Copyright notice

This slideware is © 2012 by SI6 Networks

Permission to use this slideware is granted **only** for **personal use**

Permission to distribute this slideware is granted **only without any modifications to the slides set**

For any other uses, please contact:

info@si6networks.com

www.si6networks.com

Contents

This slideware contains **part** of the materials used for the training "**Hacking IPv6 Networks**" during the BRUCON 2012 conference

More information available at:

www.hackingipv6networks.com

www.si6networks.com

Hacking IPv6 Networks Training

Fernando Gont

BRUCON 2012
Ghent, Belgium. September 24-25, 2012

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 as a security researcher 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

- Brief introduction to IPv6, and current state of affairs
- Objectives of this training
- 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

Agenda (II)

- IPv6 Address Resolution
- Stateless Address Auto-configuration (SLAAC)
- 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.)

Agenda (III)

- 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

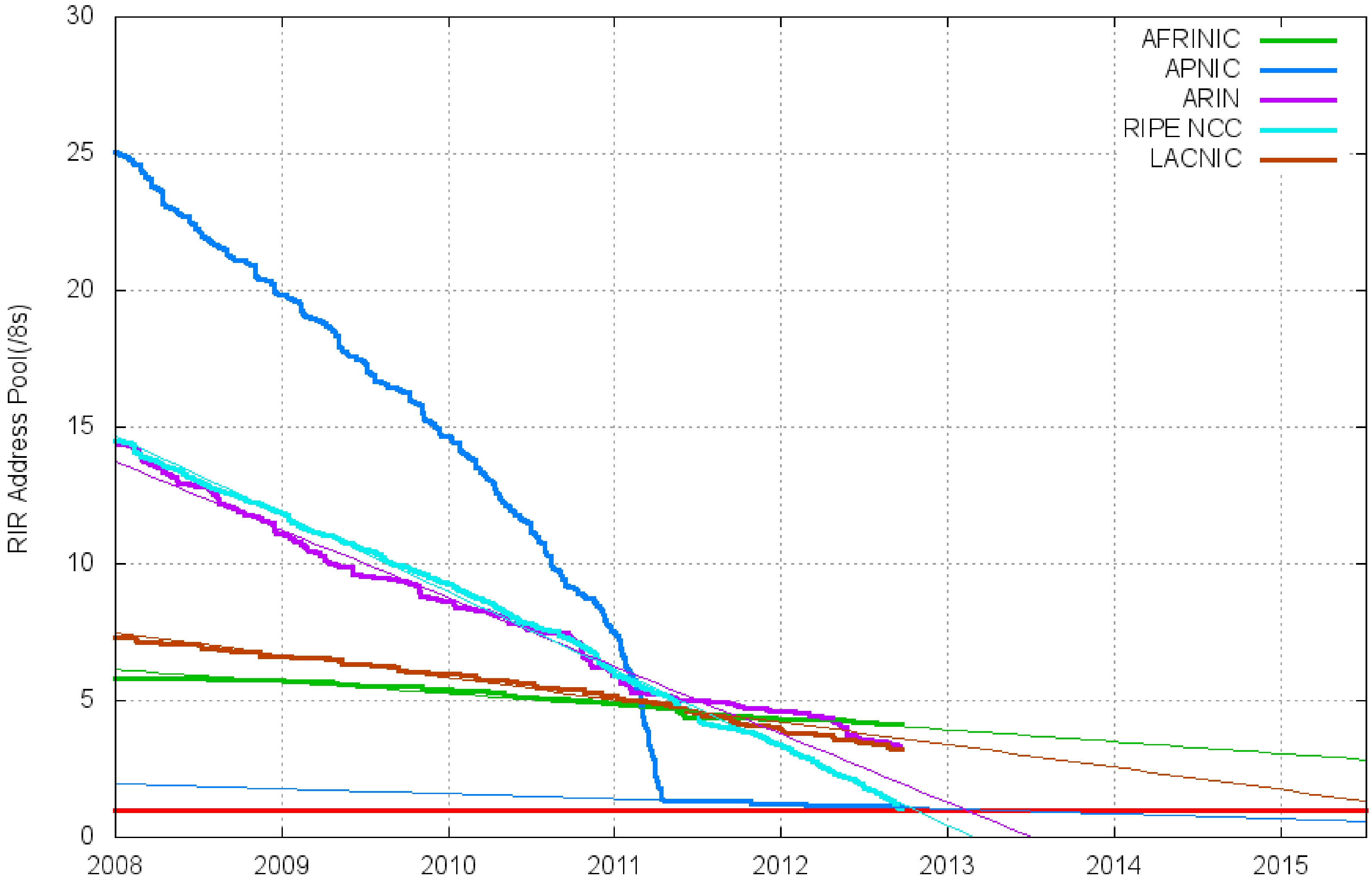
Brief introduction to IPv6

IPv4 address exhaustion

- The Internet relies on unique addresses for host communication
- More than 20 years ago it was already evident we'd eventually run out of IPv4 addresses
- Network address translators (NATs) have served as a stop-gap
- But nevertheless we're hitting IPv4 address exhaustion

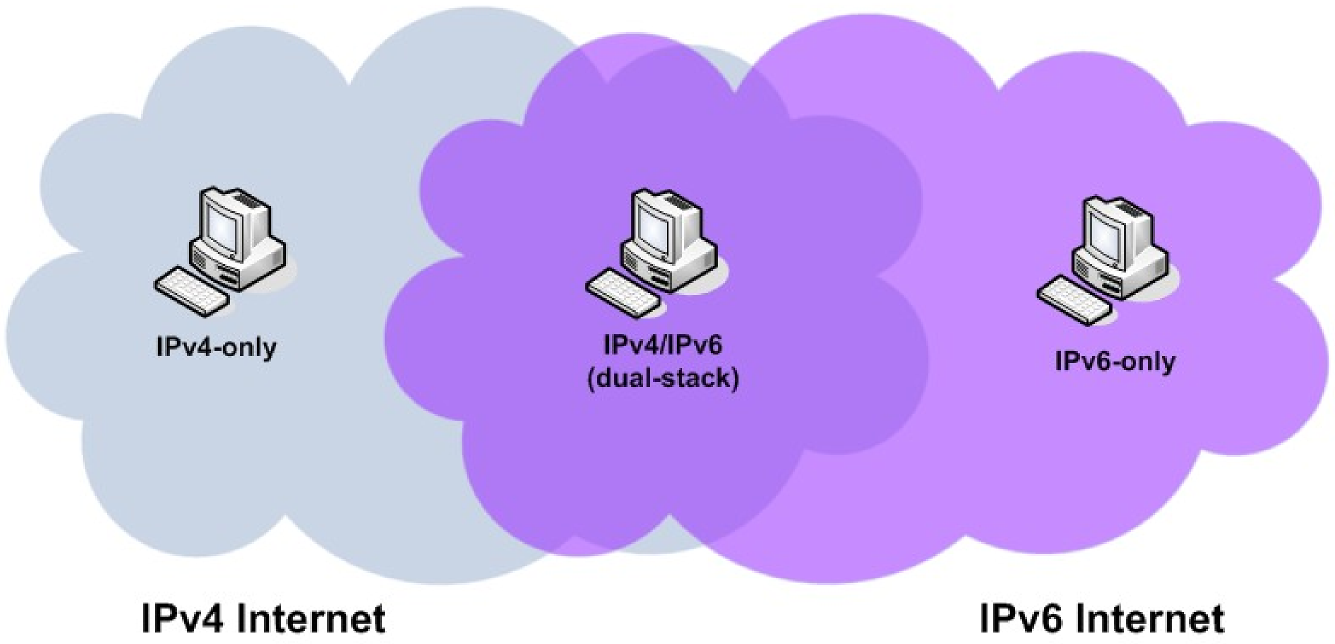
IPv4 address exhaustion (II)

- IPv4 address exhaustion, as predicted by Geoff Huston



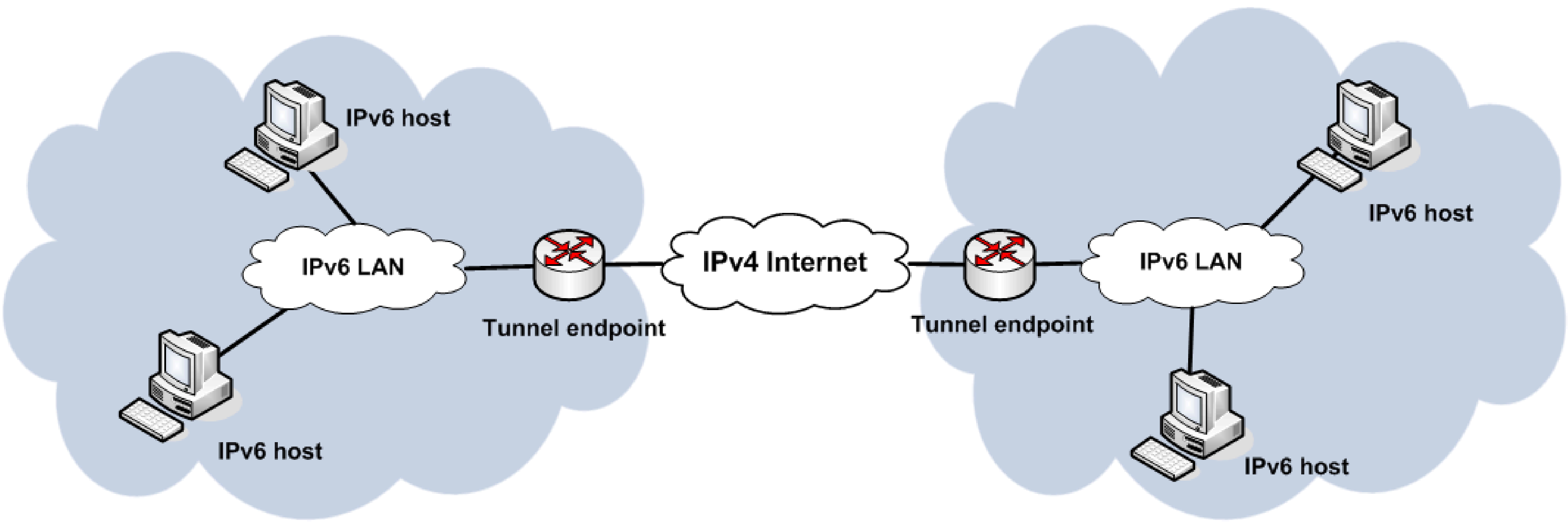
So... what is this “IPv6” thing about?

- It addresses the problem of IPv4 address exhaustion
- Employs 128-bit addresses (vs. IPv4's 32-bit addresses)
- Provides the same **service** as IPv4
- It is **not backwards-compatible with IPv4**



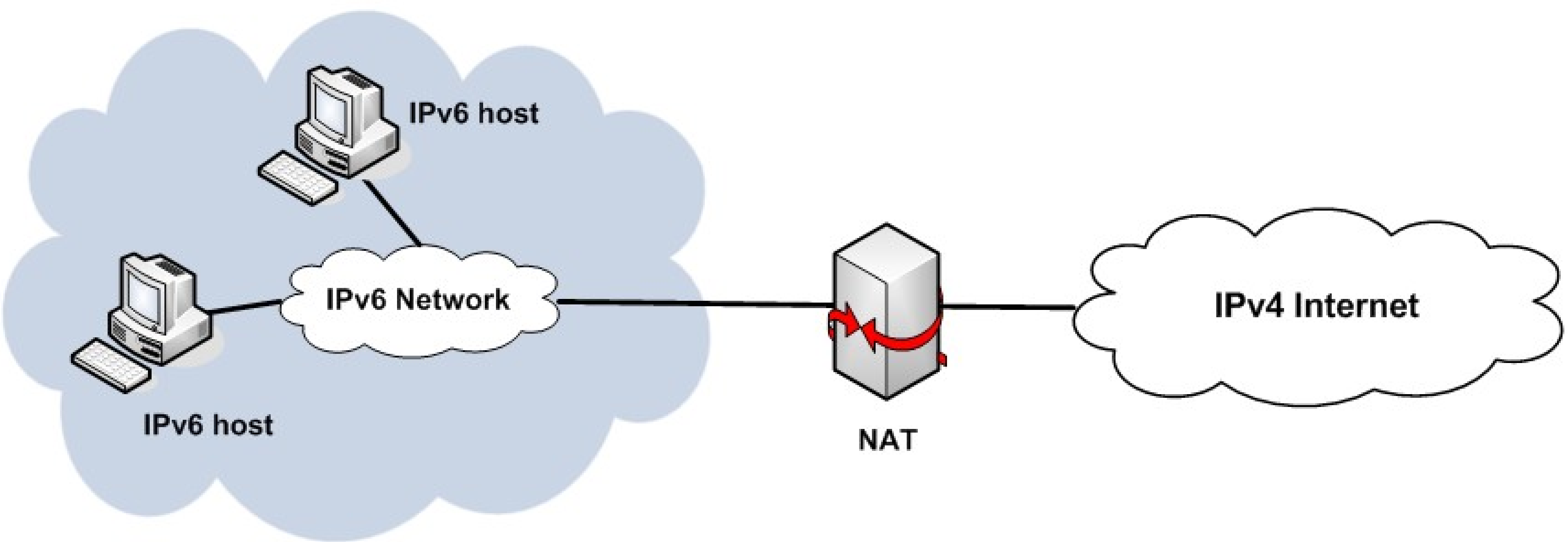
So... what is this "IPv6" thing about? (II)

- We can connect IPv6 islands across the IPv4 Internet with tunnels



So... what is this “IPv6” thing about? (III)

- We can interconnect IPv6-only hosts with IPv4-only hosts with “translators”



So... what is this “IPv6” thing about? (IV)

- For every domain name, the DNS can contain
  - A resource records (IPv4 addresses)
  - AAAA (Quad-A) resource records (IPv6 addresses)
- Host may query for A and/or AAAA resource records according different criteria
- Based on the available resource records, supported protocols, and local policy, IPv6 and/or IPv4 could be employed

Current state of affairs: Implementation

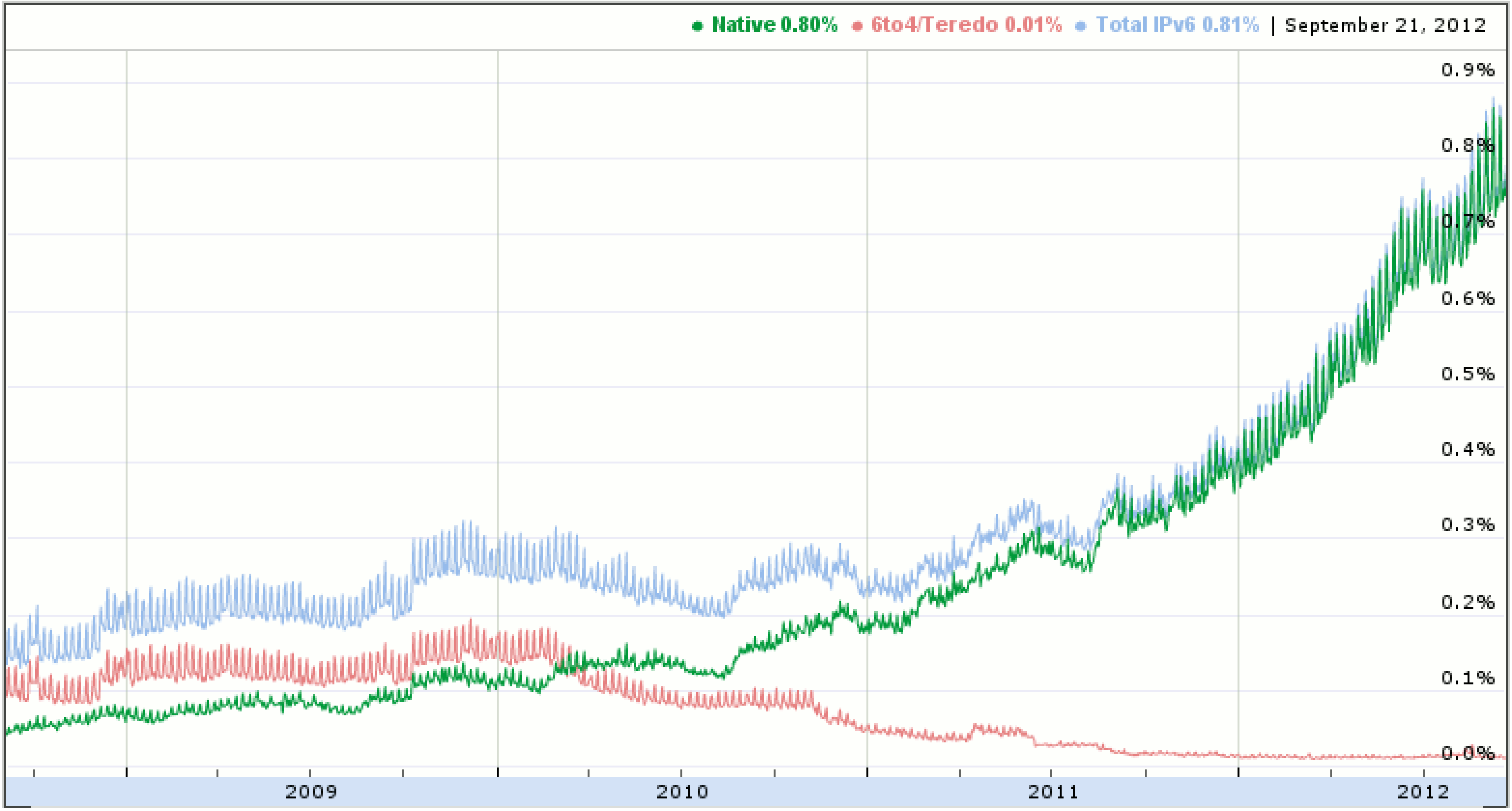
- General-purpose OSes have shipped with IPv6 support for a long time
  - part of your network is already running IPv6!
- Other devices may require updates or replacement:
  - CPE's
  - Firewalls
  - Routers
  - NIDSs
  - etc.

Current state of affairs: Deployment

- IPv6 was essentially **ignored for years**
- Many organizations have now started to take IPv6 more seriously, partly as a result of:
  - Exhaustion of the IANA IPv4 free pool
  - Imminent exhaustion of the address pool at the different RIRs
  - Awareness activities (“World IPv6 Day” & “World IPv6 Launch Day”)

Current state of affairs: Deployment (II)

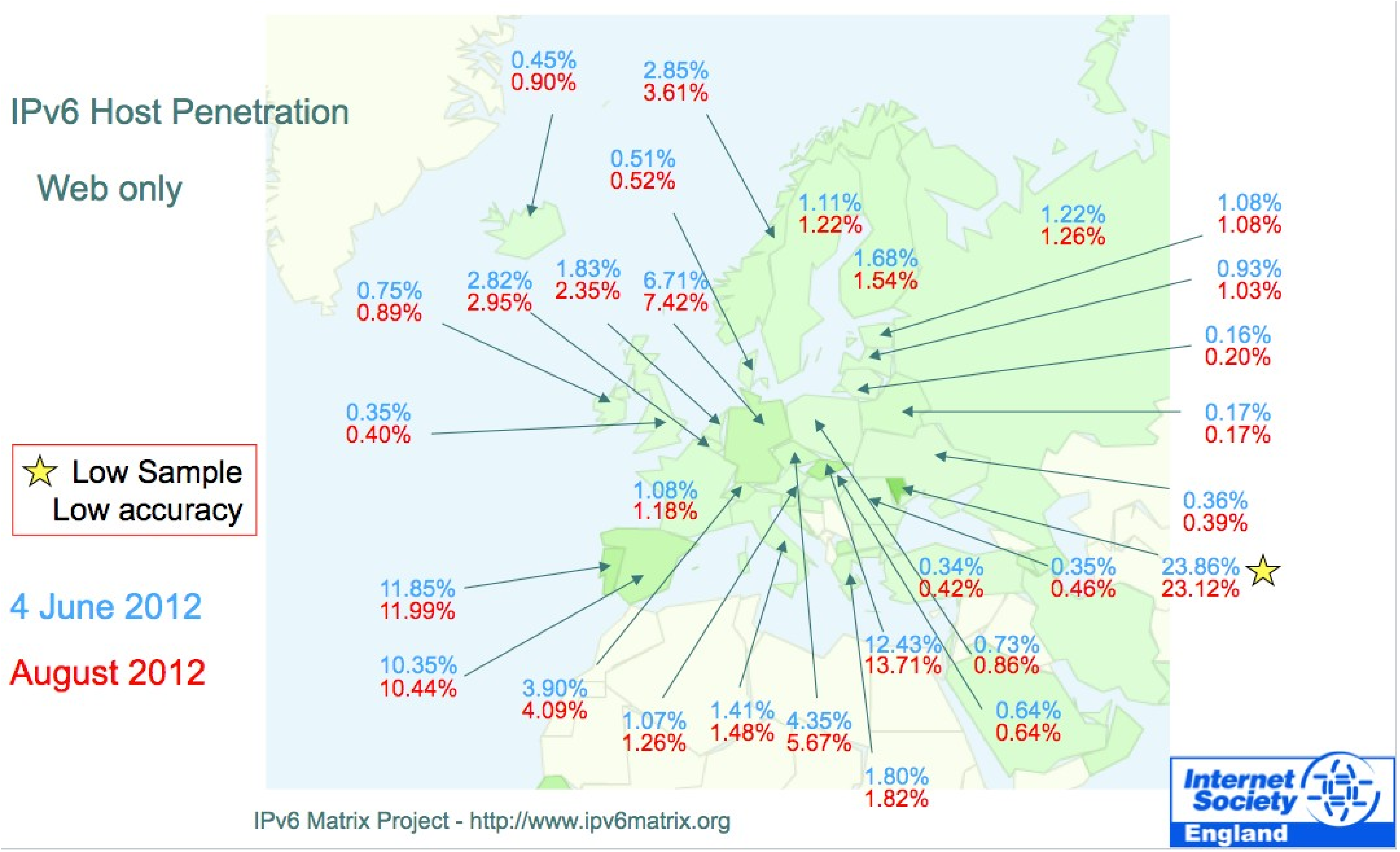
- IPv6 usage as measured by Google:



SI6 NETWORKS

Current state of affairs: Deployment (III)

- IPv6-enabled web sites (as measured by ISOC England):



Motivation for this IPv6 training

Why should I care about IPv6 security?

- Most networks have already deployed IPv6! (partially)
- You will likely perform a full-deployment in the short or near term
- Even if you **really** run an IPv4-only network, you may communicate with IPv6 systems (via transition/co-existence technologies)

Motivation for this training

- IPv6 represents a number of challenges: What can we do about them?

Option #1

Option #2

Option #3

Motivation for this training (II)

- Lot of myths have been created around IPv6 security, including:
  - Security as a key component of the protocol
  - Change from network-centric to host-centric paradigm
  - Increased use of IPsec
- They have lead to a general misunderstanding of IPv6 security
- This training:
  - separates fudge from fact, and offers a more realistic view of “IPv6 security”
  - is meant to influence how you “think” about IPv6 security
  - reinforces concepts with hands-on exercises (“hack the talk”)

Some general considerations about IPv6 security

Interesting aspects of 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
- Increased complexity in the resultin Internet::
  - Two inter-networking 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

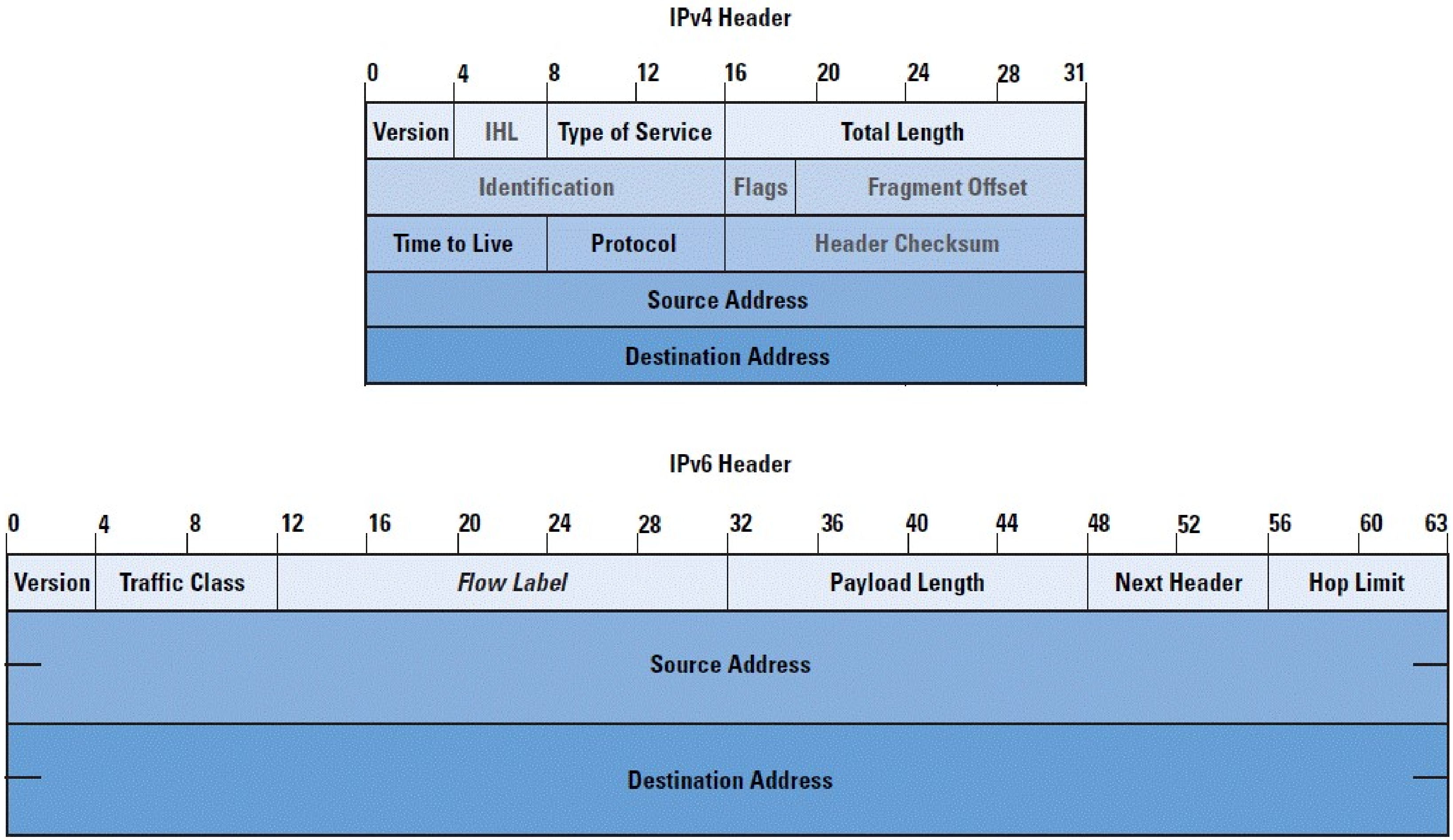
Brief comparision between IPv6/IPv4

- 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 (optional) (+ MLD)**
Fault Isolation	**ICMPv4**	**ICMPv6**
IPsec support	**Opcional**	**Optional**
Fragmentation	**Both in hosts and routers**	**Only in hosts**

Brief comparision between IPv6/IPv4

- Header formats



SI6 NETWORKS

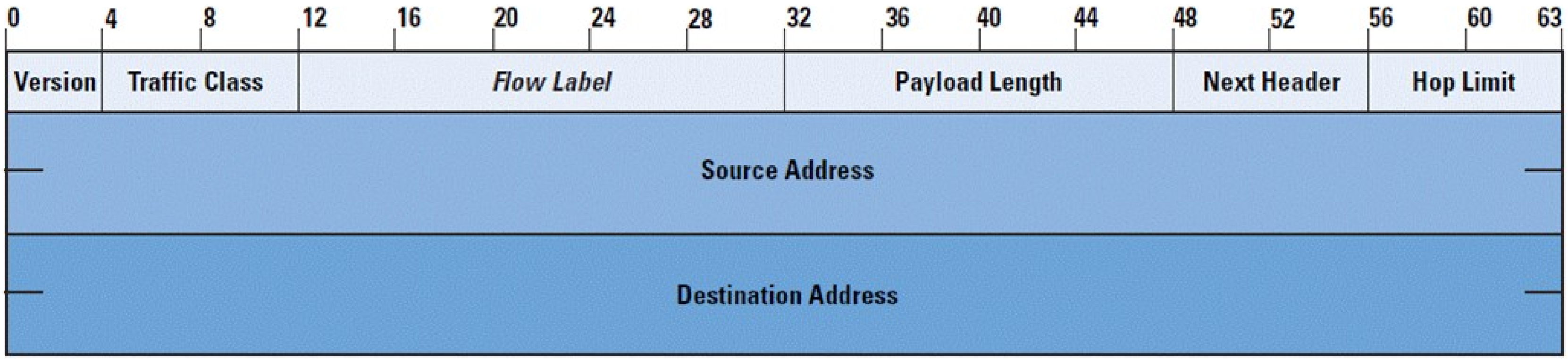
IPv6 header fields

IPv6 header fields

Basic header fields

IPv6 header

- **Fixed-length** (40-byte) 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.

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

- Meant to allow load sharing
- The three-tuple {Source Address, Destination Address, Flow Label} identifies a communication flow
  - Finding the transport-protocol port-numbers con probe to be difficult in IPv6 – if at all possible!
- Currently unused by many stacks
  - Some stacks simply set it to 0 for all packets
  - Other stacks set it improperly
- Flow Label specification has been recently updated

Flow Label (II)

- If the Flow Label is known, an attacker might interfere with load-sharing
  - Just send lots of packet with such Flow Label, such that a particular link is overloaded
- If Flow Labels are predictable, an information leakage might occur
  - e.g., if they are set according to a global counter, an attacker might infer the flow-establishment rate (e.g., new TCP connections/second)
- Advice: Randomize the Flow Label

Assessing the Flow Label policy

- The Flow Label generation policy can be assessed with:

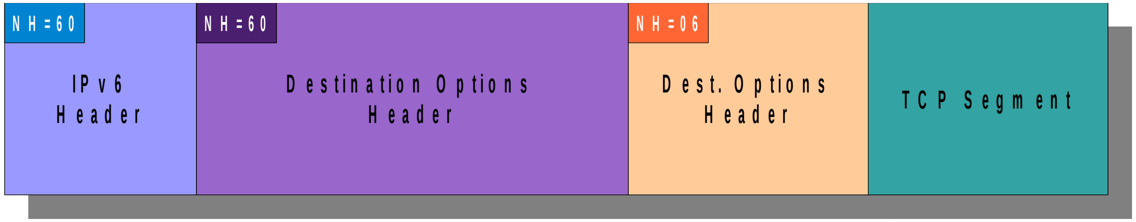
  # ./flow6 -i eth0 -v --flow-label-policy -d fc00:1::1

Payload Length

- Specifies the length of the IPv6 **payload**
- Maximum IPv6 packet is 65855 bytes. However, IPv6 “Jumbograms” can be specified
- A number of sanity checks need to be performed. e.g.:
  - IPv6 Payload Length <= “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, each with multiple options
- 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 the 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 OSes

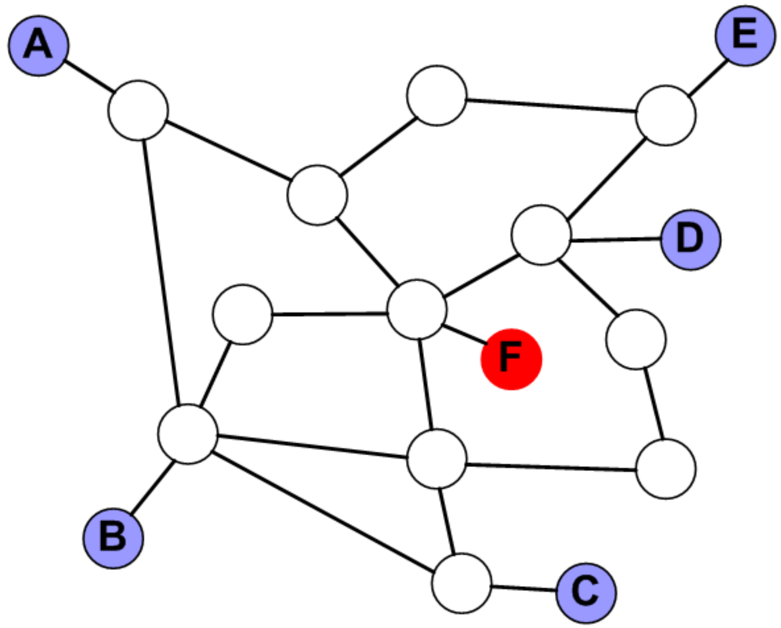
- There are a few default values for the Hop Limit in different OSes
- Based on the received Hop Limit, the original Hop Limit can be inferred
- Example:
  - We receive packets with a Hop LImit of 60
  - We can infer the original Hop LImit was 64
  - We can determine a set of possible remote OSes
- Note: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

Hop Limit: Fingerprinting 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

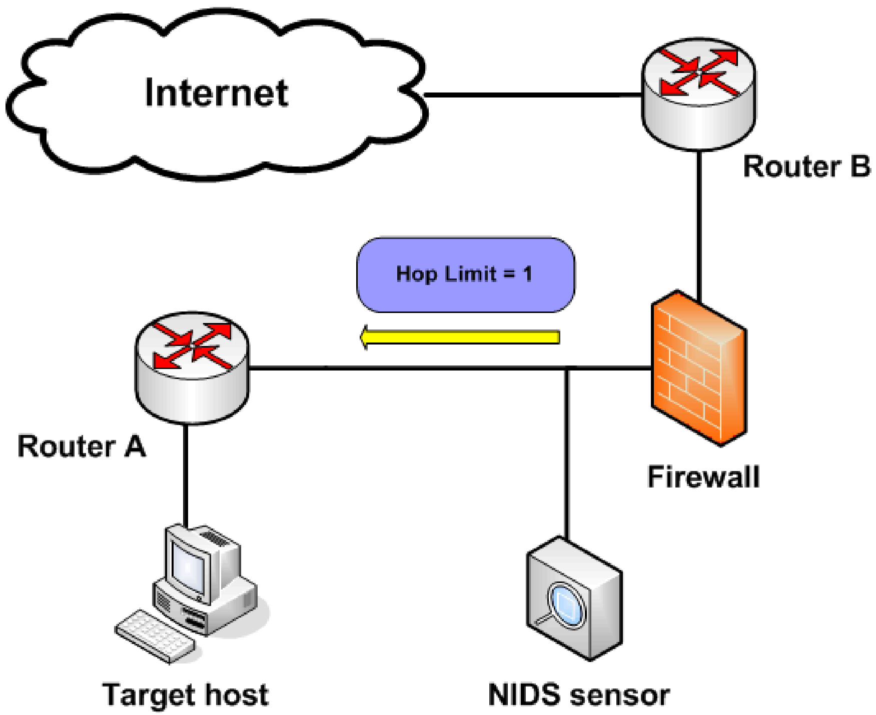


Source	Hop Limit
A	61
B	61
C	61
D	62

F is the only node that is:
- 4 “routers” from A
- 4 “routers” from B
- 4 “routers” from C
- 3 “routers” from D

Hop Limit: Evading NIDS

- 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 (e.g. to 64) or block incomming packets small “Hop Limits”



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
- Basic idea:
  - 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

SI6 NETWORKS

Hop Limit: Improving Security (GTSM) (II)

- 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.)
- 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

IPv6 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

IPv6 Global Unicast Addresses

n bits	m bits	128-n-m bits
Global Routing Prefix	Subnet ID	Interface ID

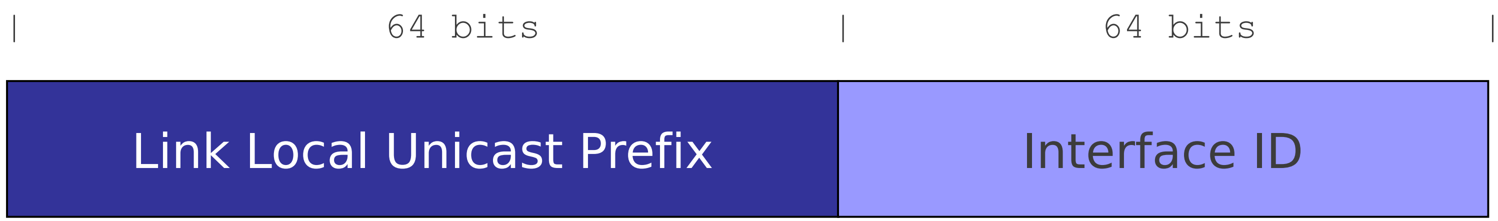
- A number of possibilities for generating the Interface ID:
  - Embed the MAC address (traditional SLAAC)
  - Embed the IPv4 address (e.g. 2001:db8::192.168.1.1)
  - Low-byte (e.g. 2001:db8::1, 2001:db8::2, etc.)
  - Wordy (e.g. 2001:db8::dead:beef)
  - According to a transition/co-existence technology (6to4, etc.)

IPv6 Global Unicast Addresses (II)

24 bits	16 bits	24 bits
IEEE OUI	**FF FE**	Lower 24 bits of MAC

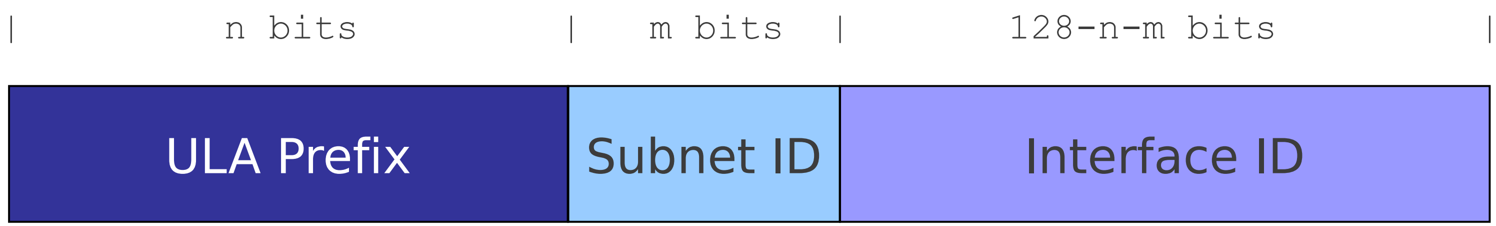
- MAC-derived Interface ID's are constructed as follows:
  - Flip the U/L bit of the OUI (bit 1 of the most significant byte)
  - Insert the word “0xfffe” in between the upper and lower 24-bits

IPv6 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)

IPv6 Unique Local Unicast Addresses



- Special prefix, but syntax equal to that of global unicast addresses

IPv6 address types

Multicast addresses

IPv6 multicast addresses

- Identify a set of nodes
- Can be of different scopes (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

IPv6 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

- The IPv4 Internet originally followed the “End-to-End” principle”:
  - Dumb network, smart hosts
  - Any system can communicate with any other system in the network
  - The “network” does not examine the packet **payloads**
- It is usually argued that this principle fosters innovation
- NATs (and other middle-boxes) have removed this property
- Since IPv6 does not require NATs, it is expected that IPv6 will return the “End to End Principle” to the Internet

IPv6 and the “End to End” Principle

Myth: “*IPv6 will return the 'end to end' property of the Internet*”

- It is assumed that the increased address space will return this property
- However,
  - Global addressability does not imply end to end connectivity
  - Most networks do not care about innovation
  - Users expect in IPv6 the same services as in IPv4
  - End-to-End connectivity increases host exposure
- In summary,
  - End-to-End connectivity is not (necessarily) a desired property
  - Typical IPv6 subnets will only allow “outgoing traffic” (and “return traffic”) (by means of IPv6 firewalls)

IPv6 addressing

Implications on address scanning of remote networks

IPv6 host scanning attacks

"Thanks to the increased IPv6 address space, IPv6 host scanning attacks are unfeasible. Scanning a /64 would take 500.000.000 years"

– Urban legend

Is the search space for a /64 really 2^{64} addresses?

IPv6 addresses in the real world

- Malone measured (*) the address generation policy of hosts and routers in real networks

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

Hosts

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

Routers

Malone, D., "Observations of IPv6 Addresses", Passive and Active Measurement Conference (PAM 2008, LNCS 4979), April 2008, <http://www.maths.tcd.ie/~dwmalone/p/addr-pam08.pdf>.

IPv6 addresses embedding IEEE IDs

24 bits	16 bits	24 bits
IEEE OUI	**FF FE**	Lower 24 bits of MAC
Known or guessable	Known	Unknown

- In practice, the search space is at most ~2^{24} bits – **feasible!**
- The low-order 24-bits are not necessarily random:
  - An organization buys a large number of boxes
  - In that case, MAC addresses are usually consecutive
  - Consecutive MAC addresses are generally in use in geographically-close locations

IPv6 addresses embedding IEEE IDs (II)

- Virtualization technologies present an interesting case
- Virtual Box employs OUI 08:00:27 (search space: ~2^{24})
- VMWare ESX employs:
  - Automatic MACs: OUI 00:05:59, and next 16 bits copied from the low order 16 bits of the host's IPv4 address (search space: ~2^{8})
  - Manually-configured MACs:OUI 00:50:56 and the rest in the range 0x000000-0x3fffff (search space: ~2^{22})

IPv6 addresses embedding IPv4 addr.

- They simply embed an IPv4 address in the IID
  - e.g.: 2000:db8::192.168..1
- Search space: same as the IPv4 search space

IPv6 “low-byte” addresses

- The IID is set to all-zeros, except for the last byte
  - e.g.: 2000:db8::1
  - There are other variants..
- Search space: usually 2^8 or 2^{16}

Mitigating address scanning attacks

- Remove any patterns in address generation
- Block traffic when scanning attacks become obvious
  - Lots of packets from the same source, directed to multiple destinations
  - Most destination addresses will correspond to non-existing nodes

Industry mitigations for scanning attacks

- Microsoft replaced the MAC-address-based identifiers with (non-standard) randomized IIDs
  - Essentially RFC 4941, but they don't vary over time
- Certainly better than MAC-address-based IIDs, but still not “good enough”
- They mitigate host-scanning, but **not** host tracking (more on this later)

Conclusions about scanning attacks

- IPv6 address scanning attacks are **feasible**, but typically harder than in IPv4
- They require more “intelligence” on the side of the attacker
- It is **possible** to make them infeasible
- It is likely that many other scanning strategies/techniques will be explored (more on this later)

IPv6 addressing

Implications on address scanning of local networks

Overview

- Leverage IPv6 all-nodes link-local multicast address
- Employ multiple probe types:
  - Normal multicasted ICMPv6 echo requests (don't work for Windows)
  - Unrecognized options of type 10xxxxxx
- Combine learned IIDs with known prefixes to learn all addresses
- Technique implemented in the scan6 tool of SI6's IPv6 toolkit
  - Available at: http://www.si6networks.com/tools

Scanning a local network with scan6

- Simply run scan6 as.
  - # ./scan6 -i INTERFACE -l

IPv6 addressing

Implications on privacy

Host-tracking attacks

- Traditional IIDs are constant for each interface
- As the host moves, the prefix changes, but the IID doesn't
  - the 64-bit IID results in a super-cookie!
- This introduces a problem not present in IPv4: **host-tracking**
- Example:
  - In net #1, host configures address: 2001:db8:1::1111:22ff:fe33:4444
  - In net #2, host configures address: 2001:db8:2::1111:22ff:fe33:4444
  - The IID "1111:22ff:fe33:4444" leaks out host "identity".

Mitigation for host-tracking attacks

- RFC 4941: privacy/temporary addresses
  - Random IIDs that change over time
  - Typically generated **in addition** to traditional SLAAC addresses
  - Only OpenBSD uses them in **replacement** of traditional SLAAC addresses
  - Traditional addresses used for server-like communications, temporary addresses for client-like communications
- Operational problems:
  - Difficult to manage!
- Security problems:
  - They mitigate host-tracking **only partially**
  - They **do not** mitigate host-scanning attacks

IPv6 addressing

Mitigating remote-scanning and privacy issues

Auto-configuration address/ID types

	Stable	Temporary
Predictable	IEEE ID-derived	None
Unpredictable	**NONE**	RFC 4941

- We lack stable privacy-enhanced IPv6 addresses
  - Used to replace IEEE ID-derived addresses
  - Pretty much orthogonal to privacy addresses
  - Probably “good enough” in most cases even without RFC 4941

Stable privacy-enhanced addresses

- draft-ietf-6man-stable-privacy-addresses proposes to generate Interface IDs as:

F(Prefix, Interface_Index, Network_ID, Secret_Key)

- Where:
  - F() is a PRF (e.g., a hash function)
  - Prefix SLAAC or link-local prefix
  - Interface_Index is the (internal) small number that identifies the interface
  - Network_ID could be e.g. the SSID of a wireless network
  - Secret_Key is unknown to the attacker (and randomly generated by default)

Stable privacy-enhanced addresses (II)

- As a host moves:
  - Prefix and Network_ID change from one network to another
  - But they remain constant within each network
  - F() varies across networks, but remains constant within each network
- This results in addresses that:
  - Are stable within the same subnet
  - Have different Interface-IDs when moving across networks
  - For the most part, they have “the best of both worlds”
- Not yet implemented in any IPv6 stack

Possible mitigations for local-scanning

- Do not respond to multicasted ICMPv6 echo requests
  - Currently implemented in Windows
- Multicasted IPv6 packets containing unsupported options of type 10xxxxxx should not elicit ICMPv6 errors
  - See draft-gont-6man-ipv6-smurf-amplifier
- **However**, it's virtually impossible to mitigate IPv6 address scanning of local networks
  - Think about mDNS, etc.

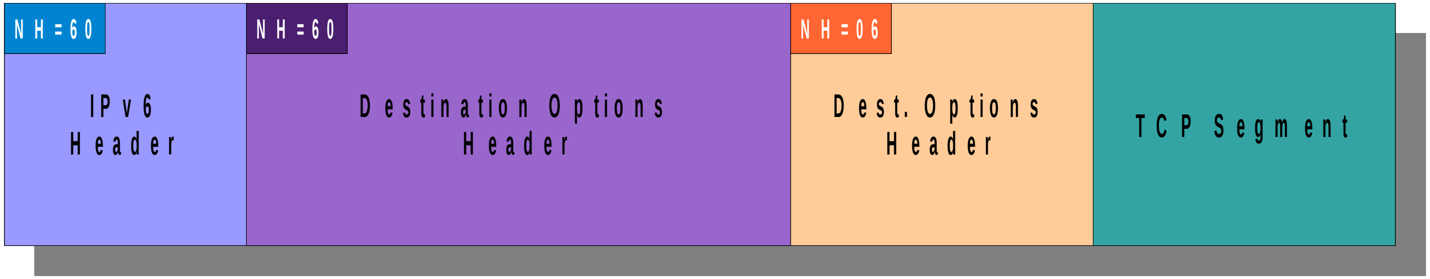
IPv6 Extension Headers

IPv6 Extension Headers

General implications of Extension Headers

General IPv6 packet format

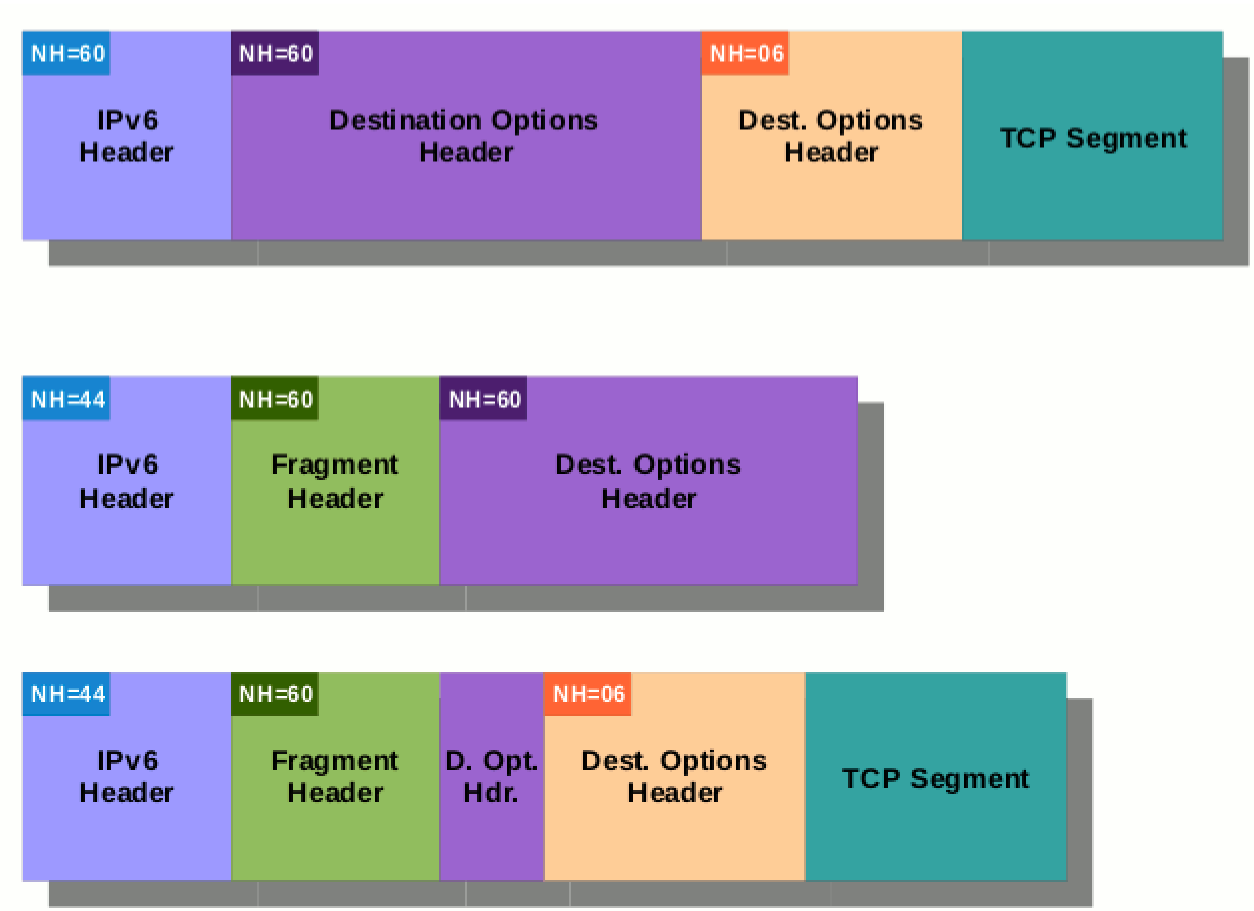
- Consists of an IPv6 header chain and an (optional) payload
- Each Extension Header is typically encoded as TLV (Type-Length-Value)
- Any number of instances of any number of different headers are allowed
- Each header may contain an arbitrary number of options



SI6 NETWORKS

General IPv6 packet format (II)

- Traffic may get uglier as a result of fragmentation



General issues with Extension Headers

- Large number of headers/options may have a negative impact on performance
- It is harder to spot e.g. layer-4 information
- It may be impossible to "identify" which "type" of packet a specific fragment belongs to.

IPv6 Extension Headers

Fragment Header

IPv6 Fragmentation Overview

- IPv6 fragmentation performed only by hosts (never by routers)
- Fragmentation support implemented in “Fragmentation Header”

8 bits	8 bits	13 bits	2b	1b
Next Header	Reserved	Fragment Offset	Res	M
Identification				

- Where:
  - Fragment Offset: Position of this fragment with respect to the start of the fragmentable part
  - M: “More Fragments”, as in IPv4
  - “Identification”: Identifies the packet (with Src IP and Dst IP)

Fragmentation: Example

- `% 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)**

Fragmentation: Security Implications

- Fragmentation known to be painful for NIDS
- Fragment reassembly is a state-full mechanism
  - Potential for DoS attacks
- Predictable Fragment IDs well-known from the IPv4 world
  - idle-scanning
  - DoS attacks (fragment ID collisions)
- Situation exacerbated by larger payloads resulting from:
  - Larger addresses
  - DNSSEC
- But no worries, since we learned the lesson from the IPv4 world... – **right?**

Fragment ID generation policies

Operating System	Algorithm
FreeBSD 9.0	Randomized
NetBSD 5.1	Randomized
OpenBSD-current	Randomized (based on SKIPJACK)
Linux 3.0.0-15	Predictable (GC init. to 0, incr. by +1)
Linux-current	Unpredictable (PDC init. to random value)
Solaris 10	Predictable (PDC, init. to 0)
Windows 7 Home Prem.	Predictable (GC, init. to 0, incr. by +2)

GC: Global Counter PDC: Per-Destination Counter

At least Solaris and Linux patched in response to our IETF I-D – more patches expected!

Predictable Fragment IDs: Example

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

IP6 (hlim 64, next-header Fragment (44) payload length: 368) 2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007a:1448|360)

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

IP6 (hlim 64, next-header Fragment (44) payload length: 584) 2004::1 > 2004::5e26:aff:fe33:7063: frag (0x4973fb3d:1232|576)

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

IP6 (hlim 64, next-header Fragment (44) payload length: 368) 2004::5e26:aff:fe33:7063 > 2004::1: frag (0x0000007b:1448|360)

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

IP6 (hlim 64, next-header Fragment (44) payload length: 584) 2004::1 > 2004::5e26:aff:fe33:7063: frag (0x2b4d7741:1232|576)

Assessing the Frag. ID policy

- The Fragment ID generation policy can be assessed with:

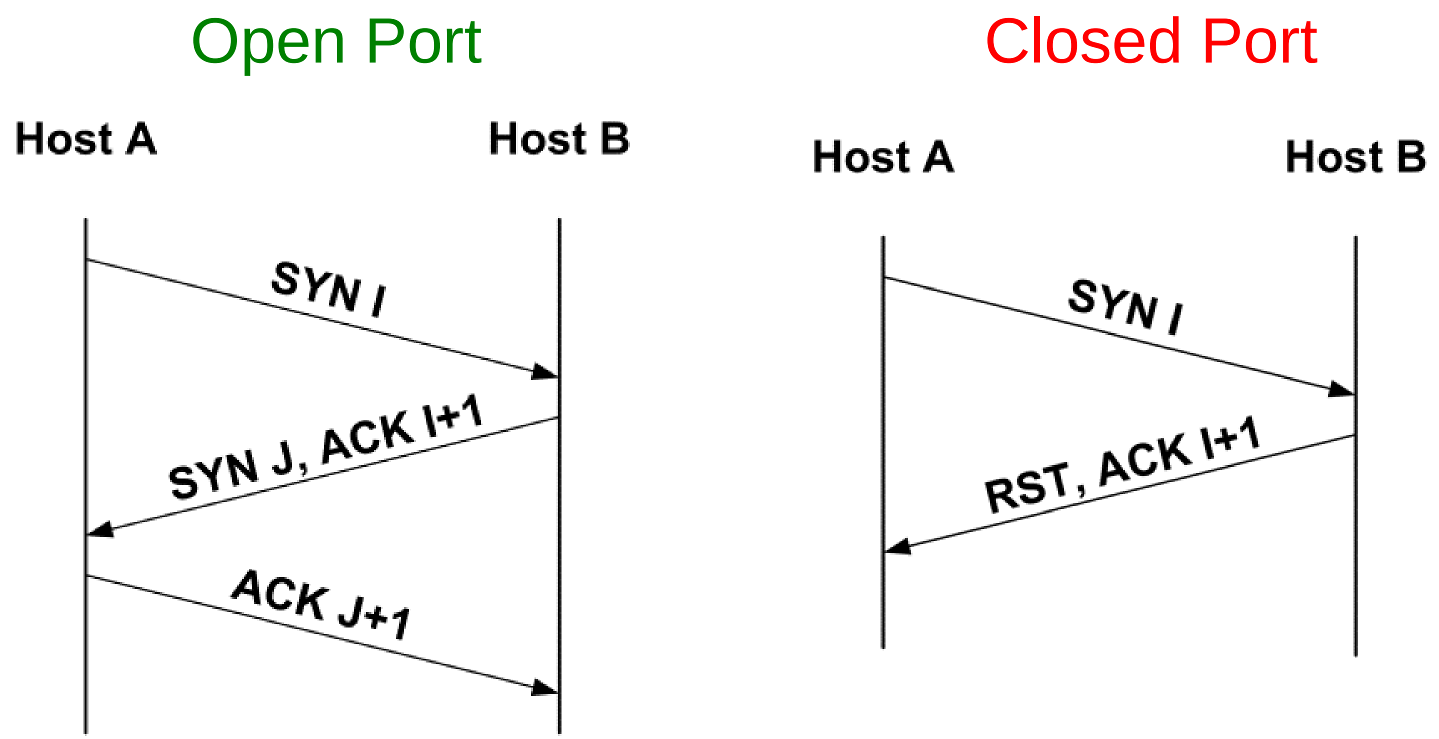
  ```
  # ./frag6 -i eth0 -v --frag-id-policy -d fc00:1::1
  ```

Idle scan: Introduction

- Stealth port scanning technique
- Allows port scanning without the attacker sending any packets to the target with its real Source Address.
- The attacker only needs a host that employs predictable Identification values.

Idle scan: TCP 3WHS review

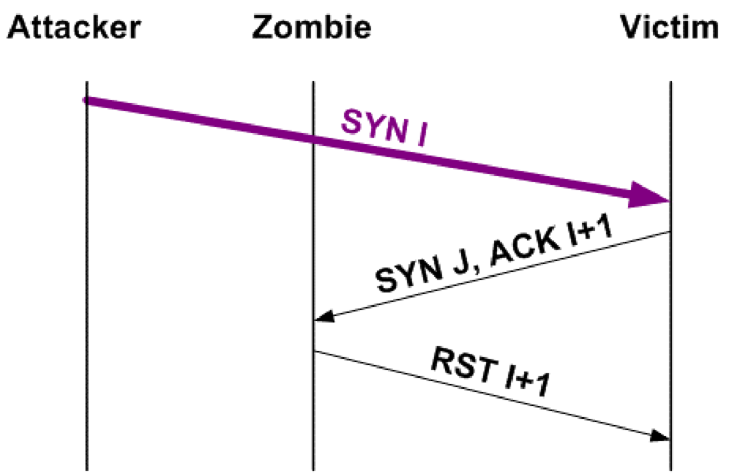
- Normal TCP 3WHS



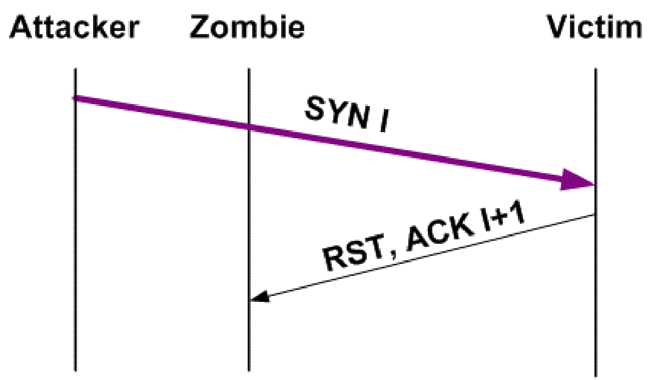
Idle scan: TCP 3WHS review

- TCP 3WHS with spoofed segments

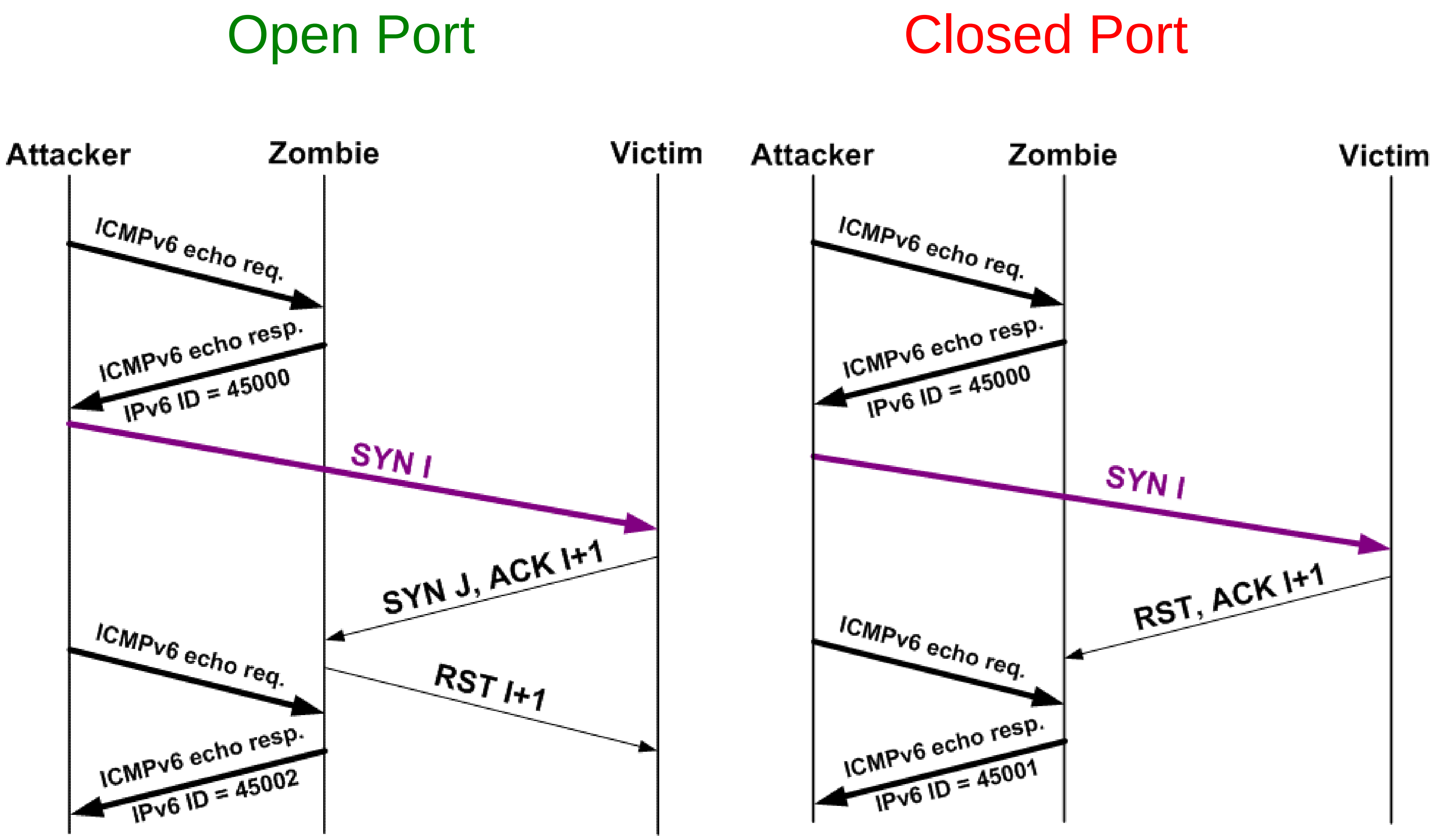
Open Port



Closed Port



Idle scan implementation



Mitigating predictable Frag. IDs

- Goal: Make the Fragment Identification unpredictable
- Border conditions:
  - Identification value is 32-bit long, but...
  - Translators only employ the low-order 16 bit
  - A Frag ID should not be reused too frequently
- Possible schemes
  - Simple randomization
  - More “elaborate” randomization schemes
  - Hash-based
- Discussed in IETF I-D: draft-gont-6man-predictable-fragment-id

IPv6 Fragmentation & NIDS

- Security implications of overlapping fragments well-known (think Ptacek & Newsham, etc,)
- Nonsensical for IPv6, but originally allowed in the specs
- Different implementations allow them, with different results
- RFC 5722 updated the specs, forbidding overlapping fragments
- Most current implementations reflect the updated standard
- See http://blog.si6networks.com
- Assess the fragment reassembly policy of a target with:

```
# ./frag6 -i eth0 -v --frag-reass-policy -d fc00:1::1
```

IPv6 Fragment Reassembly Implications

- Fragment reassembly is a state-full mechanism
- Hosts need to tie memory resources for each received fragment
- An attacker could forge lots of fragmented packets
  - System memory would be tied to them
  - Unless proper limits and garbage collection is enforced, would lead to DoS

Assessing fragment reassembly

./frag6 -i eth0 -s ::/0 –flood-frags 100 -l -z 5 -d fc00:1::1 -v

IPv6 atomic fragments

- ICMPv6 PTB < 1280 triggers inclusion of a Fragment Header in all subsequent packets to that destination
- This is not real fragmentation – the whole original datagram is contained in a single fragment
- Result: IPv6 atomic fragments (Frag. Offset=0, More Frag.=0)
- Some implementations mix these packets with “normal” fragmented traffic
- End result:
  - You can cause any “flow” to employ fragmentation
  - Then launch any fragmentation-based attack

IPv6 atomic fragments (II)

- Atomic fragment do not need to be mixed with other fragments – they are **atomic**!
- Skipping the normal reassembly procedure eliminates fragmentation-based attacks for such traffic
- draft-ietf-6man-ipv6-atomic-fragments fixes that:
  - IPv6 atomic fragments required to be processed as non-fragmented traffic
  - Document has passed WGLC
  - Should be published as an IETF RFC soon

Handling of IPv6 atomic fragments

Operating System	Atomic Frag. Support	Improved processing
FreeBSD 8.0	No	No
FreeBSD 8.2	Yes	No
FreeBSD 9.0	Yes	No
Linux 3.0.0-15	Yes	**Yes**
NetBSD 5.1	No	No
OpenBSD-current	Yes	**Yes**
Solaris 11	Yes	**Yes**
Windows Vista (build 6000)	Yes	No
Windows 7 Home Premium	Yes	No

At least OpenBSD patched in response to our IETF I-D – more patches expected!

Assessing support for atomic fragments

- Check response to atomic fragments

  # ./frag6 -i eth0 --frag-type atomic --frag-id 100 -d fc00:1::1

- Assess support for atomic fragments:

  # ./frag6 -i eth0 --frag-type first --frag-id 100 -d fc00:1::1

  # ./frag6 -i eth0 --frag-type atomic --frag-id 100 -d fc00:1::1

sysctl's for fragment 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
Routing Header

Brief overview

- RHT0 provided similar functionality to that of IPv4 source routing
- Allowed the inclusion of many more IPv6 addresses

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Next Header  |  Hdr Ext Len  |  Routing Type | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                                                               .
.                       type-specific data                      .
.                                                               .
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Brief overview (II)

- Security implications publicly discussed in CanSecWest 2006
  - Can be leveraged to make packets bounce between network addresses
  - Exacerbated by the fact that some **hosts** “forwarded” them
- Formally obsoleted by RFC 5095 in 2007
- However, it was enabled by default prior to 2007
  - Might still be the case for some deployed systems

IPv6 Options of type 10xxxxxx

Security Implications

IPv6 Smurf-like Attacks

- Options of type 10xxxxxx require hosts to generate ICMPv6 errors even if the packet was destined to a multicast address
- They could be exploited for smurf-like DoS attacks
- Probably less important than the IPv4 smurf attacks (since it requires multicast routing)
- Might be an issue if multicast routing is deployed
- draft-gont-6man-ipv6-smurf-amplifier addresses this issue:
  - Discusses the problem
  - Recommends that multicasted packets must not elicit ICMPv6 errors

Local host scanning attacks

- Options of type 10xxxxxx may be used to elicit responses from all systems in the local network
- Perform a local host scanning attack:

  # ./scan6 -i eth0 -l -p unrec

Internet Control Message Protocol version 6 (ICMPv6)

Brief Overview

- ICMPv6 is a core protocol of the IPv6 suite, and is used for:
  - Address Resolution (Neighbor Discovery)
  - Stateless address auto-configuration (SLAAC)
  - Fault isolation (ICMPv6 error messages)
  - Troubleshooting (ICMPv6 informational messages)
- ICMPv6 is **mandatory** for IPv6 operation

ICMPv6 Error Messages

IPv6 Destination Unreachable

- Specified in RFC 4443
- Popular error codes:
  - 0 - No route to destination
  - 1 - Communication with destination administratively prohibited
  - 2 - Beyond scope of source address
  - 3 - Address unreachable
  - 4 - Port unreachable
  - 5 - Source address failed ingress/egress policy
  - 6 - Reject route to destination

IPv6 Destination Unreachable (II)

- They could potentially be exploited for connection-reset attacks
  - No known vulnerable systems, though
- They can be useful to:
  - Avoid delays in connection establishments (see RFC 5461)
  - Provide more detailed information about network failures to the user
- Implications of filtering these messages discussed in draft-ietf-opsec-icmp-filtering

ICMPv6 Packet Too Big

- Specified in RFC 4443
- Employed for traditional Path MTU Discovery (RFC 1981)
  - Critical, since PMTUD is mandatory (unless an MTU of 1280 bytes is used)
  - ICMPv6 PTB are already filtered in many networks :-(
  - Hosts should implement PMTUD black-hole detection (RFC 4821)
- MUST NOT be filtered!

ICMPv6 Packet Too Big (II)

- Security implications:
  - Performance-degrading attacks (cause the MTU to be reused)
  - Trigger fragmentation on a connection, and then launch a fragmentation-based attack
- Mitigations:
  - Hosts should validate received ICMPv6 PTB (e.g. check the TCP SEQ)
  - Many implementations **do not**! :-(
  - Critical information might be missing in the received ICMPv6 PTB

ICMPv6 Time Exceeded

- Two different codes:
  - 0 - Hop limit exceeded in transit
  - 1 - Fragment reassembly time exceeded
- Essentially similar to their IPv6 counterparts
  - Code 0: typically indicates forwarding loops, or use of IPv6 traceroute
  - Code 1: indicates that fragment reassembly timed out

ICMPv6 Time Exceeded (Hop Limit)

- As noted, traceroute depends on these messages
- Sample traceroute output:

```
% 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
```

ICMPv6 Time Exceeded (Hop Limit) (II)

- Resulting tcpdump output:

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

ICMPv6 Time Exceeded (Hop Limit) (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" (or other similar value) instead

ICMPv6 Informational Messages

ICMPv6 Informational Messages

- 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 Informational Messages
Echo Request/Response

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”

```
% 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)

ICMPv6 Informational Messages

Node Information Query/Response

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

- 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).

NI Query/Response: Examples

- 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
```

NI Query/Response: Examples (II)

- 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:
  - Host A sends a NS: Who has IPv6 address fc01::1?
  - Host B responds with a NA: I have IPv6 address, and the corresponding MAC address is 06:09:12:cf:db:55.
  - Host A caches the received information in a “Neighbor Cache” for some period of time (this is similar to IPv4’s ARP cache)
  - 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”

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Type      |     Code      |          Checksum             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           Reserved                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
//                       Target Address                          //
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |   Options ...
 +-+-+-+-+-+-+-+-+-+-+-+-
```

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”

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|S|O|                     Reserved                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                       Target Address                          +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

Source/Target Link-layer Address Option

- 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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |    Link-Layer Address ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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: ND diagnostic tool

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

```
$ ndisc6 fc00:1::1 eth00
Soliciting fc00:1::1 (fc00:1::1) on eth0...
Target link-layer address: 08:00:27:F9:73:04
 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”:

```
% 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”:

```
$ 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
```

Neighbor Discovery for IPv6

Address Resolution 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.

Performing the attack with the na6 tool

- Run the tool as:

  # ./na6 –i IFACE –W VICTIMADDR –L –E MACADDR –c –o

- Now send traffic to the victim, and it should be sent to MACADDR

- Verify it with tcpdump as:

  # tcpdump –i em0 –e –vv ip6

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!

Performing the attack with the na6 tool

- Run the tool as:

  # ./na6 –i IFACE –W VICTIMADDR –L –E ff:ff:ff:ff:ff:ff –c –o

- Now send traffic to the victim, and it should be sent to ff:ff:ff:ff:ff:ff
- Verify it with tcpm dump as:

  # tcpdump –i IFACE –e –vv ip6

- It is also nice to:
  - First run tcpdump from another host on the network (you'll see no traffic)
  - Launch the attack, and repeat – now you'll see the traffic, as it is broadcasted

Introduce a forwarding loop at a router

- Respond the NS sent by a router with an NA containing the router's link-layer address
- 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.

Performing the attack with the na6

- Run the tool as:

  # ./na6 –i IFACE –W fc00:1::80 –L –E ROUTERMAC –c –o

- On the router, run:

  % ping6 –i 20 –w 40 fc00:1::80 &

- On the router, verify it with tcpdump as:

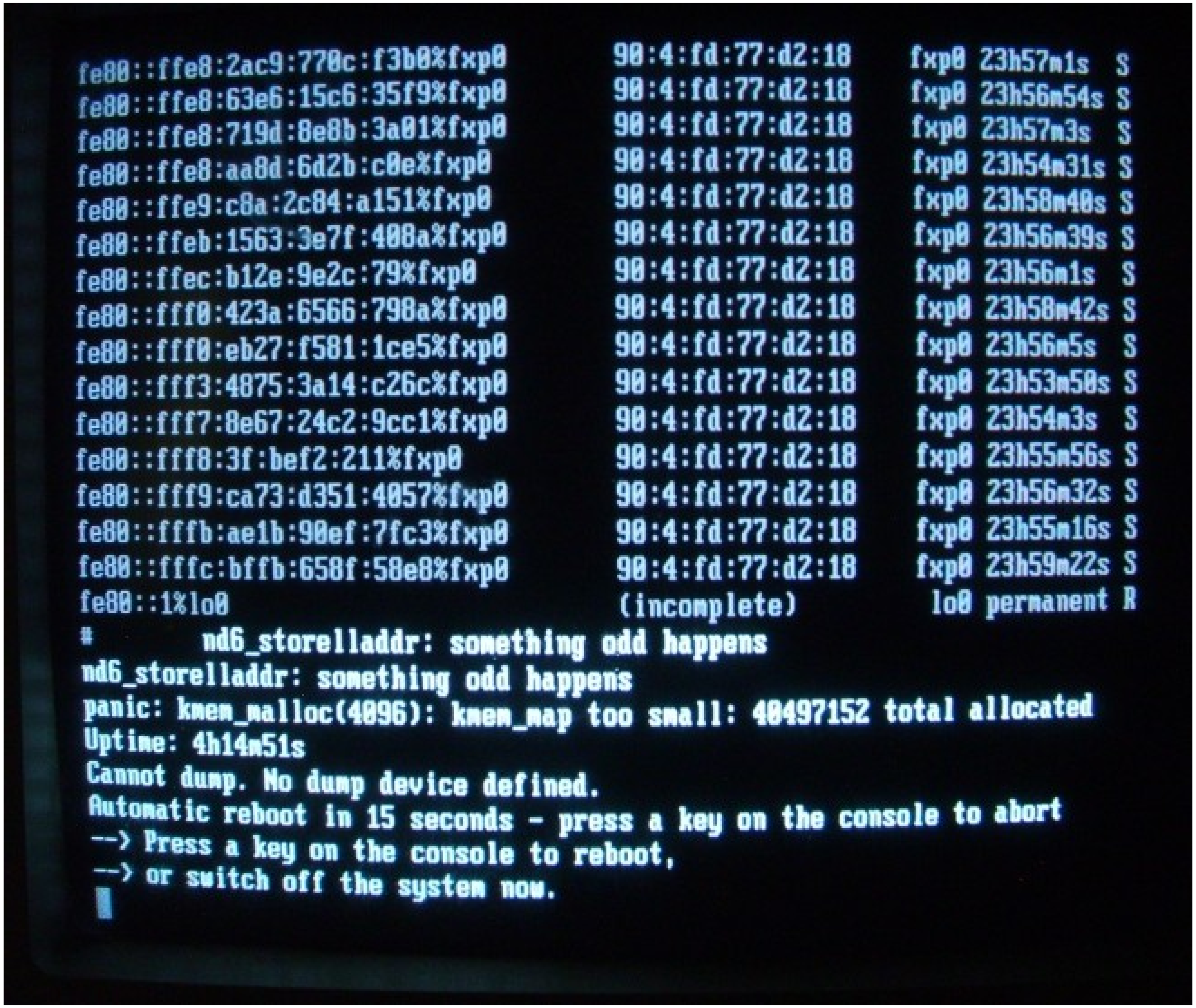
  sudo tcpdump –i em0 –e –vv ip6

- Pay attention to the Hop Limit of each of the packets!

Overflowing the Neighbor Cache

- Some implementations (e.g., FreeBSD and NetBSD) fail t enforce limits on the number of entries 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 ND (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.

Introduce a forwarding loop at a router

- Respond the NS sent by a router with an NA containing the router's link-layer address
- 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.

Neighbor Discovery for IPv6

Address Resolution Attacks – Countermeasures

Possible mitigations for ND attacks

- Deploy SEND (SEcure Neighbor Discovery)
- Monitor Neighbor Discovery traffic (e.g., with NDPMon)
- Restrict access to the local network
- Use static entries in the Neighbor Cache

Secure Neighbor Discovery (SEND)

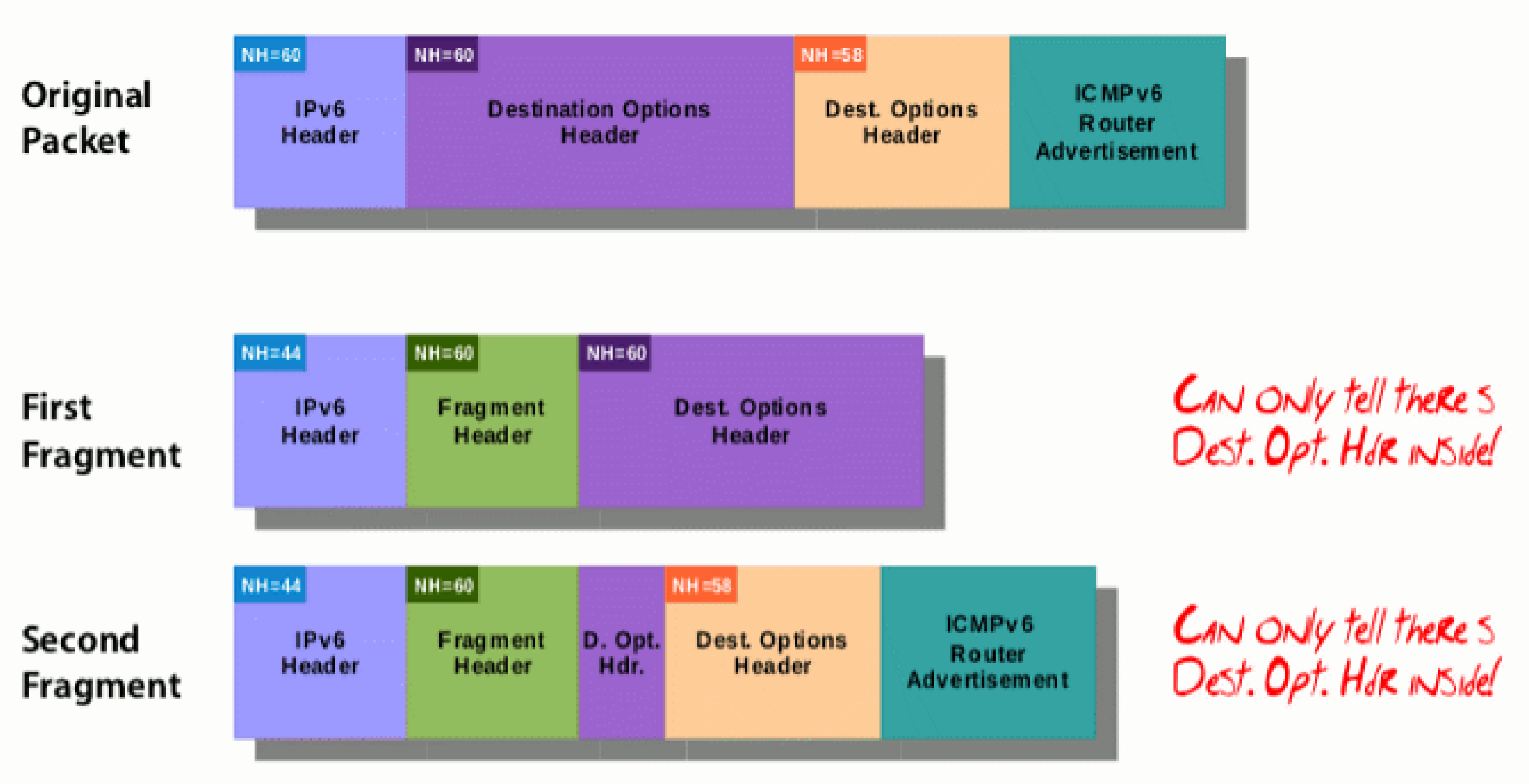
- Cryptographic approach to the problem of forged Neighbor Solicitation messages:
  - Certification paths certify the authority of routers
  - Cryptographically-Generated Addresses (CGAa) bind IPv6 addresses to an asymmetric key pair
  - RSA signatures protect all Neighbor Discovery messages
- SEND is hard to deploy:
  - Not widely supported
  - The requirement of a PKI is a key obstacle for its deployment
  - Other key pieces of the puzzle remain unsolved (DNS, etc.)

Neighbor Discovery traffic monitoring

- Some tools (e.g. NDPMon) keep 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

ND-monitoring evasion

- Fundamental problem: complexity of traffic to be "processed at layer-2"
- Example:



Solving ND traffic monitoring

- Ban the use of IPv6 fragmentation with Neighbor Discovery
- It is not needed!
  - Same amount of information can be sent in multiple packets
- Ongoing work at the IETF: draft-ietf-6man-nd-extension-headers
  - Forbids the use of fragmentation with ND messages

Static Neighbor Cache entries

- Static entries avoid "dynamic" mapping
- 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 *BSD

- 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

Neighbor Discovery for IPv6

Stateless Address Auto-configuration (SLAAC)

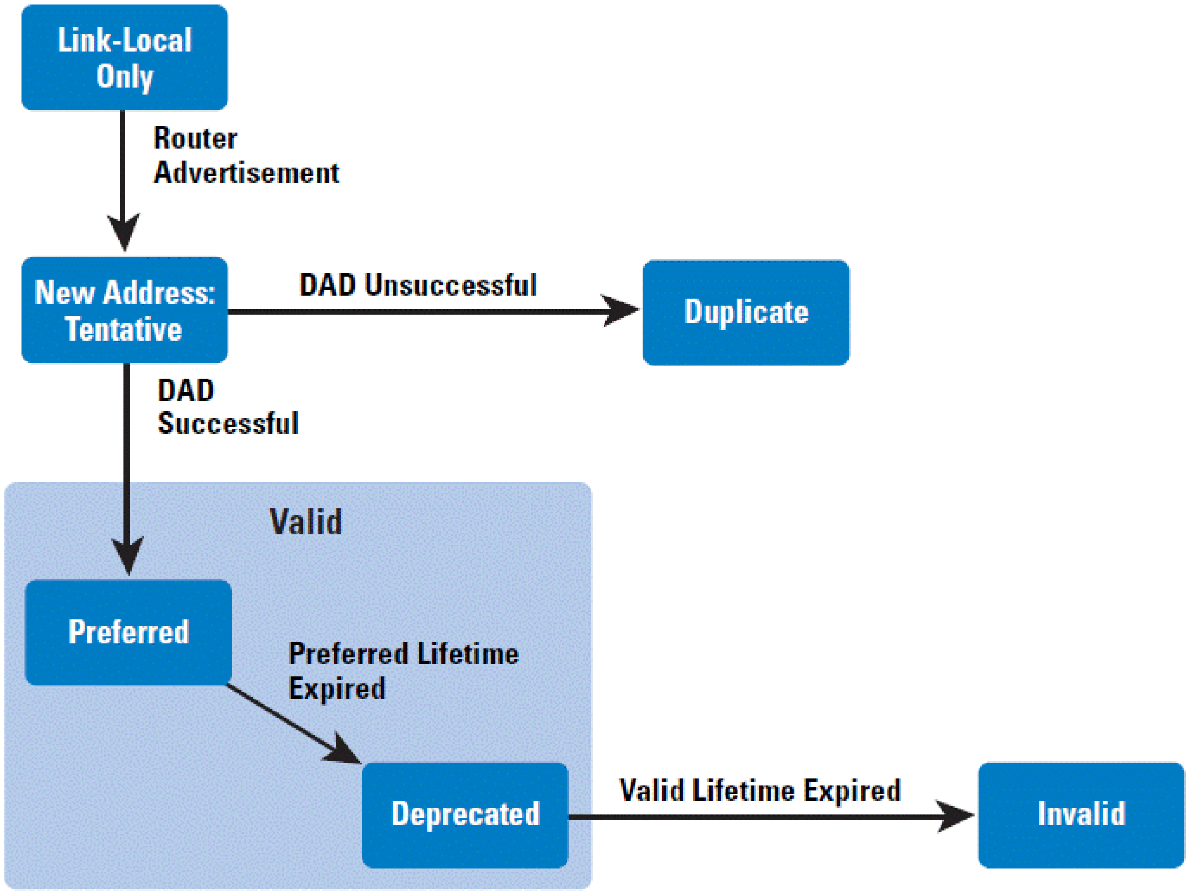
Brief overview

- Two auto-configuration mechanisms in IPv6:
  - Stateless Address Auto-Configuration (SLAAC)
    - Based on ICMPv6 messages
  - DHCPv6
    - Based on UDP packets
- SLAAC is mandatory, while DHCPv6 is optional
- Basic operation of SLAAC:
  - Host solicit configuration information by sending Router Solicitation messages
  - Routers convey that information in Router Advertisement messages:
    - Auto-configuration prefixes
    - Routes
    - Network parameters
    - etc.

SLAAC: Step by step

- 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



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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Reserved                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

Router Advertisement Messages

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cur Hop Limit |M|O|H|Prf|Resvd|       Router Lifetime         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Reachable Time                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Retrans Timer                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options ...
+-+-+-+-+-+-+-+-+-+-+-+-
```

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
- They typically include many of them

Prefix Information Option

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

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |     Type      |    Length     | Prefix Length |L|A|R|Reserved1|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                         Valid Lifetime                        |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Preferred Lifetime                      |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                           Reserved2                           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
//                            Prefix                            //
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Router Information Option

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     | Prefix Length |Resvd|Prf|Resvd|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Route Lifetime                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Prefix (Variable Length)                    |
.                                                               .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

MTU Option

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              MTU                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RDNSS Option

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Length    |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Lifetime                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:            Addresses of IPv6 Recursive DNS Servers            :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SLAAC: Sample packet log

```
17:28:50 :: > ff02::1:ffaf:1958: icmp6: neighbor sol: who has
fe80::20c:29ff:feaf:1958 (len 24, hlim 255)

17:28:52 fe80::20c:29ff:feaf:1958 > ff02::2: icmp6: router solicitation (src
lladdr: 00:0c:29:af:19:58) (len 16, hlim 255)

17:28:52 fe80::20c:29ff:fec0:97b8 > ff02::1: icmp6: router advertisement(chlim=64,
router_ltime=1800, reachable_time=0, retrans_time=0)(src lladdr: 00:0c:29:c0:97:b8)
(prefix info: LA valid_ltime=2592000, preferred_ltime=604800, prefix=2004:1::/64)
(len 56, hlim 255)

17:28:52 :: > ff02::1:ffaf:1958: icmp6: neighbor sol: who has
2004:1::20c:29ff:feaf:1958 (len 24, hlim 255)
```

rdisc6: Troubleshooting tool

- Sends RS messages, and decodes RA responses
- Sample output:

```
# rdisc6 -v eth0
Soliciting ff02::2 (ff02::2) on eth0...

Hop limit                 :           64 (      0x40)
Stateful address conf.    :           No
Stateful other conf.      :           No
Router preference         :       medium
Router lifetime           :           30 (0x0000001e) seconds
Reachable time            :  unspecified (0x00000000)
Retransmit time           :  unspecified (0x00000000)
 Prefix                   : fc00:1::/64
  Valid time              :      2592000 (0x00278d00) seconds
  Pref. time              :       604800 (0x00093a80) seconds
 Source link-layer address: 00:4F:4E:12:88:0F
 from fe80::24f:4eff:fe12:880f
```

Prefix Information (*BSD)

- % ndp -p

```
% 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
```

Default routers (*BSD)

- % ndp -r

```
% 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
```

Routing table (*BSD)

- % netstat -r -p ip6

```
# 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
```

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.

Autoconf Addresses & Privacy

- Traditional SLAAC addresses embed the MAC address of the interface
- 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

Some sysctl's for Privacy Addresses

- Sysctl's that control their operation (in FreeBSD):
  - 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)

Neighbor Discovery for IPv6

SLAAC attacks

Exploit DAD for DoS attacks

- Listen to NS messages with the Source Address set to the IPv6 “unspecified” address (::).
- Respond to such messages 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
- Perform this attack with the na6 tool as follows:

  # ./na6 -i IFACE -b ::/128 -c -o -L -vv

  Or possibly:

  # ./na6 -i em0 -b ::/128 -B VICTIMMAC -c -o -L -vv

Advertise a malicious Current Hop Limit

- Advertise a small Current Hop Limit such that packets are discarded by the intervening routers
- Perform this attack with ra6 as follows:

  # ./ra6 –i IFACE –s ROUTERADDR –d TARGETADDR –c HOPS –v

Advertise a malicious MTU

- Advertise a small Current Hop Limit such that packets are discarded by the intervening routers
- Perform this attack with ra6 as follows:

  # ./ra6 –i IFACE –s ROUTERADDR –d TARGETADDR –M MTU

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”
- Perform this attack with the ra6 tool:

  # ./ra6 -i IFACE -s ROUTERADDR -d TARGETADDR -t 0 -l 1 –v

Flood hosts with autoconf prefixes

- Flood the local network with auto-configuration prefixes
- Perform this attack with the ra6 tool as follows:

  # ./ra6 -i IFACE -d TARGETADDR --flood-prefixes 40 -P ::/64#LA -l -z 10 -e -vvv

Flood hosts with specific routes

- Flood the local network with “more specific routes”
- Perform this attack with the ra6 tool as follows:

  # ./ra6 -i IFACE -d TARGETADDR --flood-routes 40 -R ::/64#1 -l -z 10 -e -vvv

SI6 NETWORKS

Flood hosts with default routers

- Flood the local network with auto-configuration prefixes
- Perform this attack with the ra6 tool as follows:

  # ./ra6 -i IFACE -d TARGETADDR --flood-sources 40 -l -z 10 -e -vv

Neighbor Discovery for IPv6

SLAAC attacks – Countermeasures

Possible mitigations for SLAAC attacks

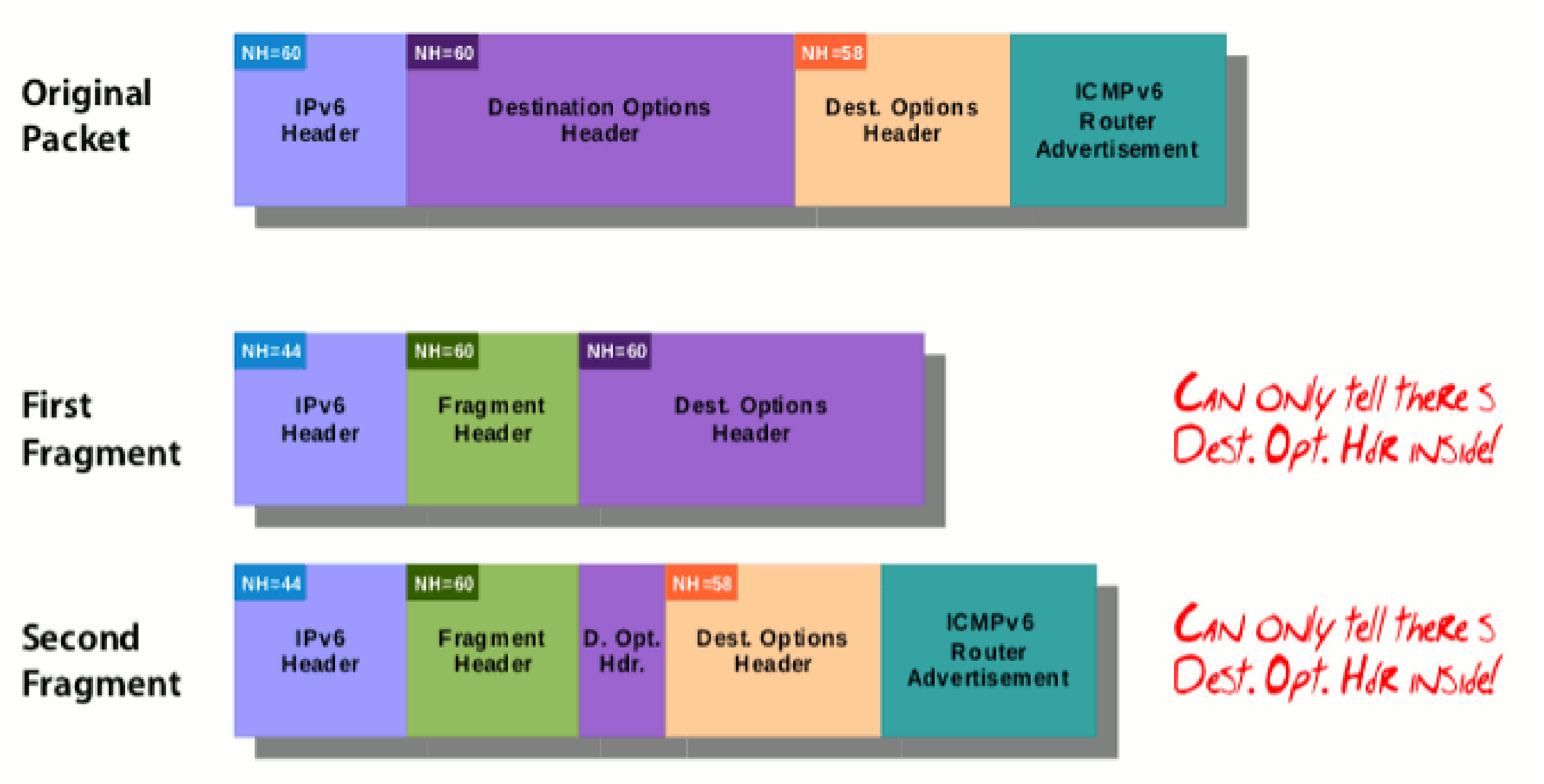
- Deploy SEND (SEcure Neighbor Discovery)
- Monitor Neighbor Discovery traffic (e.g., with NDPMon)
- Restrict access to the local network
- Deploy Router Advertisement Guard (RA-Guard)

RA-Guard (Router Advertisement Guard)

- Filtering policy enforced by layer-2 devices
- Works (roughly) as follows:
  - RA-Guard allows RAs only if they are received on pre-specified ports
  - Otherwise, they are dropped
- RA-Guard **asumes** that it is possible to identify RAs
- All known implementations can be evaded with IPv6 Extension Headers and/or fragmentation

RA-Guard evasion

- Fundamental problem: complexity of traffic to be “processed at layer-2”
- Example:



Fixing RA-Guard

- In essence,
  - Follow the entire IPv6 header chain when trying to identify RAs
  - Drop the packet if it is an RA or you cannot positively determine that the packet is non-RA
- Ongoing work at the IETF to fix RA-Guard:
  - draft-ietf-v6ops-ra-guard-implementation
  - More human-readable explanation at: <http://blog.si6networks.com>

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

Security Implications

- 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
- Possible mitigations:
  - DHCPv6-Shield: draft-gont-opsec-dhcpv6-shield
  - draft-ietf-6man-oversized-header-chain (improves DHCPv6-Shield)

Multicast Listener Discovery

Brief overview

- A generic protocol that allows hosts to inform local routers which multicast groups they are interested in.
- Routers use thi 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
- For many networks, the only use for MLD with Neighbor Discovery is MLD-snooping switches

Security Implications

- 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 optional for both IPv6 and IPv4
- It used to be mandatory for IPv6 – 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 and Considerations

- 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

SI6 NETWORKS

IPv6 Transition Co-Existence Technologies

Brief Overview

- 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

IPv6 Transition Co-Existence Technologies

Dual Stack

Brief Overview

- 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-ony 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/

IPv6 Transition Co-Existence Technologies

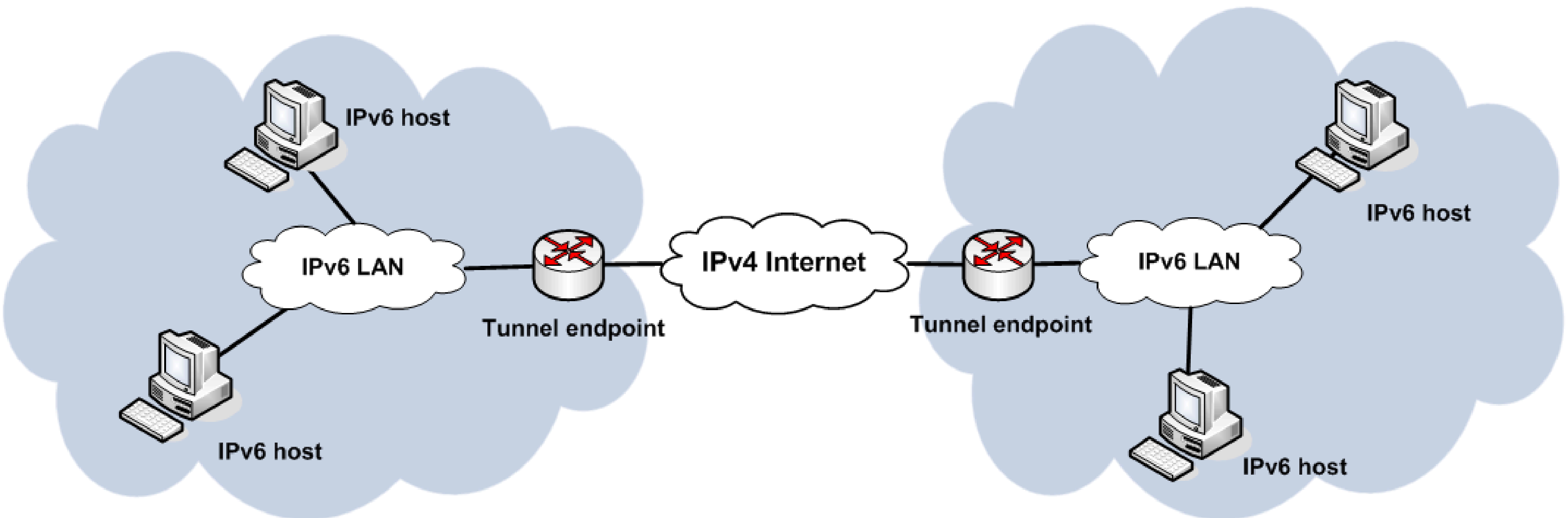
Tunnels

Tunnels

- Transport IPv6 packets from/to IPv6 islands over IPv4
- 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

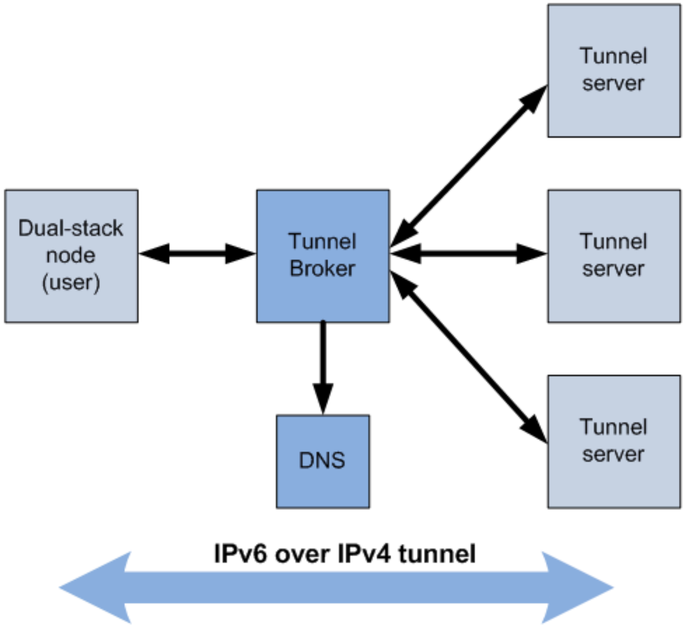
6in4

- 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)

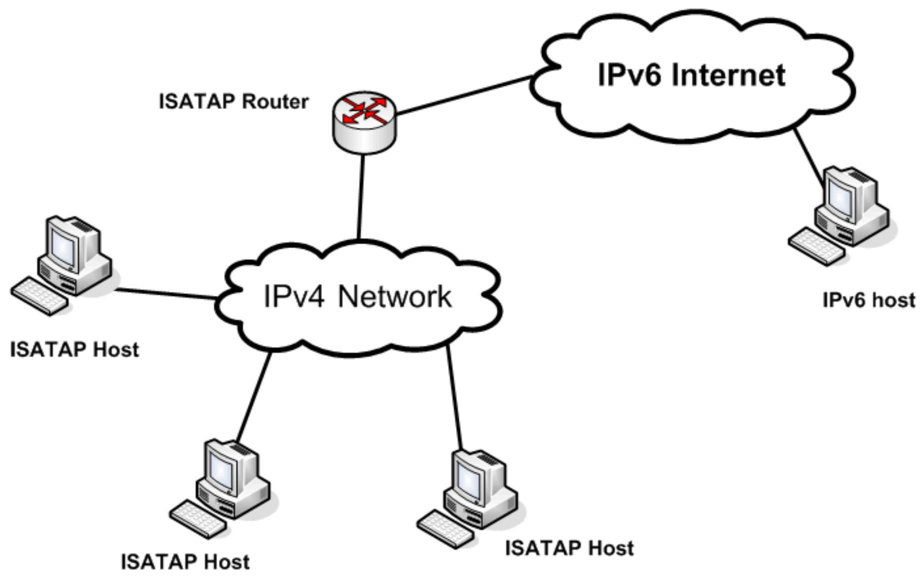


Tunnel Broker: Sample Implementation

- 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

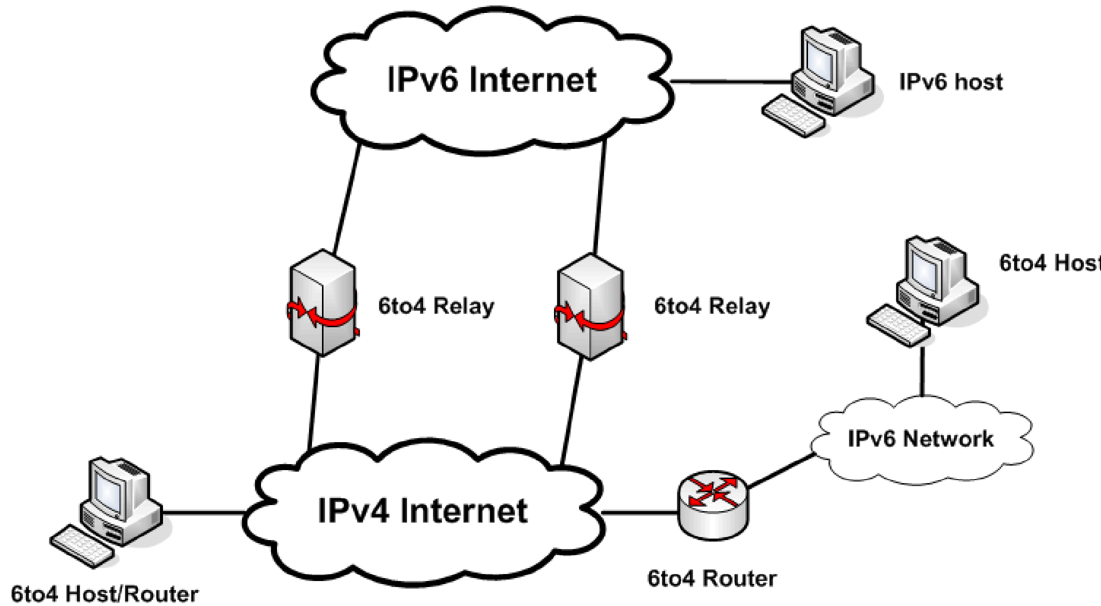
```
|0              1|1             3|3                              6|
|0              5|6             1|2                              3|
+----------------+----------------+--------------------------------+
|000000ug00000000|0101111011111110|          IPv4 address          |
+----------------+----------------+--------------------------------+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

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

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



IPv6 Address format

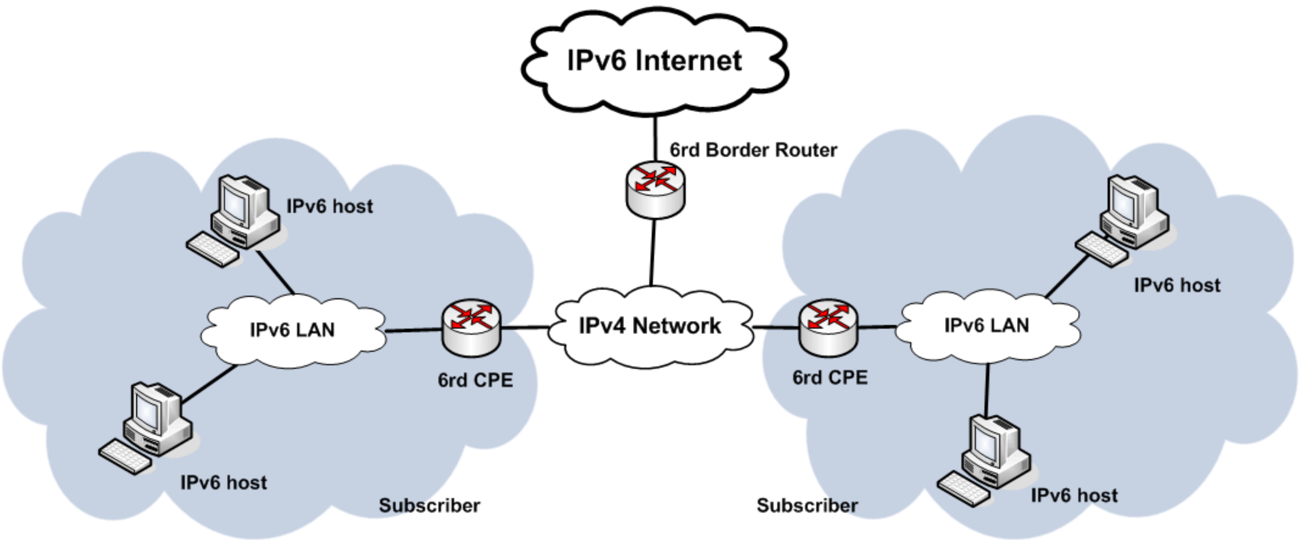
```
|   16    |     32      |   16    |               64 bits               |
+---------+-------------+---------+-------------------------------------+
|  2002   |   V4ADDR    | Subnet  |            Interface ID             |
+---------+-------------+---------+-------------------------------------+
```

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)

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

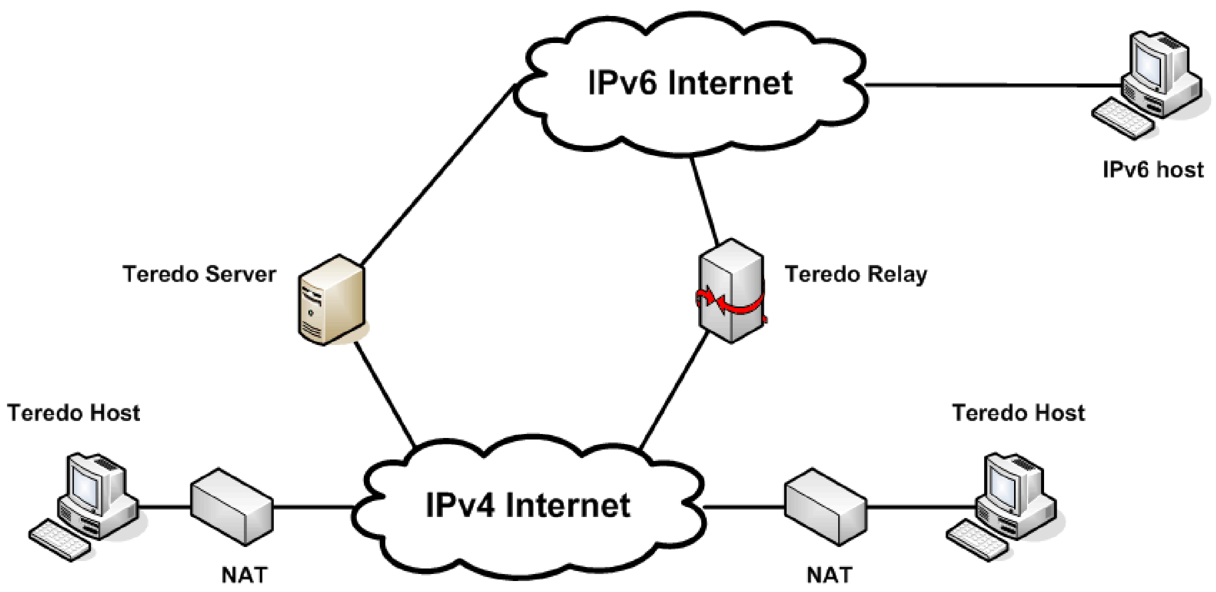


Address format

```
|     n bits     |    o bits     |   m bits  |    128-n-o-m bits      |
+----------------+---------------+-----------+------------------------+
|  6rd prefix    | IPv4 address  | subnet ID |     interface ID       |
+----------------+---------------+-----------+------------------------+
|<--- 6rd delegated prefix --->|
```

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 |
+-------------+-------------+-------+------+-------------+
```

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

IPv6 Transition Co-Existence Technologies

Translation

Translation

- All previous transition/co-existence technologies require both IPv4 and IPv6 addresses – what if there are no IPv4 addresses left?
- Mechanisms have been developed by 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…..

IPv6 firewalling

Meta-issues

- What is an IPv6 firewall?
  - Should it support just native IPv6?
  - Should it support transition technologies?
- What functionality can/should we expect from an IPv6 firewall?

A sample technical challenge

- Specs-wise, state-less IPv6 packet filtering is impossible
  - The IPv6 header chain can span multiple fragments
- draft-ietf-6man-oversized-header-chain tries to improve that:
  - The entire IPv6 header chain must be within the first PMTU bytes of the packet
  - i.e. packets with header chains that span more than one fragment may be blocked – don't send them!
  - Already accepted as a 6man wg item!
- There's an insanely large amount of work to be done in the area of IPv6 firewalling

Security Implications of IPv6 on IPv4 Networks

Security Implications

- Most implementations support and enable dual-stack by default
- Many support transition technologies, and enable them 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 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	(IP proto 41) || (UDP Dest Port 3653 || TCP Dest Port 3653)

Some conclusions

Some conclusions

- Many IPv4 vulnerabilities have been re-implemented in IPv6
  - We just didn't learn the lesson from IPv4, or,
  - Different people working in IPv6 than working in IPv4, or,
  - The specs could make implementation more straightforward, or,
  - **All of the above?** :-)
- Still lots of work to be done in IPv6 security
  - We all know that there is room for improvements
  - **We need IPv6, and should work to improve it**

Key areas where further work is needed

Key areas where 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.

Questions?

Thanks!

Fernando Gont

fgont@si6networks.com

IPv6 Hackers mailing-list

http://www.si6networks.com/community/

www.si6networks.com