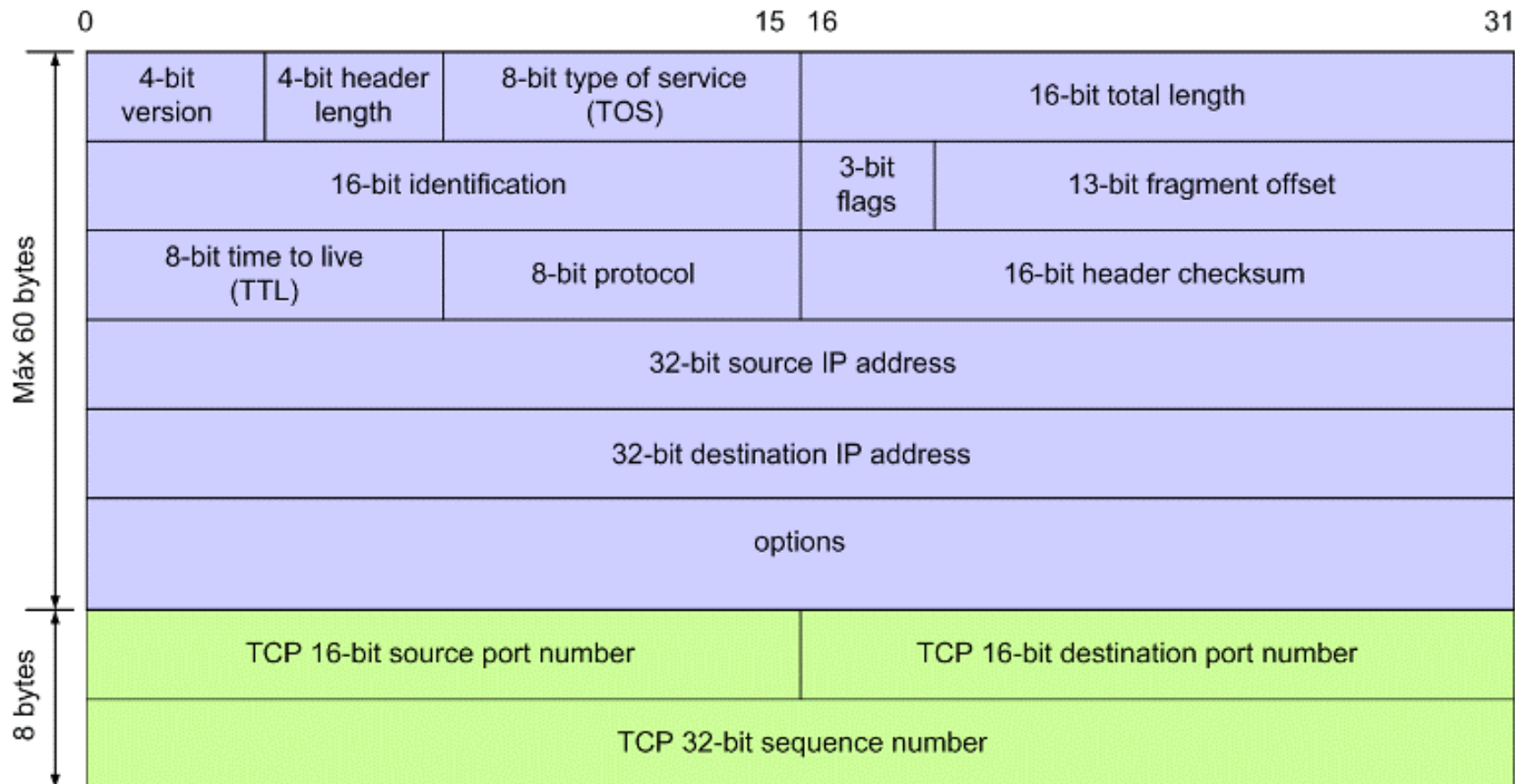


IPv6 Extension Headers

Implications on Firewalls

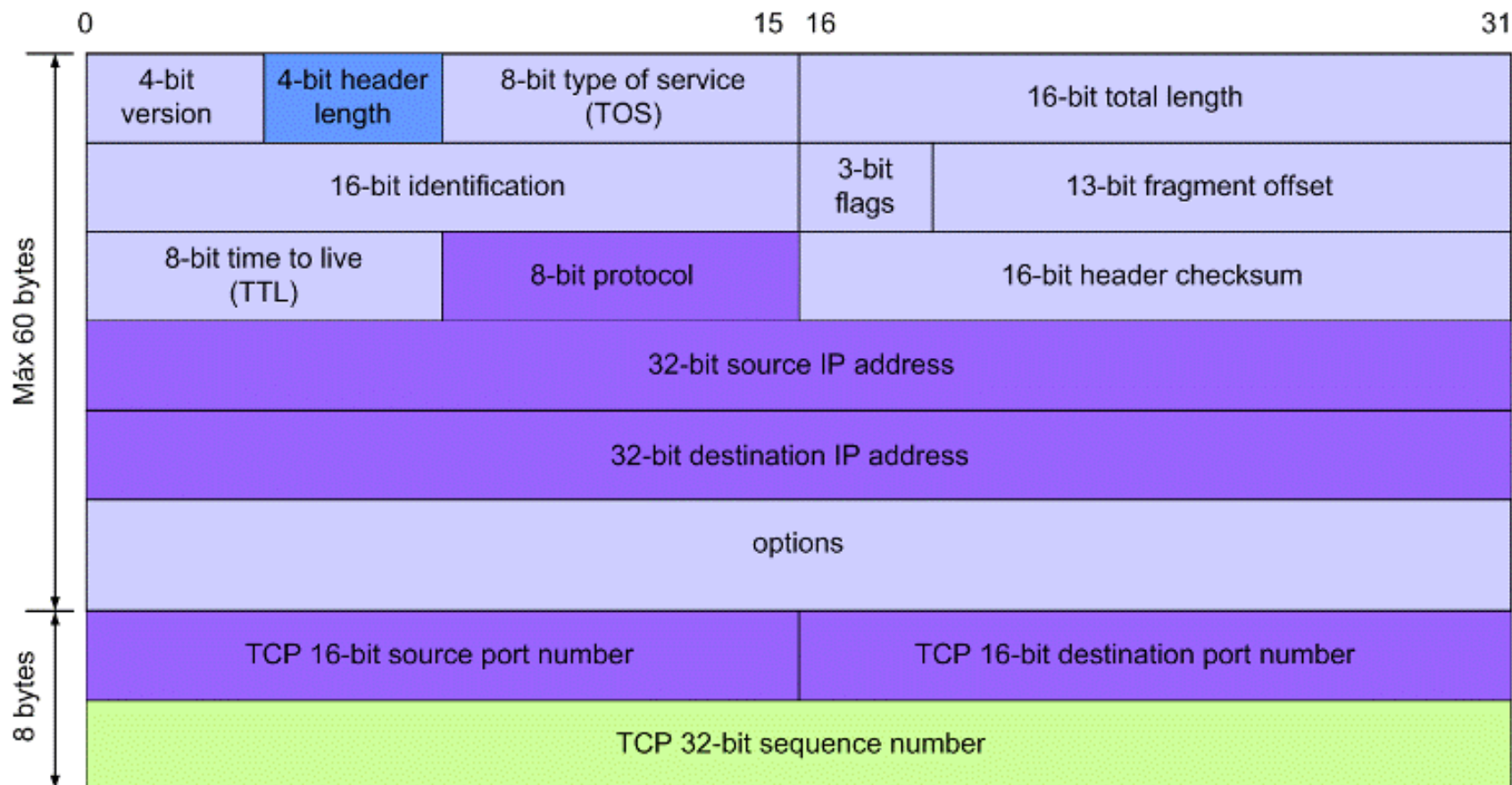
Brief Overview of the IPv4 Situation

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



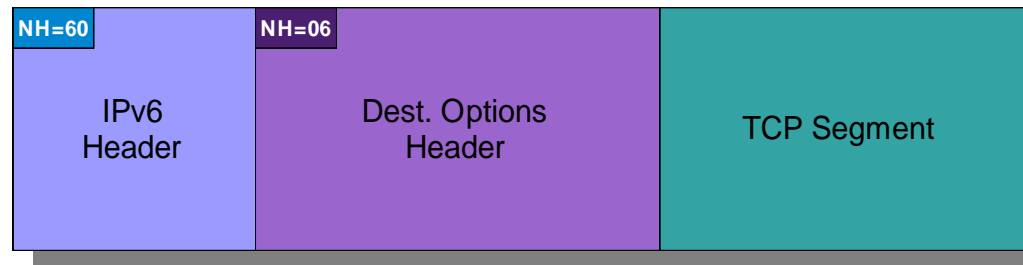
Brief Overview of the IPv4 Situation

- 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:



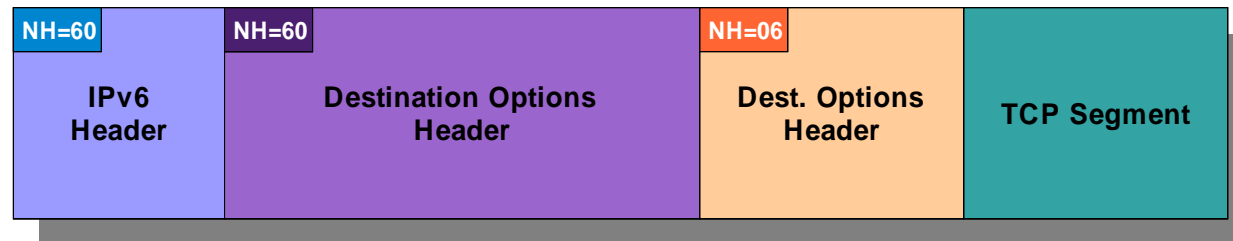
Brief Overview of the IPv6 Situation

- 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,



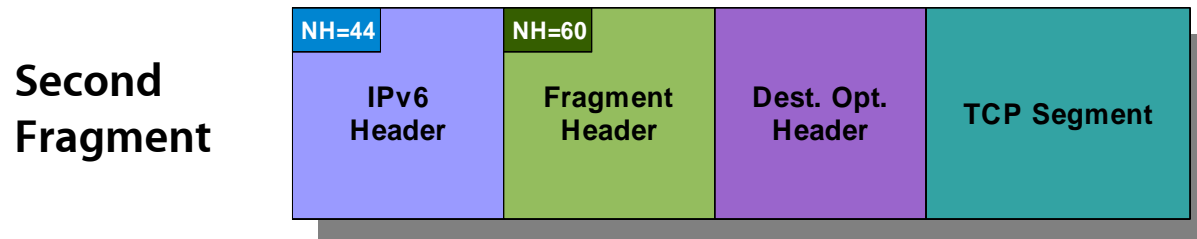
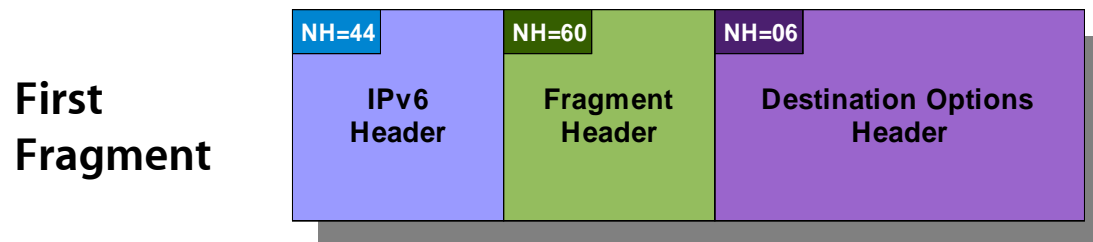
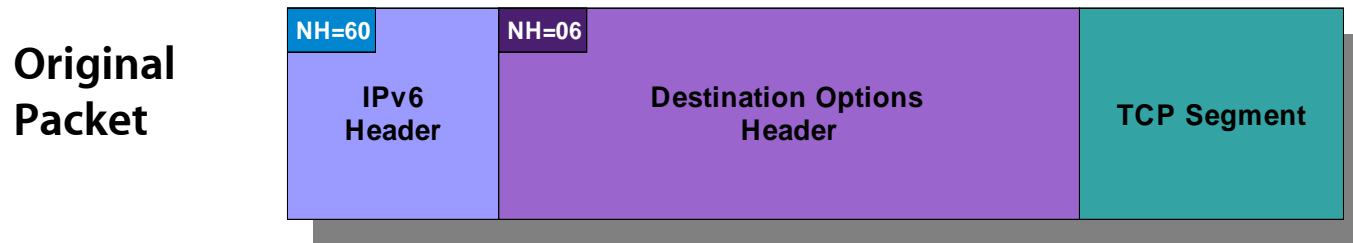
Problem Statement

- 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:



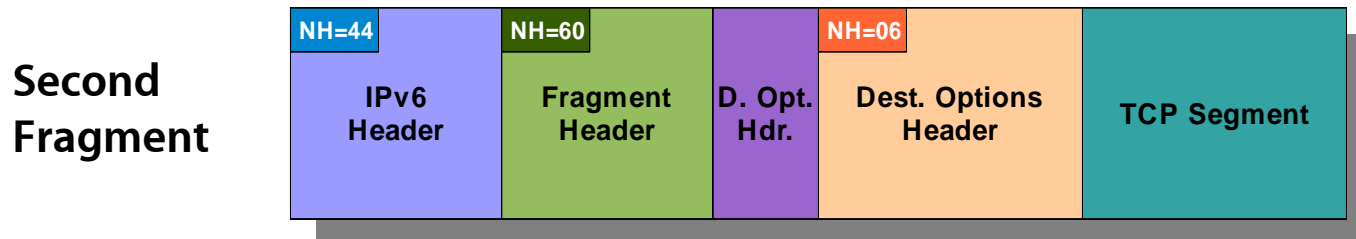
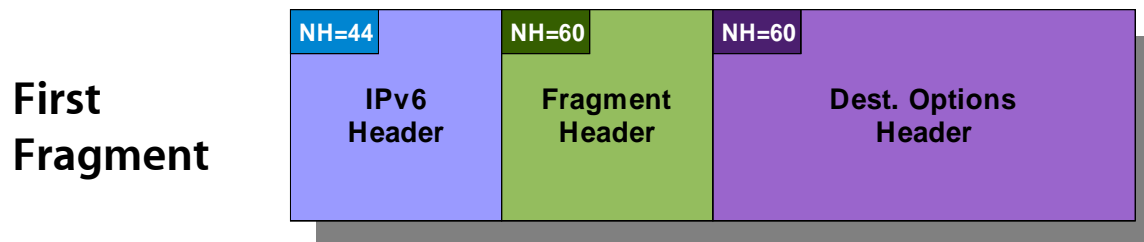
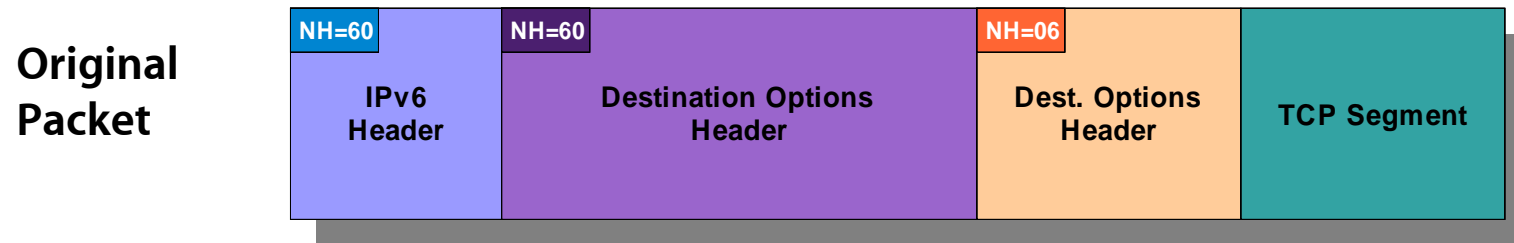
Problem Statement (II)

- Example of Destination Options and Fragmentation:



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
- The possible countermeasures are reduced if filtering is to be performed in layer-2 devices (e.g., RA-Guard)

Some Conclusions

- IPv6 can be easily leveraged for evading firewalls.
- Most likely, firewalls will block packets with extension headersEs muy probable que se haga común el filtrado (en firewalls) de paquetes que contengan en encabezados de extensión
- The result will be: less flexibility, possibly preventing any use of IPv6 extension 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
- Contrary to ICMPv4, 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 Next 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
- BSD-derived and Linux implementations don't – Good! ;-)
- Others?

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:
 - Check the embedded packet for things like TCP Sequence number, etc.
- Anyway, the MTU should not be reduced to a value less than 1280. If a smaller MTU is reported, the receiving node is just required to include a fragmentation header.
- `sysctl`'s (OpenBSD)
 - `net.inet6.icmp6.mtudisc_hiwat` (defaults to 1280): Maximum number of routes created in response to ICMP PTBs
 - `net.inet6.icmp6.mtudisc_lowat` (defaults to 256): Maximum number of routes created in response to (unverified) ICMP PTBs

ICMPv6 Redirects

- ICMP redirects are very similar to the ICMP counterpart, except for:
 - The Hop Limit is required to be 255
- ICMPv6 redirects are an optimization – hence they can be disabled with no interoperability implications
- Whether ICMPv6 are accepted is controlled in *BSD's with the `sysctl net.inet6.icmp6.rediraccept`. In OpenBSD, it defaults to 1 (on).



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
```



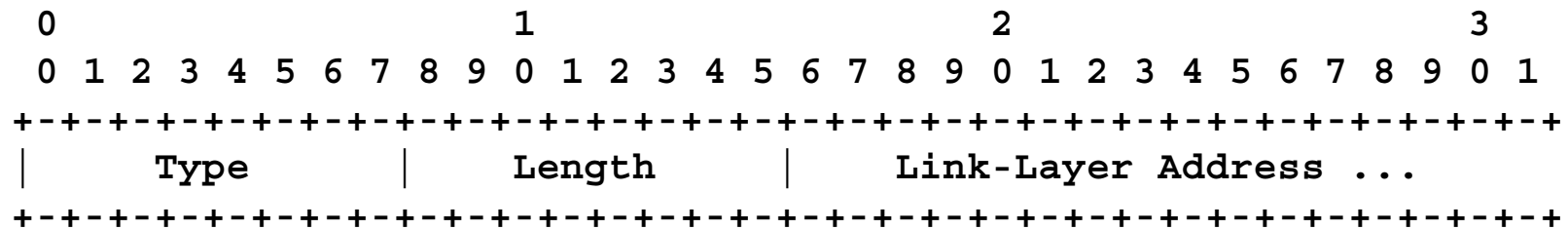
Address Resolution

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 2001:db8::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

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 corresponding 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)
```

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 reachable (probes being sent)

Neighbor Cache (contents)

- 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 (prefixes)

- Sample output of “`ndp -p`” (BSDs):

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

Neighbor Cache (default routers)

- 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
```



Address Resolution sample attacks...

Some Address Resolution Games

- Neighbor Cache Poisoning attacks – the v6 version of V4's ARP cache poisoning
 - The attacker simply listens to Neighbor Solicitations for Target addresses he is interested in, and responds with Neighbor Advertisements that contain his own link-layer address
 - Goal: Denial of Service or “man in the middle”
- Advertising “special” link-layer addresses, e.g.,
 - The broadcast Ethernet address (ff:ff:ff:ff:ff:ff)
 - Multicast Ethernet addresses (e.g., 33:33:00:00:01)
 - The link-layer address of the node sending the Neighbor Solicitation – this introduces a forwarding loop if the victim is a router!
 - All BSD variants tested don't check for these special addresses!

Overflowing the Neighbor Cache

- Some implementations (FreeBSD, NetBSD) don't enforce limits on the number of entries that can be created in the Neighbor Cache
- 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)

```
fe80::ffe8:2ac9:770c:f3b0%fxp0      90:4:fd:77:d2:18      fxp0 23h57m1s S
fe80::ffe8:63e6:15c6:35f9%fxp0      90:4:fd:77:d2:18      fxp0 23h56m54s S
fe80::ffe8:719d:8e8b:3a01%fxp0      90:4:fd:77:d2:18      fxp0 23h57m3s S
fe80::ffe8:aa8d:6d2b:c0e%fxp0        90:4:fd:77:d2:18      fxp0 23h54m31s S
fe80::ffe9:c8a:2c84:a151%fxp0        90:4:fd:77:d2:18      fxp0 23h58m48s S
fe80::ffeb:1563:3e7f:408a%fxp0       90:4:fd:77:d2:18      fxp0 23h56m39s S
fe80::ffec:b12e:9e2c:79%fxp0         90:4:fd:77:d2:18      fxp0 23h56m1s S
fe80::fff0:423a:6566:798a%fxp0       90:4:fd:77:d2:18      fxp0 23h58m42s S
fe80::fff0:eb27:f581:1ce5%fxp0       90:4:fd:77:d2:18      fxp0 23h56m5s S
fe80::fff3:4875:3a14:c26c%fxp0       90:4:fd:77:d2:18      fxp0 23h53m58s S
fe80::fff7:8e67:24c2:9cc1%fxp0       90:4:fd:77:d2:18      fxp0 23h54m3s S
fe80::fff8:3f:bef2:211%fxp0          90:4:fd:77:d2:18      fxp0 23h55m56s S
fe80::fff9:ca73:d351:4057%fxp0       90:4:fd:77:d2:18      fxp0 23h56m32s S
fe80::fffb:ae1b:90ef:7fc3%fxp0       90:4:fd:77:d2:18      fxp0 23h55m16s S
fe80::fffc:bffb:658f:58e8%fxp0       90:4:fd:77:d2:18      fxp0 23h59m22s S
fe80::1%lo0                          (incomplete)          lo0 permanent R
#      nd6_storelladdr: something odd happens
nd6_storelladdr: something odd happens
panic: knem_malloc(4096): knem_map too small: 40497152 total allocated
Uptime: 4h14m51s
Cannot dump. No dump device defined.
Automatic reboot in 15 seconds - press a key on the console to abort
--> Press a key on the console to reboot,
--> or switch off the system now.
```

“Man in the Middle” or Denial of Service

- 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.

Some Address Resolution Games

- Neighbor Cache Poisoning attacks – the v6 version of V4's ARP cache poisoning
 - The attacker simply listens to Neighbor Solicitations for Target addresses he is interested in, and responds with Neighbor Advertisements that contain his own link-layer address
- Advertising “special” link-layer addresses, e.g.,
 - The broadcast Ethernet address (ff:ff:ff:ff:ff:ff)
 - Multicast Ethernet addresses (e.g., 33:33:00:00:01)
 - The link-layer address of the node sending the Neighbor Solicitation – this introduces a forwarding loop if the victim is a router!
 - All BSD variants tested don't check for these special addresses!
- Not much support in layer-2 security boxes to mitigate these attacks
- Open source tools do exist. E.g., NDPMon, available at:
<http://ndpmon.sourceforge.net>

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.

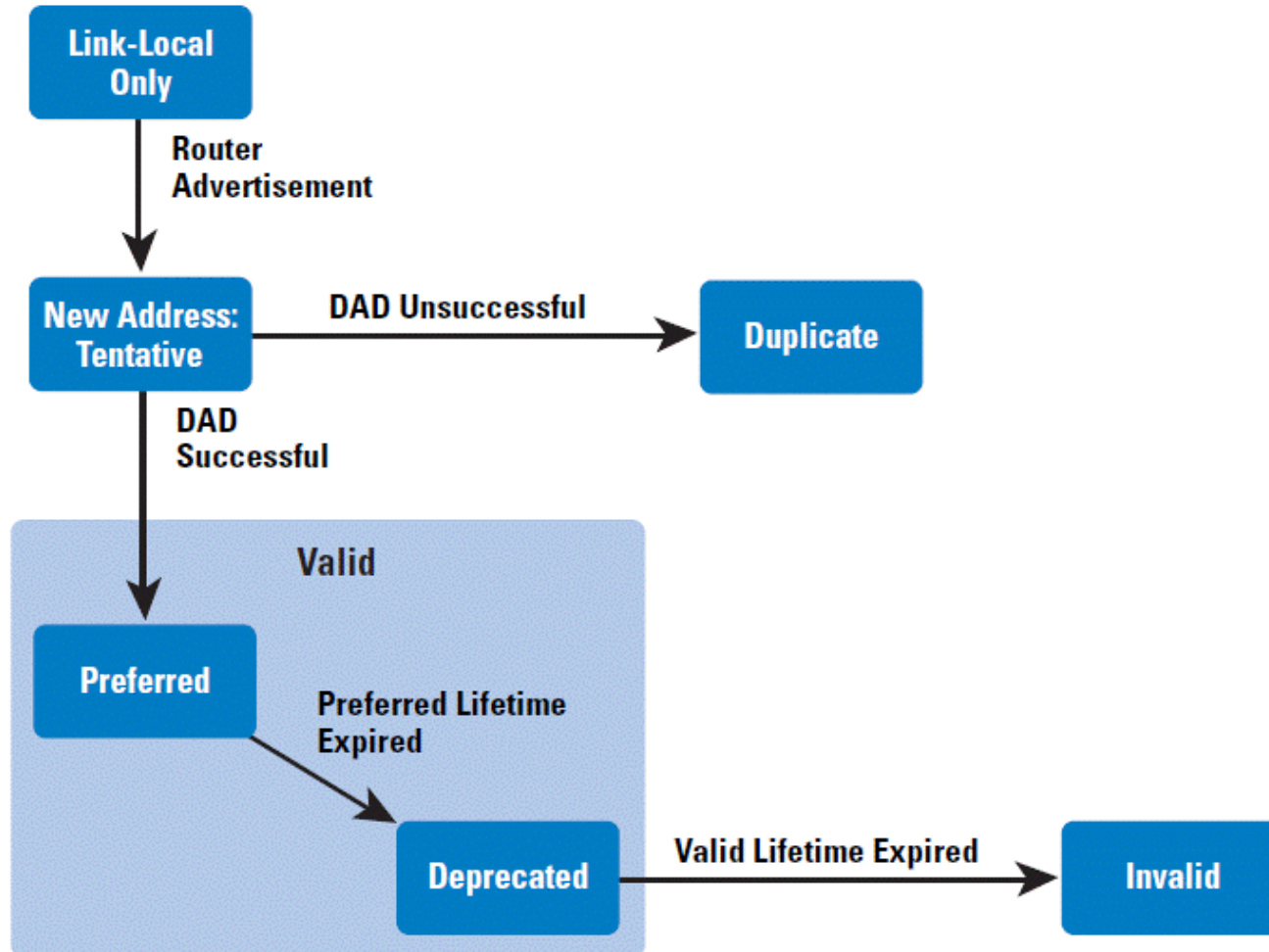


IPv6 Stateless Address Autoconfiguration (SLAAC)

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
 3. The host sends a Router Solicitation message
 4. When a Router Advertisement is received, it configures a “tentative” IPv6 address
 5. 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
 6. If the address is unique, it typically becomes a “preferred” address

Address Autoconfiguration flowchart



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

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 0xffffffff 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
```



IPv6 SLAAC

some sample attacks...



Disable an Existing Router

- Forge a Router Advertisement message that impersonates 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 result, 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.
- A very small MTU could lead to an increase of the header/data ratio, and possibly to DoS if the victim fails to validate the advertised MTU
- A very small Current Hop Limit would packets to be discarded by the intervening routers



IPv6 SLAAC

Some sysctl's...

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.

Autoconf Addresses & Privacy

- Addresses selected as part of stateless autoconfiguration contain a modified version of the MAC address of the interface
- The MAC address is globally-unique, and non-changing (OUI assigned by the IEEE to the vendor, plus a 3-byte number selected by the vendor)
- There were concerns that autoconf addresses hurt privacy, as they could be used to correlate network activity
- Privacy addresses (RFC 4941) were introduced for that purpose
 - They basically set the Interface ID to a random number, and are short
 - They are short-lived
 - They tend to be painful for the purpose of logging

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 "connections" should use privacy addresses)



IPv6 SLAAC

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

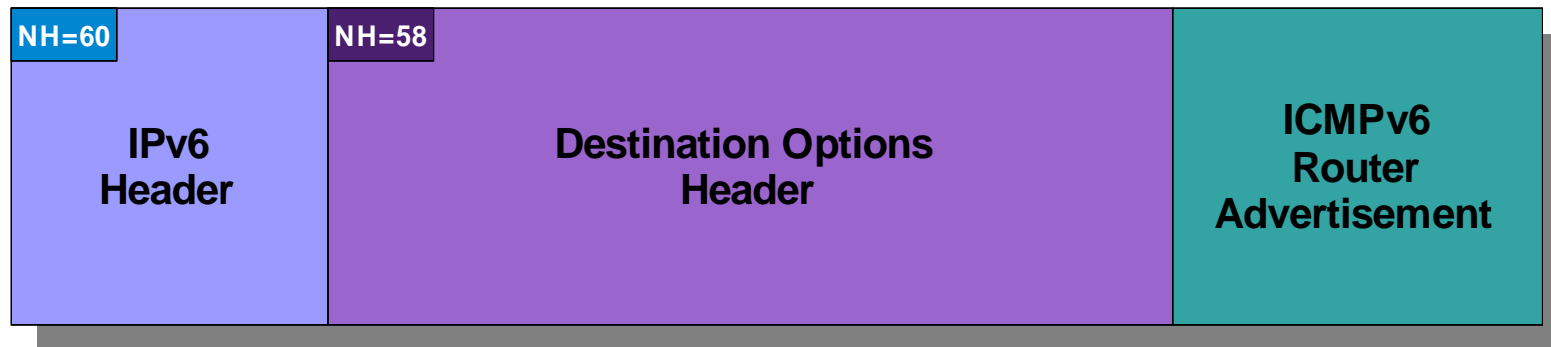


IPv6 SLAAC

RA-Guard evasion

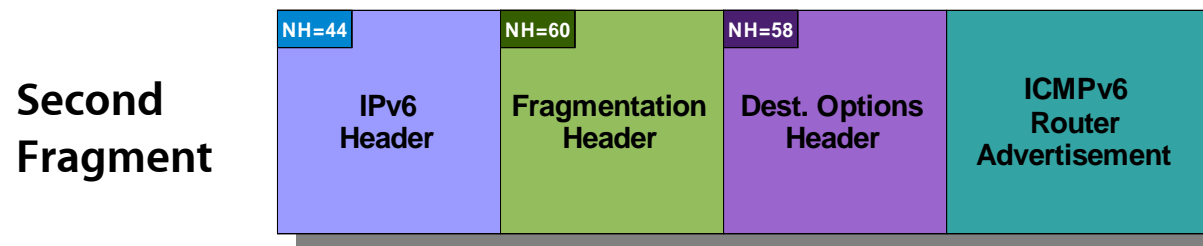
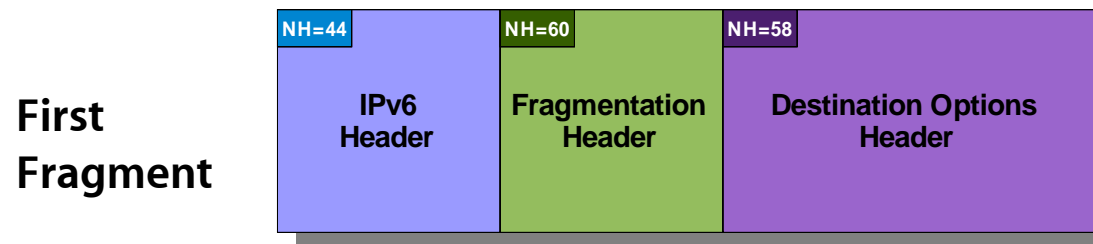
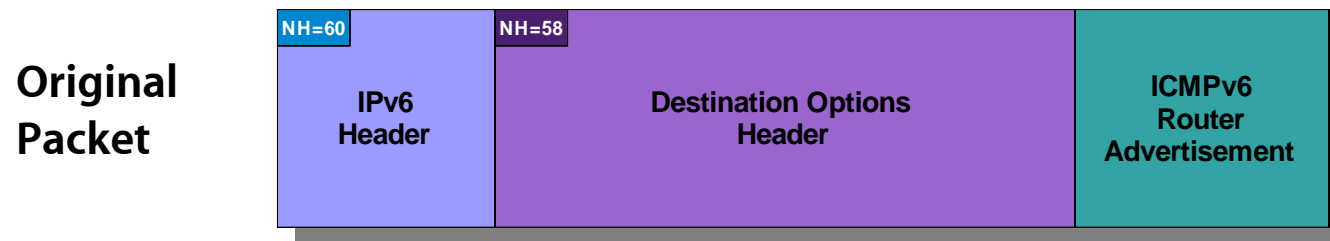
Problem Statement

- As noted before, 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.
- For example,



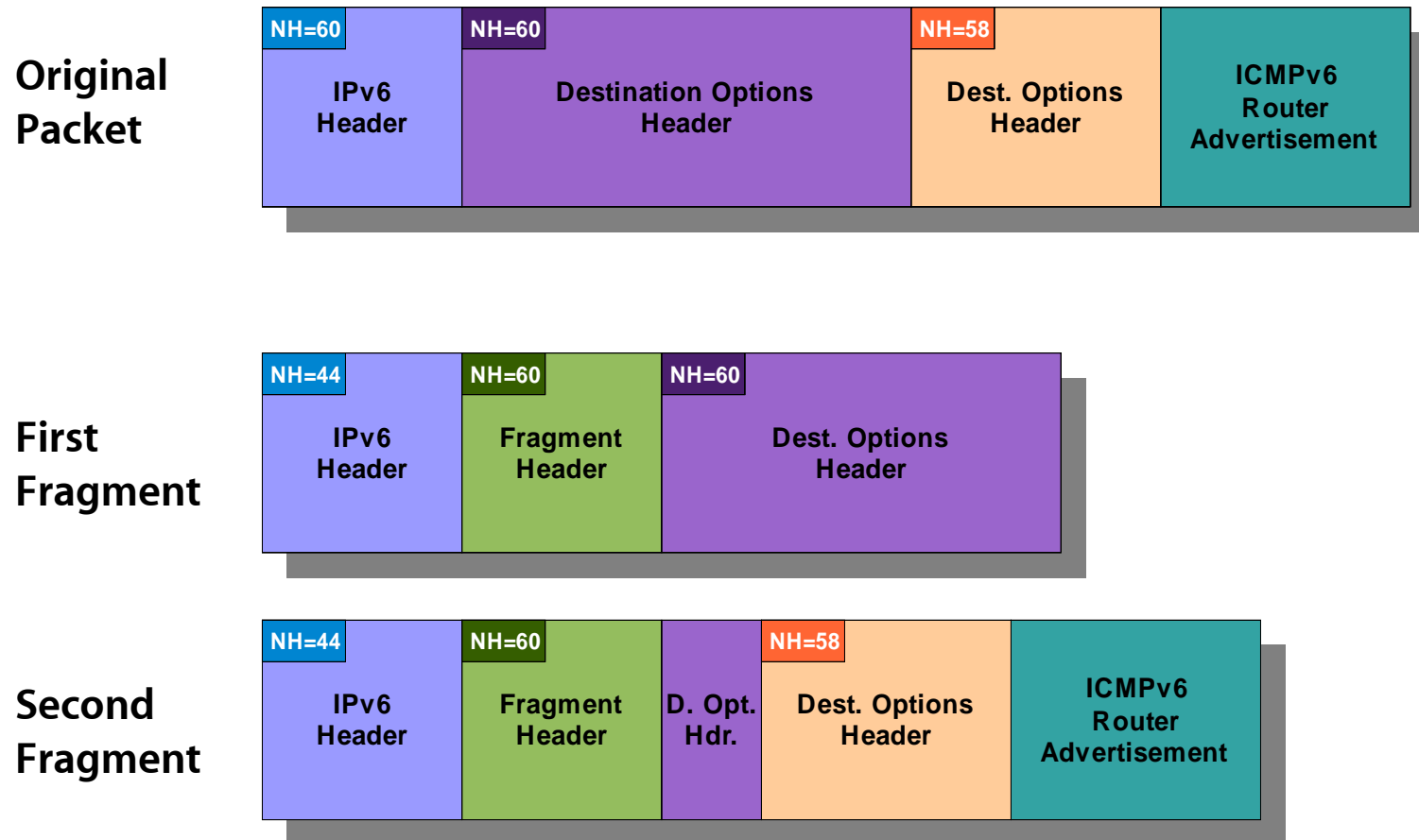
Problem Statement (II)

- Combination of a Destination Options header and fragmentation:



Problem Statement (III)

- Two Destination Options headers, and fragmentation:



Some Conclusions

- 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>



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 this information to decide which packets must be forwarded to the local segment.
- Since Neighbor Discovery uses multicast addresses (the solicited-node multicast address), MLD is used by all IPv6 nodes
- In practice, the only use for MLD with Neighbor Discovery is MLD-snooping switches – switches that interpret MLD packets 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

- IPsec support is currently mandatory for IPv6 implementations – the IETF is changing this requirement to “optional” thus acknowledging reality.
- Anyway, in practice this is irrelevant:
 - What was mandatory was IPsec *support* – not IPsec *use*.
 - Also, many IPv4 implementations support IPsec, while many IPv6 implementations do not.
- Most of the key problems (e.g., PKI) for IPsec deployment in IPv4 apply to IPv6, as well.
- There is no reason to believe that IPv6 will result in an increased use of IPsec.



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



IPv6 Transition Co-Existence Technologies



IPv6 Transition/Co-existence Technologies

- IPv6 is not backwards-compatible with IPv4
- 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*
- Current transition/co-existence plan: based on a toolbox:
 - dual-stack
 - tunnels
 - translation



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 “become” dual-stack
- 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/>



Transition Technologies

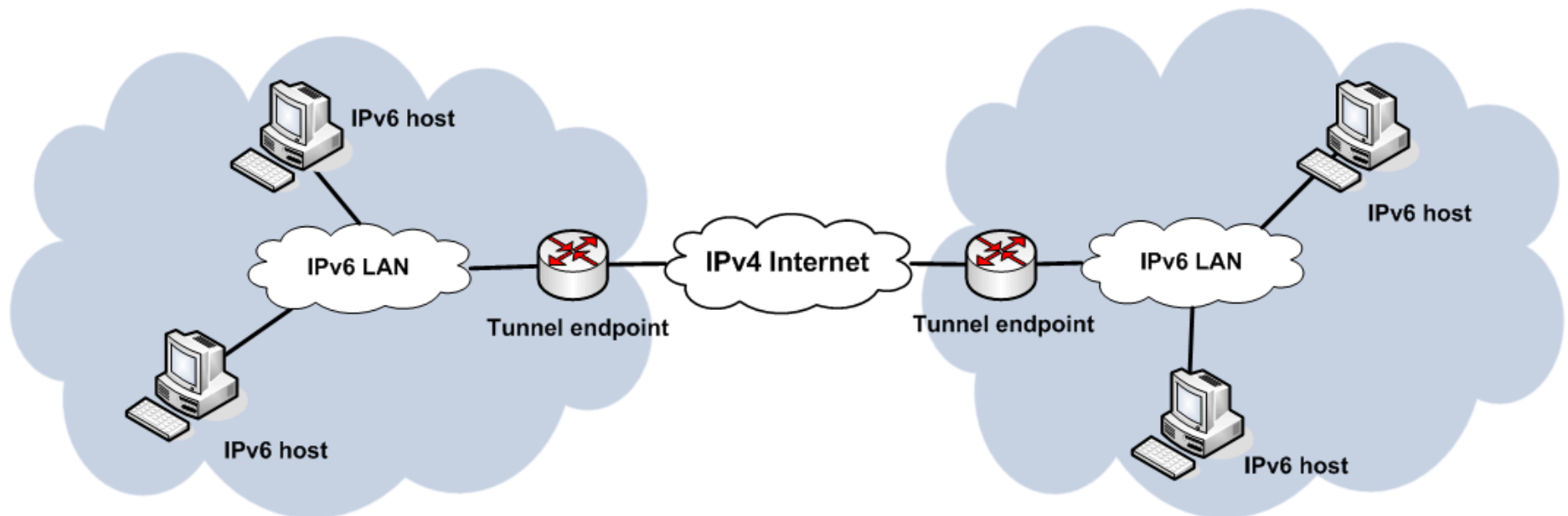
Tunnels

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

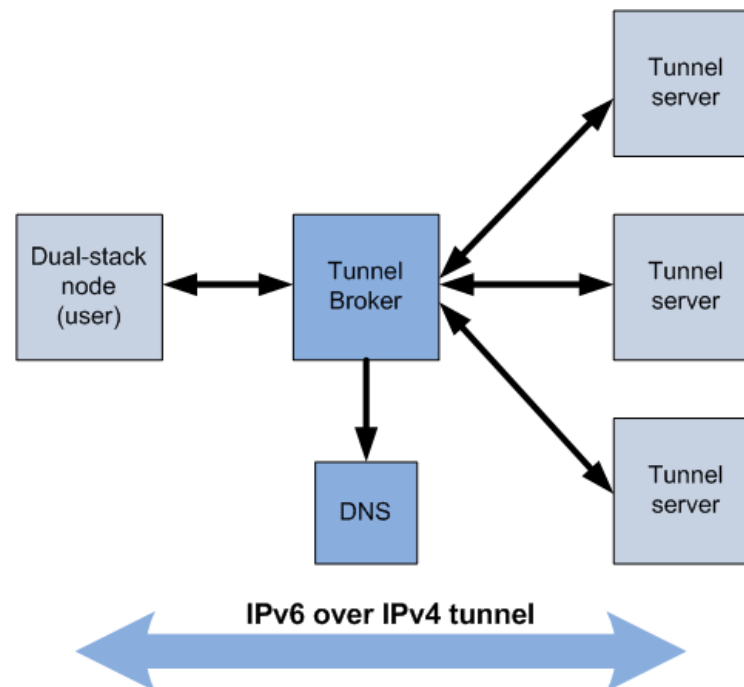
6in4

- 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)



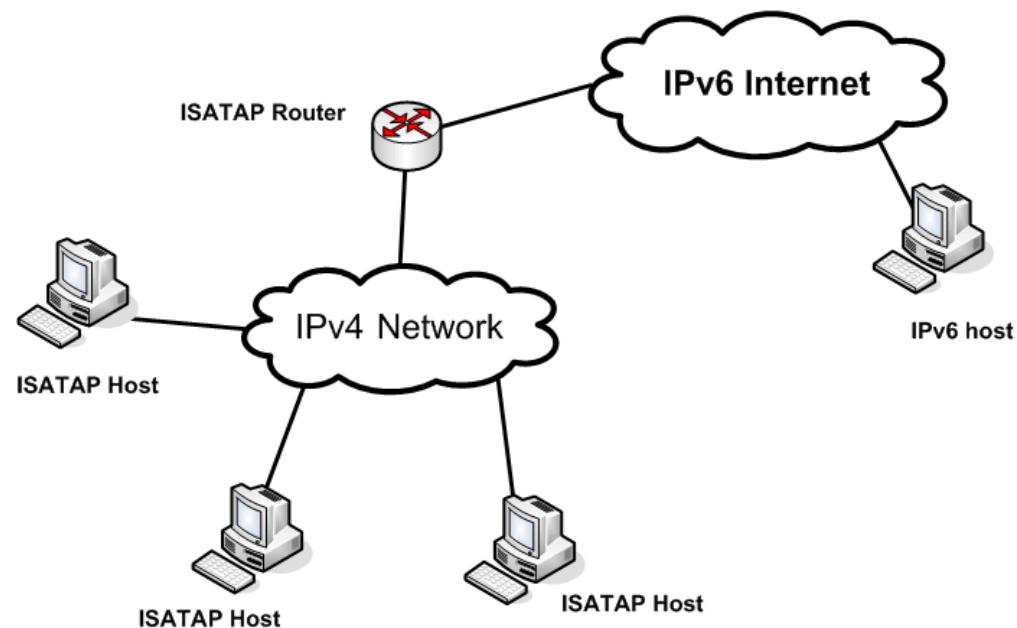


Tunnel Broker: Sample Implementation

- 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

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



Interface-ID format

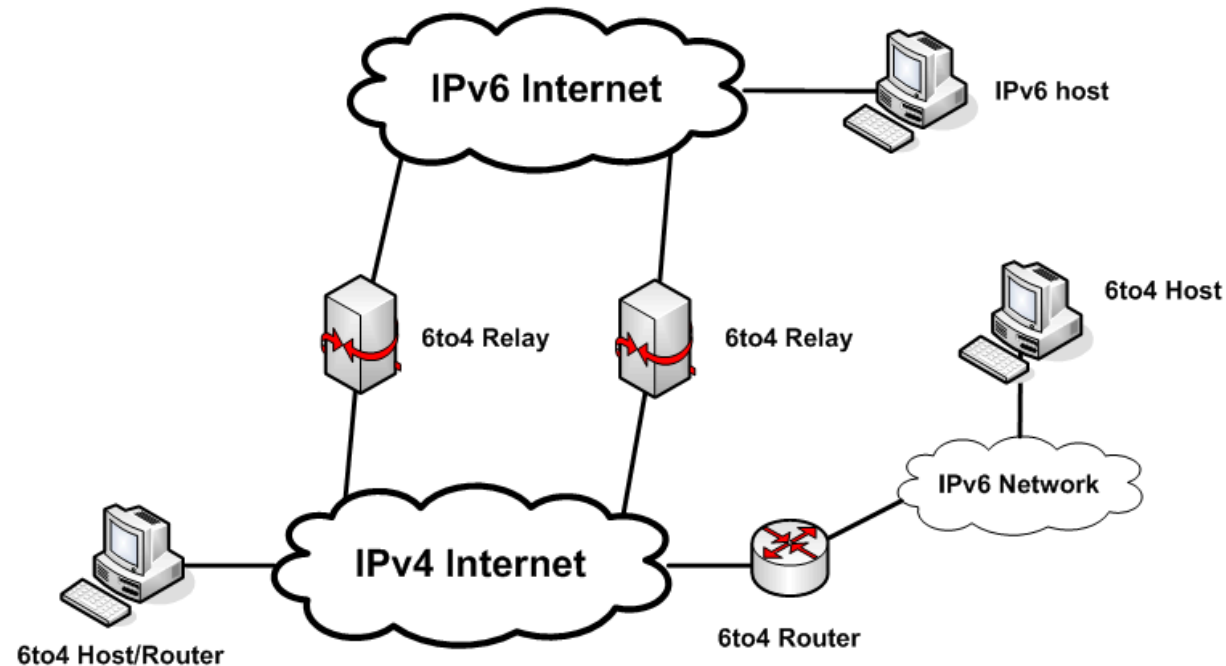
0	1 1	3 3	6
0	5 6	1 2	3
+-----+-----+-----+			
000000ug00000000	0101111011111110	IPv4 address	
+-----+-----+-----+			

Exploiting 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

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



IPv6 Address format

16	32	16	64 bits
2002	V4ADDR	Subnet	Interface ID

6to4 (II)

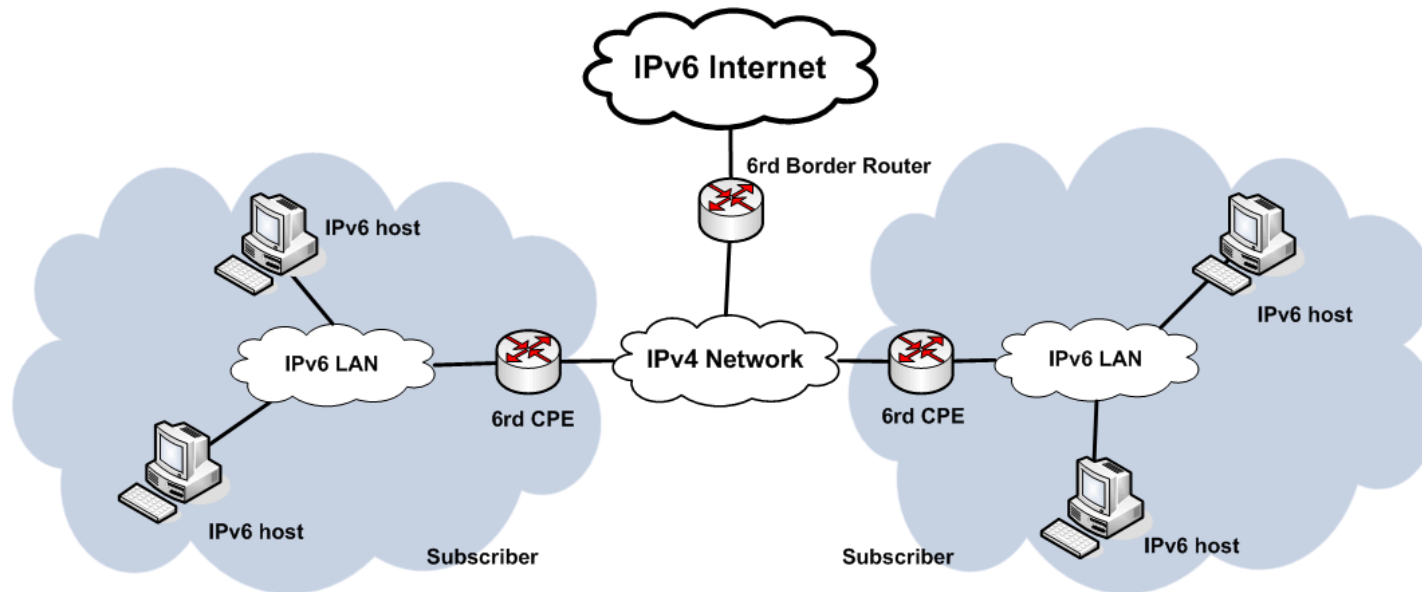
- Packets originate at a 6to4 host as native IPv6 packets
- A 6to4 router encapsulates the packet in IPv4, and sets the IPv4 Destination Address to that of a 6to4 relay (or the corresponding anycast address 192.88.99.1)
- The router decapsulates the IPv6 packet and forwards it to the IPv6 Internet
- Packets destined from a native IPv6 host to a 6to4 host are routed towards a relay (i.e., peers advertising the 6to4 IPv6 prefix)
- The relay encapsulates the packet into IPv4, and sends it to the 6to4 router
- The 6to4 router decapsulates the IPv6 packets, and forwards it to the “local” IPv6 network
- Packets from 6to4 hosts to 6to4 hosts do not enter the IPv6 Internet (the source 6to4 router sends the packet directly to the destination 6to4 router)

Problems with 6to4

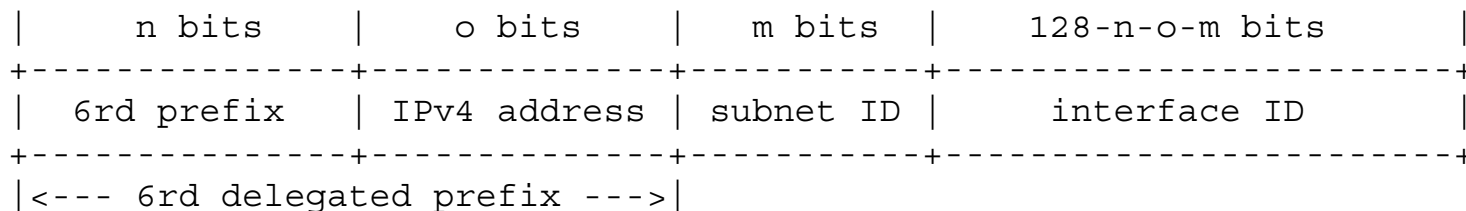
- Lots of poorly-managed 6to4 relays have been deployed
- In most cases they introduce PMTUD black-holes (e.g. as a result of ICMPv6 rate-limiting)
- Lack of control of which 6to4 relays are used make troubleshooting difficult
 - Use of the 6to4 anycast address makes it difficult to identify a poorly-managed 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 (IPv6 rapid deployment)

- Aims at enabling IPv6 deployment in a site with no IPv6 infrastructure
- Builds upon 6to4 – but the whole system is implemented within a site
- No special prefix – uses global unicast range

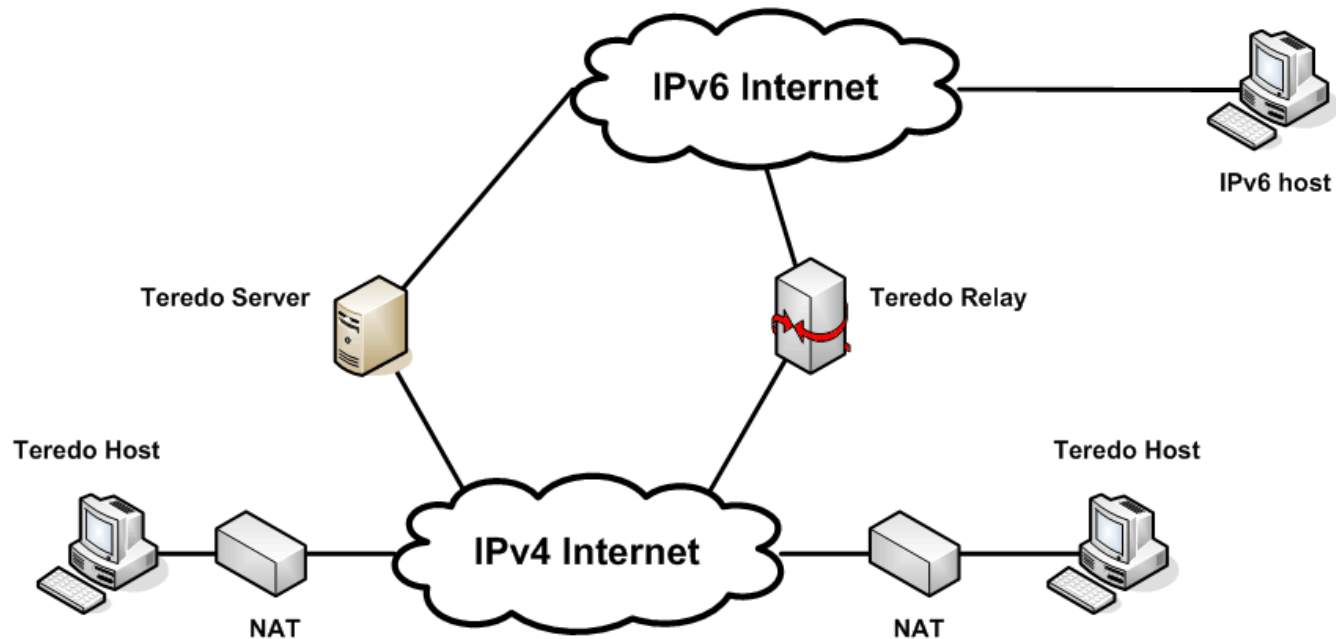


Address format



Teredo

- Aims at providing IPv6 connectivity to individual hosts behind one or more NATs -- “last resort” mechanism for IPv6 connectivity
- Suffers some of the same problems as 6to4



Teredo Address format

32	32	16	16	32
Teredo Pref	Server IPv4	Flags	Port	Client IPv4

Security 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 server 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

Translation

- All of the previous transition/co-existence technologies require assignment of both IPv4 and IPv6 addresses – what if there are no IPv4 addresses left?
- A number of technologies are currently being developed in the IETF such that:
 - IPv4 addresses can be dynamically shared by a large number of hosts, or,
 - IPv6-only nodes can still access IPv4-only nodes
- Among these technologies are:
 - CGN (Carrier-Grade NAT)
 - NAT 64
 - A+P

The future doesn't look like very NAT-free.....



Security Implications of IPv6 on IPv4 Networks



Security Implications on IPv4 Networks

Transition Technologies



Exploiting Transition Technologies

- Some systems (notably Windows) have support of transition technologies enabled by default.
- These technologies could be used to circumvent security controls.
- Technologies such as Teredo could increase the attack exposure 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

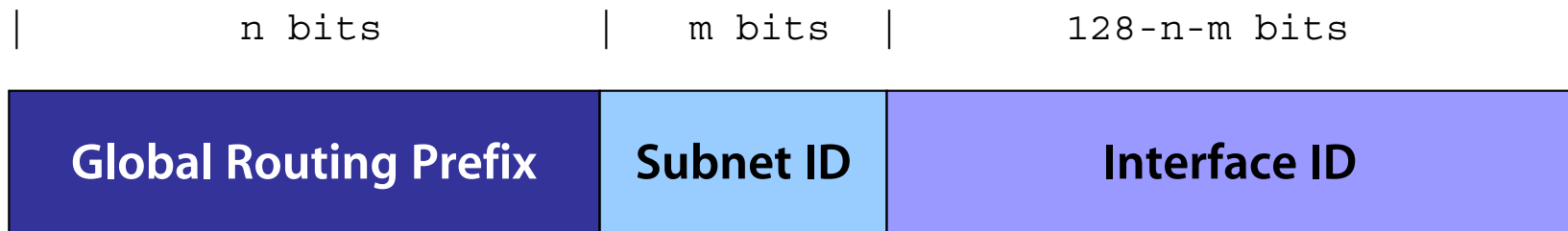


Network Reconnaissance

Implications IPv6 addressing on scanning

Global Unicast Addresses

- Syntax of the global unicast addresses:



- The interface ID is typically 64-bis
- Global Unicast Addresses can be generated with multiple 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

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: address 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 addresses 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	
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.

Some Advice

- In general, a node does not need to be “publicly reachable” (e.g., servers), privacy 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:
 - Some OSes implement the privacy extensions specified in RFC 4941
 - Others generate the Interface-ID as a result of a hash-function over (Prefix, MAC address, secret)
- 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

Possible approaches



Leveraging IPv6 features

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



Application-layer protocols

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

DNS

- IPv6 addresses can be obtained by querying the DNS for AAAA records.
- Many sites currently use domain names such as "ipv6*"
- E.g., Google for "site:ipv6*" and "site:ip6*"



Network “Neighborhood” protocols

- mDNS is being increasingly 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 occasional networking.
- This provides yet another vector for network reconnaissance.



Some thoughts on IPv6 security



Some thoughts...

- While IPv6 provides similar features than IPv4, it uses different mechanisms. – and the evil lies 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?



Acknowledgments

- Hack in Paris 2011 organizers

Fernando Gont

fernando@gont.com.ar

<http://www.gont.com.ar>