FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

IPv6

A security analysis of tomorrow's communication protocol

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering and Internet Computing

eingereicht von

Eduard Thamm, BSc.
Matrikelnummer 0525087

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Christian Platzer

Wien, 10.07.2015

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

IPv6

A security analysis of tomorrow's communication protocol

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Eduard Thamm, BSc.
Registration Number 0525087

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Christian Platzer

Vienna, 10.07.2015

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Eduard Thamm, BSc.
Neblingergasse 6/2, 1130 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Abstract

The introduction of a new basis for addressing and handling communication between networking nodes necessitates the analysis of these concepts and protocols. IPv6 was initially constructed with a huge amount of conceptual changes from IPv4, including the mandatory addition of IPSec. Some of these changes have made it to implementation level, while others have been dropped. This thesis evaluates the current state of the testing software available for IPv6 security analysis and proceeds by performing tests on the network stacks of current operating systems. This work will be facilitated by a laboratory setup specifically for this purpose. Evaluation of the data gained through this approach provides valuable information concerning the current state of IPv6. The thesis ends with the conclusion that, although there is still work to be done, IPv6 is ready to be deployed from an endpoint security point of view. Possible fields of future work include precise analysis of the reasons for discovered vulnerabilities, for example through source code review, and testing in contexts offering a higher level of complexity.

Kurzfassung

Mit der Einführung eines neuen Protokolls zur Regelung der Kommunikation zwischen Computern und Netzwerken ergibt sich die Notwendigkeit, die zugrundeliegenden Konzepte und deren Umsetzung zu analysieren. IPv6 wies ursprünglich eine große Zahl an Veränderungen im Vergleich zu IPv4 auf, zum Beispiel wurde IPSec verpflichtend für alle Umsetzungen vorgeschrieben. Einige dieser Änderungen konnten sich jedoch nicht durchsetzen. Diese Diplomarbeit evaluiert den aktuellen Stand der Technik im Hinblick auf Werkzeuge zur Überprüfung von Systemen auf Sicherheit im Bereich IPv6 und wendet diese Werkzeuge dann auf aktuelle Umsetzungen von IPv6 Kommunikationsschnittstellen an. Zu diesem Zweck wurde ein Versuchsnetzwerk aufgebaut. Die gewonnenen Daten wurden ausgewertet und geben wertvolle Hinweise zum aktuellen Entwicklungsstand von IPv6. Die Arbeit schließt mit dem Ergebnis, dass IPv6, obwohl noch einiges an Arbeit in diesem Feld vonnöten ist, vom Sicherheitsstandpunkt aus einsatzbereit ist. Zukünftige Forschung in diesem Bereich sollte eine Ergründung der Ursachen der gefundenen Schwachstellen und das Durchführen von Tests in komplexeren Umgebungen umfassen.

Contents

1 Introduction **1**
- 1.1 Aim And Context Of This Work . . . 1
- 1.2 Methodology . . . 2
- 1.3 Contribution and Structure of the Thesis . . . 3

2 Background **5**
- 2.1 The Internet: Brief History . . . 5
  - 2.1.1 ARPANET and What Followed . . . 5
  - 2.1.2 Address Depletion and IPv6 . . . 6
- 2.2 Overview of Remote Communication Theory . . . 6
  - 2.2.1 Layered Approach . . . 6
  - 2.2.2 Known Issues . . . 8
    - 2.2.2.1 Spanning Tree . . . 9
    - 2.2.2.2 Shortest Path between Vertices . . . 9
    - 2.2.2.3 Two Generals Problem . . . 9
- 2.3 IPv6 - A Brief Introduction . . . 9
  - 2.3.1 Deployment Status . . . 10
  - 2.3.2 IPv6 Addressing Architecture . . . 11
    - 2.3.2.1 IPv6 Address Scopes . . . 11
    - 2.3.2.2 Unicast Addresses . . . 12
    - 2.3.2.3 Unique Local Addresses . . . 13
    - 2.3.2.4 Anycast Addresses . . . 13
    - 2.3.2.5 Multicast Addresses . . . 13
    - 2.3.2.6 IPv6 Interface Identifiers . . . 14
  - 2.3.3 Stateless Address Autoconfiguration . . . 14
  - 2.3.4 Security Features in IPv6 . . . 15
    - 2.3.4.1 Authentication Header . . . 15
    - 2.3.4.2 Encapsulating Security Payload . . . 16
- 2.4 Comparison of IPv6 and IPv4 . . . 17
  - 2.4.1 Addresses and Address Space . . . 17
  - 2.4.2 Address Configuration . . . 17
  - 2.4.3 Internet Control Message Protocol . . . 18
    - 2.4.3.1 Path MTU Discovery . . . 18

2.4.3.2 Neighbour Discovery . . . 18
2.4.4 Header Structure . . . 19
2.4.5 Multicast . . . 20

3 Exploiting the Internetworking Layer **21**
3.1 Attacks on IPv4 with Relevance to IPv6 . . . 21
3.1.1 Smurf Attacks . . . 21
3.1.2 Fragmentation Attacks . . . 22
3.1.2.1 Fragment Overlap Attacks . . . 23
3.1.2.2 Fragment Overrun Attacks . . . 23
3.1.2.3 Too Many Datagrams Attacks . . . 23
3.1.3 Fuzzing . . . 24
3.2 Newly Rising Issues in IPv6 . . . 24
3.2.1 Headers . . . 24
3.2.1.1 Routing . . . 25
3.2.1.2 Destination Options Header . . . 27
3.2.1.3 Hop-By-Hop Options Header . . . 28
3.2.2 Neighbor Discovery . . . 30
3.2.3 Secure Neighbor Discovery . . . 34
3.2.3.1 Address Ownership . . . 35
3.2.3.2 Router Legitimacy . . . 38
3.2.3.3 Replay Protection . . . 40
3.2.3.4 Security Of This Protocol . . . 41

4 Tool Suites **43**
4.1 IPv6 Toolkit . . . 43
4.2 THC-IPV6 . . . 45
4.3 Test Suite . . . 51
4.4 Choosing Exploits . . . 51
4.4.1 Manual testing . . . 52
4.4.2 Automation . . . 53
4.5 Details of chosen Exploits . . . 54
4.5.1 denial6 . . . 54
4.5.2 exploit6 . . . 54
4.5.3 fragmentation6 . . . 55
4.5.4 fuzz_ip6 . . . 55
4.5.5 implementation6 . . . 56
4.5.6 sendpees6 . . . 58
4.5.7 sendpeesmp6 . . . 58
4.5.8 smurf6 . . . 58
4.6 Encountered Problems . . . 58

5 Laboratory Setup and Data Acquisition **61**
5.1 Setup of Testing Network . . . 61

5.2 Choice Of Hosts . . . 64
5.3 Data Acquisition . . . 66
5.4 Encountered Problems . . . 66
5.4.1 Data Acquisition . . . 66
5.4.2 Android . . . 67
5.4.3 Virtualisation . . . 67

6 Results **69**
6.1 Result Processing . . . 69
6.1.1 Automated Testing Tool . . . 69
6.1.2 Laboratory Notebook . . . 70
6.1.3 Network Dump . . . 70
6.2 Results . . . 70
6.2.1 Denial of Service . . . 71
6.2.1.1 Router Alert or Destination Header . . . 71
6.2.1.2 Secure Neighbor Discovery . . . 72
6.2.2 Fuzzing . . . 73
6.2.2.1 Multicast Listener Report . . . 73
6.2.2.2 Router Advertisements . . . 73
6.2.2.3 Android . . . 74
6.2.3 Implementation . . . 74
6.2.3.1 Node Information Query . . . 74
6.2.3.2 Echo Request From Local Multicast Address . . . 75
6.2.3.3 Fragmented and Source-Routed Echo Request From Local Multicast Address . . . 75
6.3 Summary . . . 76

7 Conclusion **77**

Bibliography **79**

CHAPTER 1

Introduction

This chapter will present an overview of the academically relevant information. The context and goals of the thesis as well as its methodology will be detailed, and the inclined reader will be able to gain a perspective of where this work fits into the current research efforts on IPv6 security.

1.1 Aim And Context Of This Work

This work has three major goals.

The first objective is a concise summary of capabilities of current exploitation toolkits available for IPv6. This is necessary due to the fact, that the documentation for these kits is often very disparate between tools within a single kit. This can most likely be attributed to the fact that either the tools differ greatly in functionality or that the documentation is written by a multitude of authors with no designated documentation manager for the complete project. In addition to this fact, some tools are documented in a fashion that does not help the understanding of their inner workings or operation.

Secondly, this thesis will evaluate the presence and prevalence of vulnerabilities, which are weaknesses or flaws in a system's implementation, design, management, or operation that when exploited violate the system's security policy. Another focus of this part of the thesis is relating current attacks and vectors to vectors that have been present in IPv4 and have been eliminated in that version of the protocol. [66]

Finally, the main research questions will be discussed:

1. Is IPv6 fit to deploy on a large scale from the perspective of computer security?
2. From a computer security perspective, what would be the possible result of loosing all IPv4 capability effective immediately?

Question 2 can be reformulated to “What would happen, if we must move to IPv6 today?”.

Given the fact that the initial Request For Comments detailing what would later be known as IPv6 was released in December 1995 and given the number 1883, one should assume that these questions have already been thoroughly answered. Requests For Comments are documents released by the Internet Engineering Task Force, containing organisational as well as technical notes and specifications relevant to the Internet. This, however, is not the case. Though extensive research was done, only very few publications directly related to the security of IPv6 as a protocol were found. A large number of those focused on the mobile parts of the protocol. [102] [140] [144] [51] [113] [46]

Looking at the overall picture gained during the literature research phase, the following can be summarised:

There is a research group focused on IPv6 security working in close relationship with the Internet Engineering Task Force. The group’s main public events and relations are managed by Frenando Gont, who is also a member of SI6Networks, a company providing software and training for IPv6 security testing. One main line of communication associated with this group appears to be the “IPv6Hackers” mailing list. [35] [118] [21]

Another main resource of information are white papers published by vendors of internetwork hardware. They often focus on very concrete configuration examples and problem/vulnerability mitigation, but at times also deliver valuable concepts, ideas and general insight into a problem. In addition, they offer a good perspective on the design versus implementation problems faced by IPv6.

Finally, there is a multitude of IPv6 security related papers from institutions and individuals which focus on small subproblems or very general comparisons of IPv4 and IPv6, often in very specific environments. [132] [145]

1.2 Methodology

To achieve the afore mentioned goals the following methodology will be adhered to.

1. Using literature analysis, key differences and similarities of IPv6 and IPv4 will be identified.
2. A laboratory will be constructed which will contain multiple hosts. Each host will run a different operating system. In addition, different mobile hardware platforms will be used to run nodes.
3. A detailed analysis of the capabilities of current exploitation frameworks will be performed by reviewing their documentation and performing manual review of their features using a “field-test” approach.
4. From the results of the previous steps, exploits will be selected which will then be run against the hosts of the laboratory environment.

5. A presentation of results will then highlight the findings and discuss possible reasons and problems related to the discovered weakness.

6. A general discussion will then sum up all knowledge gained throughout this thesis and revisit the points that have arisen in context of the overall research question.

The use of exploitation frameworks grants multiple advantages over developing every used exploit manually.

- They are tested by multiple users and contributors. This ensures a higher quality of work when compared with single developer solutions.
- The implementation of exploits is a process usually consuming large amounts of time. Frameworks are often readily available. This enables the tester to cover a larger number of test cases.
- The use of a framework also decreases the post-test workload. This enables the tester to spend more time interpreting and verifying results.

1.3 Contribution and Structure of the Thesis

During the progression of this thesis 4 major results will be presented:

- A detailed comparison of attack vectors and known attacks on IPv4 and IPv6.
- A previously non-existent analysis and comparison of available IPv6 exploitation toolkits.
- A modular and easily extendible framework for automated testing based on current tools.
- An analysis of the current IPv6 security state of modern day operating systems.

This thesis is divided into eight chapters. The first chapter presents general information about the thesis, talks about context and methodological details of the work and sharpens the goals this thesis is meant to achieve.

The second and third chapter endeavour to provide an exhaustive overview of relevant information needed in the further parts of the thesis. While chapter two focuses on general internetworking concepts, basics of IPv6 and the comparison of IPv6 and IPv4, chapter 3 is concerned with the attack surface of the new protocol. It is subdivided into two sections which deal with vectors that persisted from IPv4 to IPv6 and new issues respectively.

The fourth chapter then presents preparatory concepts by providing an in depth review of available toolsets. Building on the knowledge gained during this step, chapter five presents the laboratory that was used to conduct tests and gives details on the methods used to acquire data. Chapter six will then discuss the results of the laboratory and provide details on how the data were processed. In addition, results of the laboratory and literature research phase will be reviewed and put into context.

The final chapter will then provide a concise summary of the knowledge gained in this thesis and give an outlook on possible future work.

CHAPTER 2

Background

This chapter will provide a brief historical overview of the Internet along with some details on the core concepts used throughout the rest of this work. It will then provide an introduction into IPv6 and a comparison of IPv6 and IPv4.

2.1 The Internet: Brief History

In this section, a short overview of the history and development of the Internet will be given. Not only will this motivate the need for IPv6, but insight into some historical milestones and road bumps might also help gain a better understanding for why certain architectural and design choices were made the way they were made.

The overview provided here is not and can not be a complete history. There are a multitude of books solely focusing on this topic, some of which go into tremendous amounts of detail. All this section aims to do is to provide an understanding of the fact that the Internet is a constantly changing and evolving thing, in which change is not only necessary but also embraced.

2.1.1 ARPANET and What Followed

In the late 1950's, the United Stated Department of Defence wanted a command-and-control network able to survive a nuclear war. Around 1960, RAND Corporation was contracted to find a solution to that problem and one of its employees, Paul Baran, designed a highly distributed and fault-tolerant telecommunication system. He proposed the use of digital packet switching technology due to technical limitation of analog signals. Although the Pentagon liked the concept, AT&T, a large telecommunications company, dismissed it out of hand. [143] [9]

In 1968, the Advanced Research Projects Agency commissioned the building of a subnet from BBN, a Massachusetts based consulting company. BBN decided to spilt the software into two parts. Namely a host to interface message processor and interface message processor to interface message processor. They implemented the later part and considered their job done, leaving the Advanced Research Projects Agency with a system missing important parts. [143] [9]

To address this problem, Larry Roberts, the director of the Advanced Research Projects Agency, convened a meeting of network researchers, mostly graduate students, in the summer of 1969. Although there was no overall concept and no grand design, an experimental network went live in December of that same year. Initially, four universities were connected through the "ARPANET". These four universities were chosen due to their completely incompatible computers and the fact that they all had a large number of Advanced Research Projects Agency contracts. From there on, the network expanded rapidly and by September 1972, thirty-four organisations were connected. [143] [9]

Further research effort was directed at the use of satellite networks and mobile packet radio.

In the years to follow, protocols for internetwork communications, especially TCP/IP, were developed. Easy access to the network interface was established through the development of sockets. Ways to find specific hosts became necessary and a multitude of other tools were developed to meet specific needs. In addition, the first wide area networks were established and by the mid 1980's the collection of joined networks became, through the use of this word by the community, the Internet. [143] [9]

2.1.2 Address Depletion and IPv6

If 1969 is considered as the year the Internet was born, it took approximately 42 years to use up most of the addressing space we thought we would ever need. On April 15th 2011, the Regional Internet Registry for the Asian-Pacific region allocated the last IPv4 addresses available to them, followed by Europe in 2012 and Latin America and the Caribbean in 2014. With all of IPv4 gone, it is now inescapable to start switching to IPv6. [16] [43] [24] [143] [9]

2.2 Overview of Remote Communication Theory

This section will give a very short overview of the layered design of networking and known issues in communication theory. The goal is to introduce some core concepts and show a few of the problems that limit remote communication not because of lacking implementation, but for reasons of architecture, design or simply mathematics.

2.2.1 Layered Approach

In networking, the standard model used to describe how internetwork communication works is the Open System Interconnection model. For reasons of brevity, this approach will be skipped and TCP/IP, the de facto standard for internetwork communication, will be presented directly. Readers with knowledge of the Open System Interconnection model can reference Figure 2.2 for a quick overview of the relation between the two approaches. [143]

In modern networks, the communication between two hosts follows a layered architecture, with each layer only talking to the layer directly above or below it. A slightly different way to approach this is that each layer talks to its counterpart on the remote machine, that is to say that the Transport Layer on machine 1 communicates with the Transport Layer on machine 2. While

both views are correct, the first view offers a better representation of what happens technically, while the later offers a better representation of what happens on a conceptual level.

Through this approach, layers can easily be exchanged as long as the interfaces presented to the layers above and below remain stable. [143]

Building the stack from the bottom up, things start out with the Link Layer. This comprises the physical medium and all steps necessary to access it. One example of a Physical Layer would be Ethernet. [47] [28]

Moving further up next in line is the Internet Layer. This is where the Internet Protocol is situated. The Internet Layer is a connectionless layer only concerned with addressing and identifying hosts, and transmitting packets of data from their source to their destination by means of forwarding. It is also agnostic of data structures used by layers above and does not distinguish between various Transport Layer operations. Due to these facts, a multitude of Transport Layer protocols can be carried. Additionally the Internet Layer operates on a best-effort basis. It does not guarantee delivery of packets, order of delivery, or duplicate protection. [47] [109] [59]

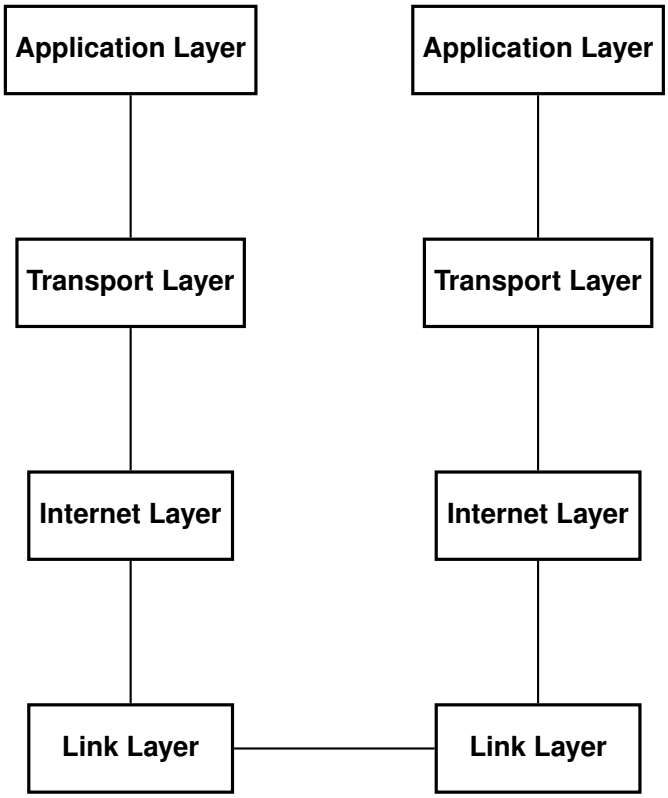

Figure 2.1: TCP/IP Protocol Stack – technical view

The Transport Layer is tasked with providing end-to-end services independent of data structures of Application Layer protocols. Basically this layer can be divided into two categories. Connection oriented, for example the Transmission Control Protocol (TCP), and connectionless,

for example the User Datagram Protocol (UDP). Depending on the chosen protocol this layer offers guarantees to the layer above concerning certain aspects of delivery like packet order and duplicate protection. There are other protocols working at this layer which are not transport oriented. This includes, among others, the Internet Control Message Protocol. Depending on the point of view, these protocols are at times attributed to the Internet Layer. [47] [111] [108] [110]

As topmost layer the Application Layer provides the most diversity. Almost everything not fitting into layers below can be found here. The range spans "simple" services like the Network Time Protocol as well as data intensive and highly complex interaction mechanisms like CORBA, which is a framework that allows remote method invocation on a variety of different platforms and in a multitude of programming languages. Of course things like the Hypertext Transfer Protocol also fall into this layer. [47] [61] [93] [122]

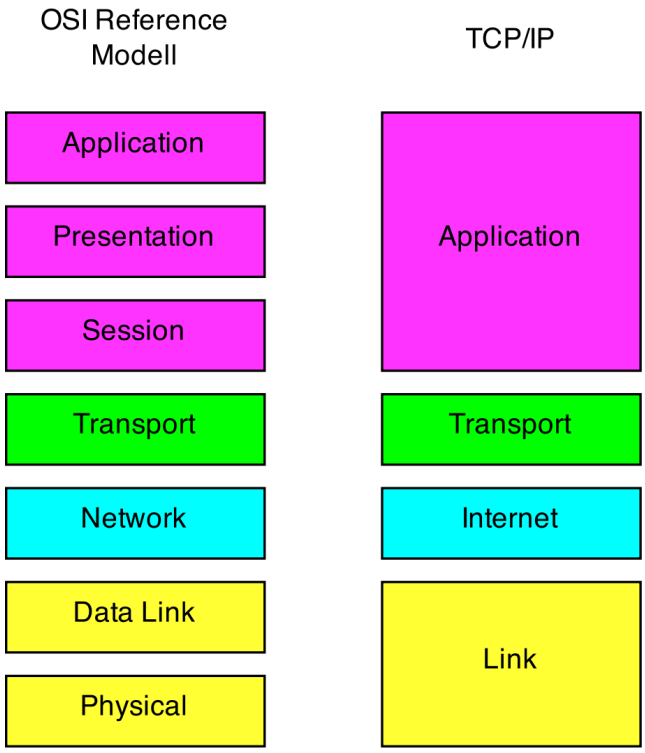

Figure 2.2: Equivalences between OSI and the TCP/IP Protocol Stack

2.2.2 Known Issues

At its core, network theory can be reduced to graph theory, which can be considered a rather old field. "Seven Bridges of Königsberg" by Leonhard Euler, published in 1736, can be considered the first paper on graph theory. Consequently, some problems found in this field can be addressed by proven mathematical methods, and bounds on the efficiency and correctness of employed algorithms can reliably be defined. Other problems of the field have also been studied

and while some problems remain open, many problems are at a stage at which at least rough estimates about solutions can be made. [130]

In the following paragraphs, three problems relevant to internetworking, especially on an Internet Protocol level, will be sketched out. The goal is to generate understanding as to what can and can not be done and thereby help grasping why some concepts in the Internet Protocol might seem strange at first.

2.2.2.1 Spanning Tree

A Spanning Tree is a connected, undirected graph which does not contain loops and connects all vertices. This construct is necessary in networks to prevent infinite loops of forwarded messages while maintaining a redundant layout. It is usually employed at the level of Link Layer devices, but the concept applies whenever a tree reaching all nodes is necessary. An example would be the control plane for router interaction in a hierarchical router deployment. This problem is solvable and solutions are well documented. [27] [134]

2.2.2.2 Shortest Path between Vertices

Shortest Path is a common problem in IP routing. Finding the fastest way between two nodes in a network is a question which is often not trivial to answer. In theory, approaches like Dijkstra's algorithm can solve this problem with decent worst case performance, but assigning weight to edges in a highly dynamic setting with constantly changing loads is a hard problem. In the reality of networking, Dijkstra's algorithm does not always yield globally optimal results. However, it is employed by many routing algorithms and has proven sufficient. [134] [91]

2.2.2.3 Two Generals Problem

The Two Generals Problem is a thought experiment, which demonstrates the problems and challenges of agreeing on a certain piece of information when communicating over an unreliable channel. There is no exact solution to this problem, in the sense, that the problem can be solved in all cases and with certainty. However, there are techniques which are employed to lower the risk of the message getting lost to an acceptable degree. For example, if duplicate addresses represent a serious problem within a network, duplicate address detection could be run multiple times, decreasing the likelihood of failure due to the nature of the communications channel. [135]

2.3 IPv6 - A Brief Introduction

The initial Request For Comments for IPv6 was released in December 1998 and numbered 2460. Thoughts about a successor to IPv4 have, however, been around for some time. RFC1883 issued in December 1995 gives a rough first draft of what IPv6 was supposed to look like. A Request For Comments is a document which contains technical and/or organisational notes about Internet-relevant topics, like concepts, protocols, procedures, meeting notes and opinions.

An index of Requests for Comments is maintained by the Internet Engineering Task Force. This section is going to give a short overview of the general architecture of IPv6, and about the major changes with respect to IPv4. Details relevant to exploits or possible threats will be given in the next chapter. [59] [51] [46]

2.3.1 Deployment Status

Before diving into technical details about IPv6, a short summary of its current usage and state of deployment seems appropriate. After all, the best protocol is worthless if no one uses it.

Depending on the metrics used to determine deployment status, results will differ wildly when answering the question on how far the world is on the way to global coverage. For this reason, 3 different metrics will be presented.

1. *How many autonomous systems, that is a collection of connected routing prefixes that present a common and clearly defined routing policy, have IPv6 enabled?*

   This question is hard to answer since no authoritative information on the total number of registered and currently active autonomous systems is available through the Internet Assigned Numbers Authority. [52]

   The most reliable resource for answering this questions are the statistics provided by RIPE, which is the European Regional Internet Registry. They state that as of the first of November 2014, 9056 out of 48760 autonomous systems world wide announce an IPv6 prefix, that is, had IPv6 enabled. This amounts to 18.57% of all autonomous systems. [29]

2. *How many of the Alexa Top 1000 Internet sites are currently available via IPv6?*

   This metric presents an interesting measurement of available content. As it is possible to argue that content drives the Web and that the Web drives the Internet, the question of what is actually available to IPv6-only users is of high relevance.

   Currently, around 14% of the Alexa Top 1000 Internet sites are available through IPv6. Therefore on the scale of available content, IPv6 fares slightly worse than on overall availability. [36]

3. *How much IPv6 traffic do we see at a large Internet exchange?*

   For this purpose, AMS-IX, which is the Amsterdam Internet Exchange, was chosen. They report a maximum throughput of 24 Gbps for IPv6 and 3.36 Tbps for IPv4 for the year 2014. The average for the same time period is reported as 13.9 Gbps for IPv6 and 1.69 Tbps for IPv4. So the percentage of IPv6 traffic flowing through AMS-IX in the year 2014 is 0.74% with respect to maximum throughput and 0.82% on average. [7]

These results, especially the one concerning traffic, show that there is still a long way to go until IPv6 can be considered widely used.

2.3.2 IPv6 Addressing Architecture

IPv6 completely redesigns the way addresses and address assignment are handled. This section aims at providing an overview of the different types of addresses used in IPv6 and how they interact with each other. After introducing the concept of scopes, the different address types and their uses will be discussed. In addition, some information on interface identifiers will be presented. All information provided in this section can be found in more depth and detail in [82] and its updates.

2.3.2.1 IPv6 Address Scopes

In IPv6, scopes are used to define the context in which an address is valid. The relevant standards currently define six zones and reserve two values. With the exception of the unspecified address (::) every IPv6 Address has a scope. In addition to scopes, addresses can also have a lifetime, which means they can be set to expire after a time interval or at a specific point in time. This enables life cycle management on an address level.

In the unicast class of addresses (see next section for details), all addresses which are not link-local, loopback or unique local addresses have "global" scope. This means they are globally routable and can be used to connect to any address. The mentioned exceptions have "link" scope, that is they can only be used to communicate with other nodes on the same physical link, and will, with the exception of unique local addresses under special circumstances, not be routed. [59] [82]

In the class of multicast addresses, the four least significant bits of the second octet determine the scope of a multicast address. This scope determines propagation ranges for messages to this multicast address and of course for the address itself.

Scope	Value
interface local	0x1
link local	0x2
admin local	0x4
site local	0x5
organisation local	0x8
global	0xe
reserved	0x0, 0xf

Table 2.1: Defined scopes for IPv6 Addresses

Scopes for the anycast address class are defined identical to those of the unicast address class. [82] [79] [106] [107]

2.3.2.2 Unicast Addresses

As in IPv4, a unicast address identifies a single interface and packets sent to a unicast address will be delivered to that interface. A unicast address, per definition referring to a single interface, may be assigned to multiple physical interfaces, if the implementation treats these as a single interface when presenting them to the Internet Layer. The intended purpose of this is for example load-sharing or elimination of a single point of failure.

An interface however will usually have more than one IPv6 address assigned to it. At least one link-local unicast address must be assigned, but as soon as more than simple local communication is required, more addresses on this interface are needed.

For example the ifconfig output for an IPv6 interface could look like this:

```
sixxs     Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet6 addr: 2001:15c0:65ff:6ec::2/64 Scope:Global
          inet6 addr: fe80::14c0:65ff:6ec:2/64 Scope:Link
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1280  Metric:1
          RX packets:3078 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1925 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:3532536 (3.5 MB)  TX bytes:225204 (225.2 KB)
```

Figure 2.3: A possible setup of an IPv6 interface.

A regular unicast address typically consists of 2 parts. The first 64 bit represent the network prefix used for routing, followed by a 64 bit interface identifier, used to identify the node on the network.

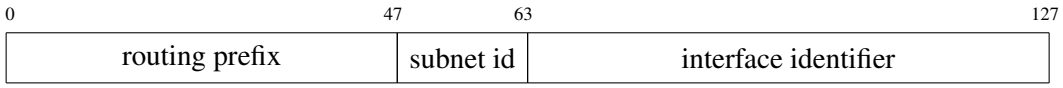

Figure 2.4: The layout of a regular unicast address

The routing prefix can be up to 64 bit while the subnet id can be as small as 0 bit, if no subnetting is performed.

A link-local unicast address consists of three parts: a prefix, a fixed value part and again an interface identifier.

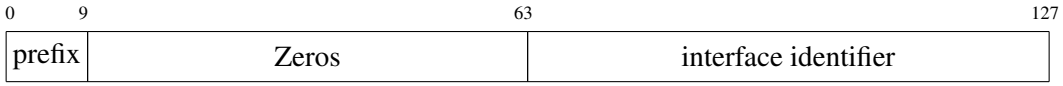

Figure 2.5: The layout of a link-local unicast address

The prefix is 1111111010. The combination of the prefix and the fixed amount of zeros yields the network prefix for all link-local addresses, fe80::/64. [59] [82]

2.3.2.3 Unique Local Addresses

This class of addresses can be viewed as conceptually similar to the IPv4 private addresses, in the sense, that they will generally not be routed outside of a well defined scope. On a router, messages from these addresses will not be forwarded to any link other than the one they originated from, unless the router has been specifically configured to do so. The idea behind this is to give organisations the ability to span multiple links with a common addressing schema which is not globally routable. Unique local addresses must be in the block of fc00::/7. [81]

Initially, the address block fec0::/10 was reserved for "site-local" use. However, due to insufficiently clear definition of the term site and the debate following this oversight, the range was deprecated in 2004. Approximately one year later, unique local addresses were introduced to solve the problem that should have been addressed by site-local addresses. [81] [75] [51] [71]

2.3.2.4 Anycast Addresses

Anycast addresses are indistinguishable from unicast addresses in IPv6. Typically, many interfaces distributed over multiple nodes will listen on this address, for example all Domain Name System servers.

A request sent to an anycast address will be forwarded to the "nearest" node listening to this address. Nearest in this context is determined by the routing protocols distance metric. The rational behind introducing this class of addresses is that in some cases a node needs information provided by a certain kind of server, but does not care who answers, and needs the answer only once. Again, the Domain Name System would be a prime example of this kind of behavioural pattern. [59] [82] [48] [60]

2.3.2.5 Multicast Addresses

In IPv6 multicast addresses are taken from the ff00::/8 range.

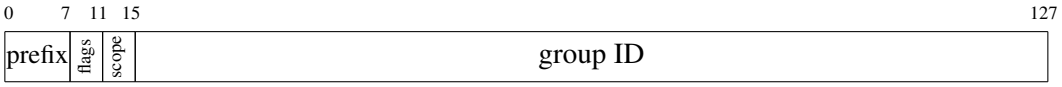

Figure 2.6: The layout of a multicast address

The prefix is 11111111. The flag bits are defined as follows:

Flag	0	1
Reserved	Reserved	Reserved
(R)endezvous	No rendezvous point embedded	Rendezvous point embedded
(P)refix	No prefix	Based on network prefix
(T)ransient	Multicast Address well known	Multicast address dynamically assigned

Table 2.2: The flags section of a multicast packet from most to least significant bit

Together with the scope discussed in Section 2.3.2.1 and a group ID, as defined by the Internet Assigned Numbers Authority, it is now possible to construct a meaningful multicast address. For example ff02::101 denotes all Network Time Protocol servers on the local link, while ff0e::101 would be all these servers which can be reached by the message.

One special multicast address worth noting is the Solicited-Node Multicast Address. This address is used in the process of neighbor discovery. They are constructed using a prefix and appending the low order 24 bits of the nodes unicast or anycast address. Each node is required to join all solicited node multicast addresses resulting from unicast or anycast addresses configured on its interfaces. The rational behind this will become clearer in the next chapter when the Neighbor Discovery Protocol is discussed. [59] [82] [76] [69] [31]

2.3.2.6 IPv6 Interface Identifiers

The interface identifier in IPv6 can be determined using multiple methods, some of which will be detailed in this section.

Modified EUI-64 is a 64 bit identifier most commonly derived from the Media Access Control address of the interface, which is a globally unique 48 bit address. This is done by splitting the address in half and inserting FF:FE into the middle. In addition, the 7th most significant bit of the resulting address is set to 1 indicating the global uniqueness of the resulting address.

Dynamic Host Configuration Protocol for IPv6 is a possible way to retrieve interface identification information over a network. This enables fixing identifiers to interfaces at a central point within the network enabling easy configuration of "fixed" addresses.

Randomly chosen was introduced to reduce the potential for long term eavesdropping. In this scenario, the interface identifier and therefore the node's address change over time. This method is commonly associated with Privacy Extensions. [88]

With all of the above methods there are restrictions on when and where they are allowed to be used and in what way. In general, modified EUI-64 addresses can be considered the default and/or fallback, which is acceptable in all situations from the standards point of view. [82] [103] [92] [88]

2.3.3 Stateless Address Autoconfiguration

This is an aspect of IPv6 designed to ease deployment. This section will describe the process from a host's point of view. The idea behind this mechanism is the following:

- No manual configuration of a machine is required before it is connected to a network.
- Small sites should be able to create networks able to communicate on the link without the deployment of a router or a Dynamic Host Configuration Protocol server.

- Nodes should be able to generate unique global addresses in large sites with multiple networks, even without the presence of a Dynamic Host Configuration Protocol server.

- Address renumbering should be facilitated in a graceful way.

To meet all these goals, at interface initialisation an IPv6 node generates a link-local address for this interface and performs duplicate address detection for this address.

If this fails, the node will need to be configured manually, otherwise from this point forward the node has access to all nodes on the same link.

The next step is trying to generate an address with global scope. To this end, the node tries to obtain a global prefix from a router. This happens on demand, in addition to periodical updates sent by routers, which nodes can use to verify and update their information. A router can supply zero or more global prefixes to a node, from which the node can then generate unicast addresses. Routers can also provide additional information like the address of a Dynamic Host Configuration Protocol or Domain Name System server or address lifetime information. [82] [87] [70]

Nodes should by default perform duplicate address detection on all addresses prior assigning them to an interface no matter where the address originated. This feature can be administratively disabled. [82] [87] [70]

Stateless Address Autoconfiguration can be used simultaneously with Dynamic Host Configuration Protocol version 6. This enables maximal flexibility, while maintaining manageability. [82] [87] [70]

2.3.4 Security Features in IPv6

This section is a short excursus into the world of Internet Protocol Security, a protocol originally developed for IPv6 and later adapted for use in IPv4. While Internet Protocol Security was intended to be mandatory for IPv6 nodes and was introduced that way, it has since been made optional. Internet Protocol Security and IPv6 interact through the use of two headers: the Authentication Header and the Encapsulating Security Payload header.

Internet Protocol Security provides functionality to authenticate and encrypt data on an IP packet level, offers mutual authentication of parties and negotiation of session keys. It can operate on connections established between a pair of hosts, a pair of networks or a network and a host. [59] [84] [99] [83] [48] [94]

2.3.4.1 Authentication Header

This header provides connectionless replay protection, data integrity and data origin authentication. In IPv6, it protects most of the base header with exception of mutable fields like hop limit, itself and non-mutable extension headers after itself, as well as the payload.

Payload Length denominates the length of the Authentication Header in 32 bit words minus two.

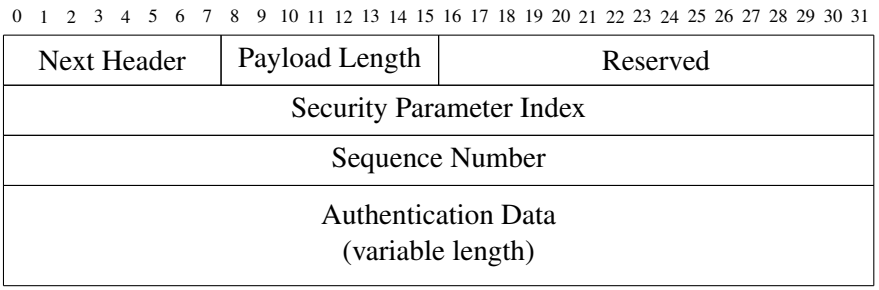

Figure 2.7: The layout of an Authentication Header

The *Security Parameter Index* is a random value which, together with the destination address and security protocol, in this case Authentication Header, is used to determine the Security Association. This is a set of shared security attributes like cryptographic algorithm and mode for this datagram.

The *Sequence Number* is a monotonically increasing number initialized to zero at establishment of a Security Association. The sender must always transmit this number, but the receiver may choose to ignore it, thereby disabling replay protection. The counter must not be allowed to cycle, Therefore, the connection must be reset before the 2^{32}nd packet is sent within a Security Association.

The *Authentication Data* field contains a variable length Integrity Check Value for the packet. The length must be an integral multiple of 32 bits. Allowed paddings and algorithms are defined in the standards and subject to regular change as the fields of cryptography and cryptanalysis evolve. [57] [105]

2.3.4.2 Encapsulating Security Payload

This part of Internet Protocol Security provides authenticity, integrity and confidentiality for packets. While supporting authentication-only and encryption-only modes, the use of these, especially of encryption without authentication, can pose significant security risks. [129] [133]

In contrast to the Authentication Header, integrity and authentication is only provided for parts of the IP packet, if transport mode is used. In tunnel mode, the packet is wrapped and a new packet is generated. Therefore, the complete inner packet is protected.

Security Parameter Index, *Sequence Number* and *Authentication Data* work in the same way as they would in an Authentication Header.

The *Next Header* field indicates the type of protected content within the payload field and *Pad Length* denotes the number of octets directly preceding it which contain padding.

The encryption algorithms used in combination with the Encapsulating Security Payload mechanism of Internet Protocol Security specify the exact layout of the payload field in their respective Requests For Comments, and may include data like initialisation vectors in this field for cryptographic synchronisation.

Authentication data covers everything up to the *Next Header* field and confidentiality is guaranteed for the payload data. [58] [105]

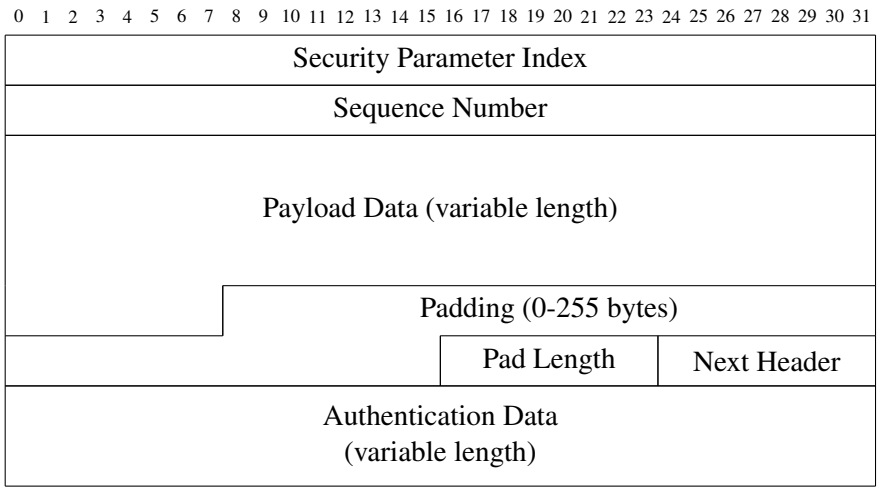

Figure 2.8: The layout of an Encapsulating Security Payload Header

2.4 Comparison of IPv6 and IPv4

In this section, a short comparison between IPv4 and IPv6 will be performed. This comparison is by no means complete, but rather focuses on key points relevant to this work.

2.4.1 Addresses and Address Space

When comparing IPv4 and IPv6 the first visible difference is the length and format of the addresses used by the two protocols. While IPv4 uses the well known decimal represented 32-bit address format, that is for example: 192.168.0.1, IPv6 uses a hexadecimal representation of 128 bits, for example: fe80::e246:ff93:ab16:2273. The :: indicates that the missing numbers in between are all 0. [59] [109]

Translating this difference into the number of available addresses, IPv4 yields a maximum of 2^{32} unique addresses, while IPv6 yields 2^{128} addresses. Calculating the number of available addresses, while providing a nice exercise in using a calculator, does little to increase the understanding of how huge that difference between IPv4 and IPv6 actually is. [59] [109]

One possible way of visualizing this is that if an IP address, regardless of whether it is IPv4 or IPv6, has a fixed size, and all IPv4 addresses together have the size of a golf ball, then all IPv6 addresses together would roughly have the size of our sun.

Depending on metrics, that is on what addresses of IPv6 are considered "dead" in the sense of unusable for uniquely addressing a node, IPv6 can be considered as between 10^7 and 10^{29} times bigger than IPv4. [22] [23] [96] [82]

2.4.2 Address Configuration

A key difference in the concept of address configuration between IPv4 and IPv6 is that in IPv4 an interface per default has one address. Configuring multiple addresses per interface will usually result in rather large configuration files defining "sub interfaces", which are virtual interfaces

on top of a physical interface, which in turn each have one IPv4 address. In IPv6, this form of complexity disappears. An IPv6 interface will usually have more than one address assigned, and most likely these addresses will have different scopes, as discussed in the previous section.

When moving from conceptual differences to hands-on issues it is known that the prevalent method of managing address configuration in IPv4 networks today is Dynamic Host Configuration Protocol, which is a standardized way of distributing network configuration parameters using a client-server model. Alternatively, manual static configuration can be used, but as expected this approach does not scale well in large networks. [55]

IPv6 offers more ways of configuring addresses. Besides manual static configuration and Dynamic Host Configuration Protocol Version 6, IPv6 offers Stateless Address Autoconfiguration. Additional to using each of these approaches on its own, they can be combined to create complex multi-tiered configuration solutions. [87]

For example, a network administrator could configure a server with a static global unicast address, have all link-local configuration handled via Stateless Address Autoconfiguration, while configuring nameservers, which are used for IP Address to human readable resource identifier translation and vice versa, via Dynamic Host Configuration.

2.4.3 Internet Control Message Protocol

While in IPv4 the Internet Control Message Protocol is mainly used for diagnostic purposes and can usually be completely filtered by a firewall at a network boundary without any adverse affect on the networks operations, this does not hold true for IPv6. [25] [109]

Internet Control Message Protocol Version 6 is vital to the operations of an IPv6 network as can easily be seen in the following examples.

2.4.3.1 Path MTU Discovery

In IPv4 fragmentation, the process of splitting a large package into smaller ones due to size limits on the path between endpoints, happened “on-the-fly”, transparent to the endpoints of a connection, unless their own physical layer required the fragmentation to take place. In IPv6, the endpoints of a connection are responsible for fragmentation. In IPv4, if an intermediate node received a packet it could not forward without fragmentation, it fragmented the packet. An IPv6 node would send a control message type 2 to the packets originator, indicating that the packet is too big for this path. The originator then has multiple ways to proceed, as defined in [53], but must ultimately send smaller packets.

2.4.3.2 Neighbour Discovery

In IPv4, discovering nodes on the same link was handled using the Address Resolution Protocol, which basically uses table lookups and broadcasts at the Link Layer, to map IP Addresses to physical addresses. In IPv6, this functionality is provided and extended by the Neigh-

bor Discovery Protocol, which uses control messages of type 133 through 137 to achieve its goals. [86] [112]

2.4.4 Header Structure

Although a regular IPv6 header is at least twice as long as a regular IPv4 header, the processing necessary to forward a package is generally more efficient. This takes load off routers and allows for better use of existing hardware for on-route packet handling. Efficiency is further increased by extending the end-to-end principle of the Internet, which has been somewhat lost in IPv4. [59] [49] [68] [93]

For a quick comparison, the following graphics show the two headers side by side and colour code the differences. Red fields have been removed. Cyan fields have been kept in the same position with the same name. Yellow fields have been renamed or placed at a different position, and green fields are new in IPv6.

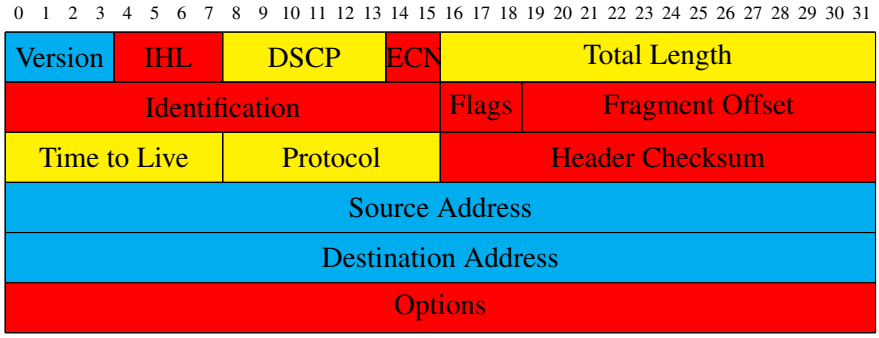

Figure 2.9: IPv4 Header

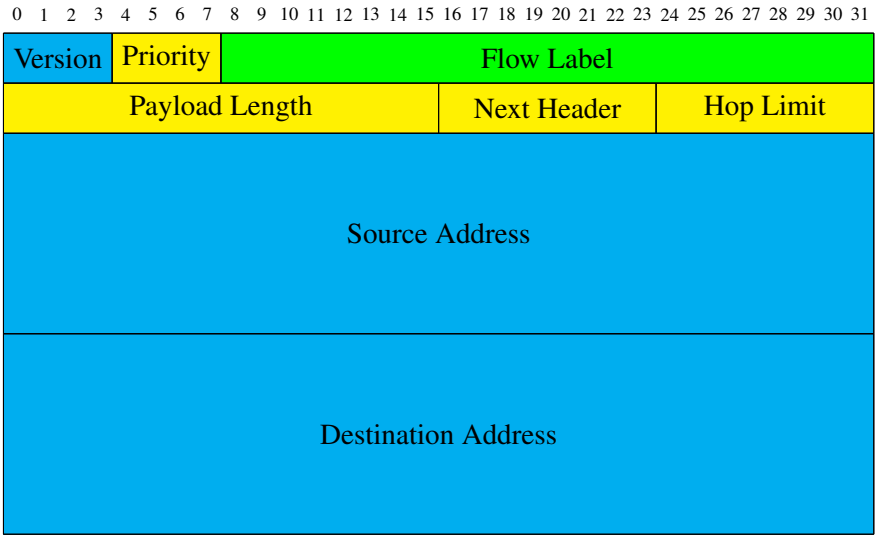

Figure 2.10: IPv6 Header

Since all information concerning fragmentation, security and other "extension" features have been moved to their own header, the main header has been greatly simplified. A node not concerned with extended features is not required to parse those headers and can simply pass the packet on "as is" after inspecting just this first header.

2.4.5 Multicast

While IPv4 offered the ability to broadcast a message to all nodes on a network/subnetwork, IPv6 has eliminated broadcasting from its set of features. Instead, it focuses more on multicast and offers a new functionality called anycast which, in contrast to multicast, only results in an answer from one of the receiving nodes. While multicasting in IPv4 is commonly implemented, it is not required. [109]

The more common use of multicast, for example to retrieve information from all routers on a subnet, together with the new address schema, allows for the use of globally routable multicast groups which are source specific. This feature has the ability to greatly impact network and asset management within companies, while possibly posing a security risk due to information disclosure vulnerabilities. [69]

CHAPTER 3

Exploiting the Internetworking Layer

This chapter will provide the theoretical basis of the attacks employed in the laboratory part of this thesis. It is split in to a section detailing exploit mechanics that applied to IPv4 and are still relevant in IPv6 and a section going into detail on issues new to IPv6.

In general, attacks on the internetworking layer can be divided into two categories. Attacks that originate in the local network and attacks that originate in a remote network. For the first class of attacks, the level of entry is high. In order to initiate these attacks, physical access to the network and, in some cases, an authentication token is required. For the second class of attacks, these hurdles do not exist. All that is needed is access to a network that can deliver packets to the target. This is one reason, why network borders are usually well secured. Routers, Firewalls and Intrusion Prevention Systems all aim at keeping attackers out of a network.

3.1 Attacks on IPv4 with Relevance to IPv6

In this section, well known attacks and vulnerabilities will be revisited. In this process, an outline of their mechanics and underlying concepts will help in judging their relevance to IPv6. Where applicable, real world exploit references are used to demonstrate the impact of exploitation. In addition, an attempt will be made to classify the exploited mechanics into one of two categories. Vulnerabilities are either conceptual, based on the underlying standards and designs, or practical, indicating a flaw in the interpretation of the standards or in the actual implementation.

3.1.1 Smurf Attacks

A smurf attack is a distributed denial of service attack. In this scenario, two factors greatly influence the effectiveness of the attack. The packet multiplication rate and the data multiplication rate. That is how many packets of traffic are generated per packet sent and how many units of data are generated per unit of data sent, respectively. While the first puts stress on the network interface of the victim, the second can also generate significant load on other system components, depending on how data is treated.

Classically, a smurf attack is performed by sending a response eliciting Internet Control Message to a network's broadcast address. The source address field of this packet is spoofed by the attacker to contain the address of the victim. By default most network devices will respond to such messages, if they originate within the network. In this scenario, packet multiplication depends on the number of hosts connected to a network. Data multiplication can usually be ignored with the common Internet Control Messages like Echo Request, because the response is not much bigger than the initial packet.

To mitigate this threat, a two factor approach is utilized. First, all hosts on the network are configured to suppress responses to control messages with a broadcast destination address. Then, all routers are configured not to forward packets with a broadcast destination. In addition to these two approaches, which are best used in combination, network ingress filtering can be used to further decrease the attacker's chances of success. [62] [65] [138]

Smurf attacks can be separated into two groups, local and remote. In a local scenario, the attacker is located in the same subnet as the victim. Therefore, no routers are involved in handling the traffic and only defenses at a host level take effect.

Remote smurf attacks can leverage multiple networks as amplifiers, and the victim can be located in any subnet. For example, an attacker discovers that the networks 10.0.1.0/24 through 10.0.5.0/24 are susceptible to remote smurf attacks. The victim is located at 10.0.6.3/24. All that is left to do is send an Echo Request to the vulnerable networks and set the source address to that of the victim. Since the broadcast domain ends at the router, this attack should not be viable in real life, but there have been examples of this attack working.

In IPv6, the landscape is slightly different. There is no broadcast address. Nevertheless, IPv6 relies on multicast to provide similar functionality. In a local context, the addresses FF02::1 and FF02::2, pointing to all nodes and all routes on the link-local scope, are of particular interest for these kinds of attack.

An attacker could therefore send an Echo Request to the local all-nodes address spoofing the source address to any valid address and trigger a response from all nodes not configured to ignore such packets.

Exploiting such vulnerabilities remotely is almost impossible in IPv6 due to the way scopes and multicast domains are defined. There were however erroneous implementations of the IPv6 stack allowing some hosts to respond to an Echo Reply originating from a local multicast address. For details see the section on toolkits.

In most modern operating systems, the implementations of the IPv6 stack are sufficiently advanced to just drop packets that would trigger the vulnerability. [141]

3.1.2 Fragmentation Attacks

Fragmentation in networking describes the process of splitting a datagram into two or more smaller datagrams. This is done in order to accommodate different physical layers which might impose different restrictions on the maximal size of a datagram. [109]

For example, while Ethernet version two has a maximum transmission unit (MTU) of 1500 bytes, wireless local area networks can transport up to 7981 bytes per unit. This means that a datagram with a size of 2000 bytes could be sent through a wireless-only infrastructure without

the use of fragmentation, whereas it would have to be fragmented if a part of its route through the network was based on Ethernet. [28] [26]

Before going into detail on how fragments can be used to attack a host or network, one should consider that fragmentation can also be used as an evasion tool, hiding an attack on higher level protocols. This is enabled by two main factors. Different operating systems have varying implementations of the IP stack leading to different treatment of fragmented packets, for example in reassembly. Therefore, if an intrusion detection system runs on a different operating system than the host under attack, it might not be able to detect the event due to the afore mentioned differences in network stack implementation. In addition to this, different types of networking devices have diverging capacities of understanding and evaluating fragments. The capabilities of a router will usually differ from those of a firewall or intrusion detection system when fragmentation is concerned. This can be used to target specific parts of the infrastructure without touching or alerting others. Solutions for both problems exist, either in the form of host based intrusion detection systems or via the use of proxy servers and combinations thereof. These mitigation approaches, however, can be very expensive or hard to manage, for example in large networks with huge amounts of traffic.

In the next paragraphs, some fragmentation based exploits will be presented in detail.

3.1.2.1 Fragment Overlap Attacks

These attacks are based on the fact that, when fragment offsets indicate that two fragments overlap, one fragment overwrites a part of the other fragment. This can be as little as a bit, or as much as the complete packet. Since operating systems can be very diverse in the way they handle reassembly, this technique can be used to evade intrusion detection systems, hiding patterns of malicious code by padding them with harmless data. Another application would be a direct attack on the reassembly code, for example an attack known as “Teardrop attack” which triggered a vulnerability in the TCP/IP stack of some operating systems leading to a denial of service on the network side of the operating system. [121] [50]

3.1.2.2 Fragment Overrun Attacks

This type of attack tries to exploit that IP datagrams have a maximum size as defined in the relevant Requests For Comments. By sending fragments that are larger than this maximum size when reassembled, the attacker tries to exploit possible weaknesses in the victim’s TCP/IP stack.

3.1.2.3 Too Many Datagrams Attacks

These attacks are similar to the fragment overrun in the mechanism that is applied, but differ in target. In this instance, the target is the data structure holding the not yet reassembled packets. By sending a large amount of fragments an attacker can try to overflow this data structure.

Fragmentation in IPv6 is end-to-end as compared to IPv4 where it is done at intermediate nodes. This changes the mechanics but not the concepts behind these attacks. In addition,

implementation differences are likely not going to disappear just because the protocol stack is exchanged. Therefore one can assume that fragmentation attacks will remain a valid concern in IPv6 networks and might even gain momentum, considering that intermediate nodes are not obliged to run any form of checks on fragmented traffic.

3.1.3 Fuzzing

Fuzzing is strictly speaking not a class of attacks, but rather a testing technique used to discover faults in a system or program.

In fuzz testing the input mechanism of a system is connected to a generator. This generator yields invalid, random or unexpected data while the system under test is monitored for exceptions or unwanted side effects such as memory leakage which happens when memory assigned to an application is not properly released by that application after it is not used any more. [137]

Naïve fuzz testing is rather easy to implement and once automated does not require a large amount of human resources. It can therefore provide a valuable tool in finding entry points for in-depth code reviews. Due to its inherent randomness it can also find rather odd defects that even an overly diligent tester would miss because of their obscure nature.

The major downsides of fuzzing include that, with a naïve approach, most of the time only very simple faults are discovered. In addition, boundary values, which are the values that restrict accepted input, are often not tested or not tested well with this technique due to their randomness. Furthermore, code coverage, which is the percentage of code of a program actually executed during a test run, can be very low. [142]

Still, the benefits of this testing technique outweigh the drawbacks in a security context. Especially since the alternatives are either very time consuming, as for example binary or source code review, or simply do not yield results applicable to real life scenarios, as can be the case with fault injection, where a fault is artificially induced. [146] [131]

3.2 Newly Rising Issues in IPv6

This section will provide information about vulnerabilities new to IPv6. It will focus on the underlying concepts and design choices which lead to exploitable behaviour. Implementation specific faults will be mentioned where applicable. The three subsections will deal with the topics of headers, neighbor discovery and secure neighbor discovery. This is of course non exhaustive for the problem space, but these three categories cover vital parts of the protocol and have already seen their share of exploits, making them prime subjects for discussion.

3.2.1 Headers

As has already been discussed, IPv6 breaks backward compatibility to IPv4 not only by the massive change in addressing, but also by changing the layout of the IP header. Extension headers are introduced, all of which have specific purposes and are designed with a subset of nodes in mind which will act on them. [59] In this section, some of the new header types will be presented and their security scrutinized on the design level.

3.2.1.1 Routing

The Routing Header can be used to specify nodes on the path to the target node. These nodes are to be "visited" by the packet during its journey through the network.

A routing header is indicated by a Next Header value of 43. Initially, only type 0, source route, was defined. This option was later deprecated. The details of this header and the circumstances leading to its deprecation will be discussed later.

Type 1 was introduced and than deprecated. It should have served as a binding component to the Nimrod project, which aimed at specifying and implementing a scalable internetwork routing architecture. [54] This project, however, never reached the implementation stage.

Types 2 and 3 are currently in use and will be warranted a closer inspection. [97] [100]

The layout of a routing header as defined in the Request For Comments 2460 is visualized in Figure 3.1.

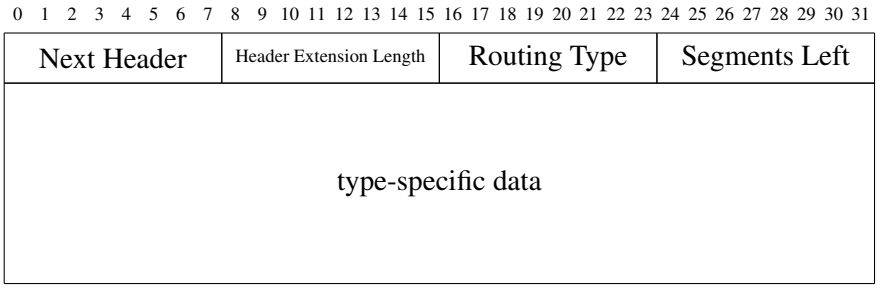

Figure 3.1: Layout of a Routing Header

Header Extension Length is the length of this header excluding the first eight octets. *Segments Left* specifies the number of explicitly listed intermediate nodes still en route before the packet reaches its final destination. [59]

Routing Header Type 0 is the functional analogue of IPv4's source route option. Conceptually, this type of routing header can contain an array of addresses of arbitrary length and the packet will be routed to every node in this array. This can be seen in Figure 3.2.

It is also allowed to include a specific address more than once. This functionality creates a vector of attack enabling a malicious user to construct a packet which will oscillate between two routing header type 0 enabled devices. [59] [89]

This can be used to amplify traffic along a route, possibly leading to congestion on this path and subsequent denial-of-service for all nodes solely connected through this link. Up to 88-fold amplification of traffic has been reported in a laboratory setting. [32]

The high severity rating and attention this attack received is largely due to the fact that this attack does not only affect a relatively small amount of nodes but a whole path. In contrast, the IPv4 version of this attack is less severe since it does not support the chaining of as many intermediate nodes.

These factors led to the total deprecation of this type of header in December 2007. [89]

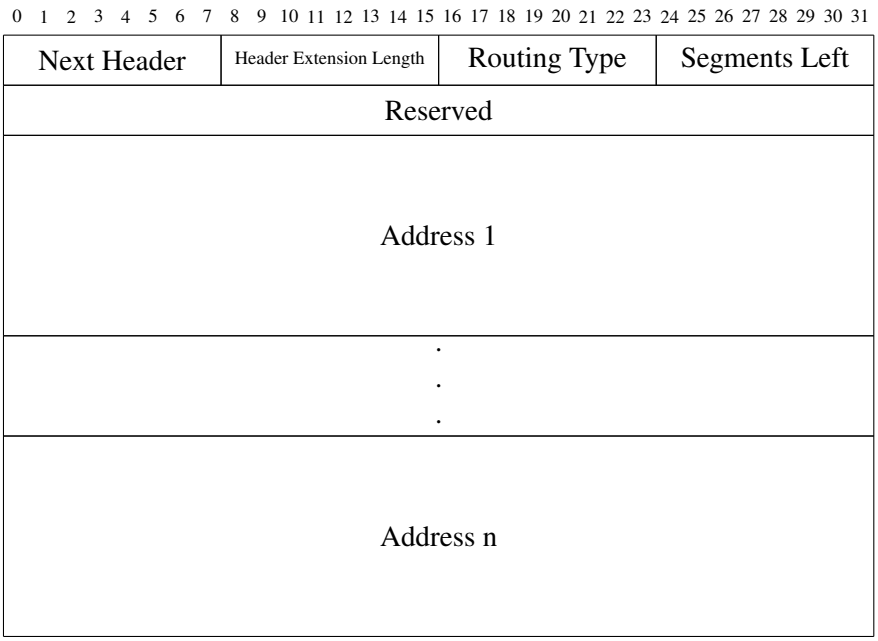

Figure 3.2: Layout of a Routing Header Type 0

Routing Header Type 2 is a part of the mobile portion of IPv6. A full exploration of the mobile functionality is out of the scope of this work. Due to this limitation only the most relevant concepts needed to understand this routing header will be described here.

The intended functionality of this routing header is to facilitate direct forwarding of a packet from a correspondent to the care-of address of a mobile device. A correspondent can be any node, fixed or mobile, in the network communicating with the mobile node. The care-of address is the temporary address of the mobile device indicating its current location. [97]

Considering the security impact of this header, multiple aspects have to be taken into account.

The aspects shared by all routing headers, for example the risk of bypassing IP-based firewall rules, are of course applicable to this header. They do, however, not increase the potential harm above a threshold of acceptability, if the functionality is desired.

Moving to more specific threats, reflection attacks are still possible, but this also is inherent to the use of routing headers. The amplification present in type 0 routing headers, however, is mitigated by limiting the length of the list of addresses to one. [97]

With regard to these points, it can be concluded that the use of type 2 routing headers does not incur a greater than necessary risk for the provided functionality, in the sense that all risks incurred are inherent to the technology employed.

Routing Header Type 3 was introduced for Low-Power and Lossy Networks. Nodes in these kinds of networks often exhibit strong constraints on memory and computation power. This entails that in routers the number of routes that can be kept in memory is small, especially when compared to standard routers.

Type 3 routing headers were designed to circumvent the need of storing the whole routing table on all routers. The idea is to enable a single router, or a small subset of all routers, to use source routing techniques to direct package flows. [100]

In order to mitigate security issues similar to those in the type 0, routing header multiple steps were taken. The protocol specification requires routers to drop packets containing this header at routing domain boundaries. This mitigates remote triggering of attacks associated with this header. However, attacks can still be mounted from inside the network.

To minimize the possibility of bandwidth exhaustion, routers are required to check for loops in addresses in the header and drop datagrams where loops are present. [100]

Type 3 routing headers can therefore be considered to introduce no risk not inherently introduced by this technology.

In general, routing headers and the concept of source routing provide relevant performance and cost benefits for specific use cases. Given the correct usage of mitigation techniques, like loop checking and in- and egress filtering, they can be considered safe, in the sense of acceptable risk.

3.2.1.2 Destination Options Header

As the name suggests, this header carries information that is only relevant for the destination node or nodes. Therefore, this header needs not be examined by intermediate nodes. As before, the *Header Extension Length* is an eight bit unsigned integer value.

The *Options* field contains one or more type-length-value encoded options. The complete header must be an integer multiple of eight octets long. [59]

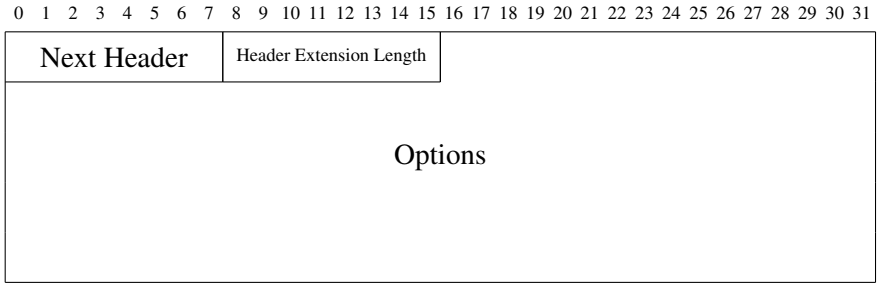

Figure 3.3: Layout of a Destination Options Header

Type-length-value encoding is used for this and the Hop-By-Hop header. In addition to the *Option Data*, a variable length field, it contains the *Option Type*, an eight bit identifier, and the *Option Data Length*, an eight bit unsigned integer denominating the length of the Option Data field in octets. [59]

Figure 3.4: Type-Length-Value Format

The highest order bits of the option type field encode the action to be taken if the destination option is not recognised by the processing node. These actions range from silently dropping the packet to sending a Parameter Problem message even if the originating address is a multicast address. The third highest order bit specifies if the Option Data can change in transit. This influences the behaviour of software that calculates checksums for the packet, for example the part of the stack processing Authentication Headers. [59]

The IPv6 specification currently defines 2 options, namely Pad1 and PadN. These options are used to align options defined in different Requests For Comments, should alignment be necessary. [59]

Pad1 inserts one octet of zeros into the option header. It is a special case since it has neither a length nor a value field. The use of multiple Pad1's is discouraged. Instead PadN should be used. [59]

PadN is used to pad with two ore more octets of zeros. To pad with N octets, the Option Data field contains N-2 octets of zeros. [59]

Additional options, such as Jumbo Payload and Home Address, have been defined in various additional Requests For Comments and will be discussed as needed.

Aside from attacks on separately defined options, exhaustion attacks could be attempted by chaining padding to exceed memory restrictions. However, these can easily be mitigated by receiver side checks and rate limiting.

Given that the sending node defines how erroneous packets should be handled, the potential for traffic multiplication has to be considered.

Assuming that an attacker sent a crafted packet containing a malformed Destination Options header to a multicast address, specifying that on error a Parameter Problem message should be returned to the originator, then the victim would receive as many messages as there are members in that multicast group.

Since multicast addresses are scoped, this form of attacks can easily be limited to the local scope by applying in- and egress filtering at network boundaries. For example, by filtering all incoming traffic from multicast addresses if the dataflow was not established by the destination address of the packet.

Within the local scope, the severity of this kind of attack depends on the size of the established multicast groups, but does not exceed the potential of other attacks reflecting traffic off multicast groups. Since there is no way of creating an amplification loop no additional risk is added.

It can therefore be concluded that through this option no additional risk is incurred.

3.2.1.3 Hop-By-Hop Options Header

This header is very similar to the Destination Options header. The layout and options defined in the IPv6 specification are identical. The only major difference is the set of processing nodes.

While destination options are only parsed at the destination, options of this header are parsed by every node along the delivery path. [59]

Besides increasing load on the route, this header in combination with a Routing Header could be used to target specific nodes in a network, under the assumption that a known subset of nodes is vulnerable to an attack.

Router Alert represents an option defined in a separate Request For Comments. The purpose of this option is to inform all routers along a path that processing of layers above the Internet Protocol Layer of the packet is necessary. [64]

This is desired for protocols like the Resource Reservation Protocol, which inform nodes ahead of time of resource requirements of a dataflow. It is designed to provide a robust scalable mechanism for resource allocation in an integrated service network. [56]

Without this option a router would have to, at least partially, parse upper layer information for all packets to achieve functional equivalence. Behaviour of this sort would lead to a significant decrease in router performance.

In a nutshell, Router Alert gives external entities access to change parameters usually reserved to individuals with control plane access to the network. This opens up a host of possible security issues with various degrees of severity. A selection will be presented here. Further reading and an in depth discussion of most threats can be found in [80] and [98].

Some of the more obvious attack vectors of this option are associated with denial of service. This can be achieved via multiple paths.

Modern routers conceptually have two handling lanes for packets, the slow and the fast track. Packets in the fast track are, most of the time, handled mostly or completely in hardware, that is layer three information is parsed, a routing decision made and the packet forwarded. Slow track packets require processing in software. This significantly increases the time from packet ingress to packet egress and the packet has to be kept in memory much longer.

Depending on the router in place, most if not all Router Alert packets will be pushed into the slow track. This can lead to an exhaustion of resources on the targeted router. [98]

Alternatively, an attacker could try allocating resources on the path to an extent that renders the path unusable.

For both vectors simple rate limiting as suggested in [64] seems inappropriate, as the granularity of limiting must be very precise to allow attack mitigation without degrading performance. [98]

While a simple limit for maximum allocatable resources could mitigate the second scenario, defending against scenario one does not seem trivial.

Possible solutions could include the selective tunnelling of packets containing Router Alerts in order to prevent external attacks on a transit network, or the filtering within an administrative domain. [98]

In summary, this option has a huge potential for abuse and mitigation seems to be non-trivial.

3.2.2 Neighbor Discovery

The Neighbor Discovery mechanism in IPv6 provides a large amount of functionality. Most of it vital for correct operation of the network.
Among other things it encompasses:

- Address auto configuration
- Link Layer address discovery of nodes on the link
- Duplicate address detection
- Detection of nodes present on the link
- Router discovery
- Domain Name System server discovery
- Reachability information provisioning
- Prefix discovery

In a nutshell, it replaces, improves and extends IPv4's Address Resolution Protocol as well as the Internet Control Message Protocol version 4 Router Discovery and Router Redirect messages. [86] [109]

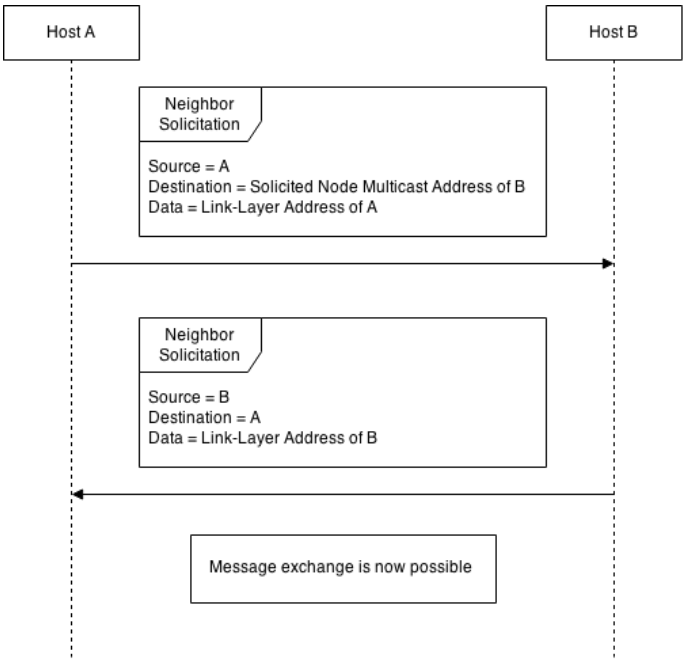

Figure 3.5: Regular address discovery between two nodes.

Figure 3.5 shows a standard address exchange between two hosts.

To achieve this, five control message types are defined, which are detailed in the following paragraphs.

Router Solicitation messages are typically sent to the all-routers multicast address. The source address can be any address assigned to the sending interface, or the unspecified address (::/128). Unless the source is the unspecified address the Link Layer address of the originating interface should be included as an option. Upon receiving such a message a router will immediately generate a Router Advertisement packet. [86]

Router Advertisement messages are generated by routers periodically or in response to a Router Solicitation message.

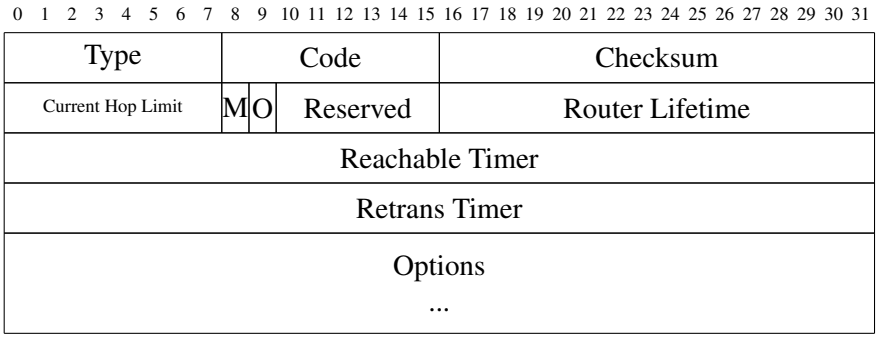

Figure 3.6: Format of a Router Advertisement message

These packets contain a wealth of information about the network. The *M* or managed bit indicates if addresses are managed by a Dynamic Host Configuration Protocol server. The *O* or other flag signals that other information, for example the addresses of Domain Name System servers are also available via dynamic configuration.

The *Router Lifetime* field contains an unsigned 16-bit integer and is used to indicate the default router's lifetime in seconds. This mechanism determines the usefulness of a router as default router. It does not impose a validity period or restriction on the information of the packet.

Reachable Time sets the time in milliseconds the router is to be considered reachable after a reachability confirmation, which is part of the Neighbor Unreachability Detection discussed later.

A *Retrans Timer* field is used to publish the time between retransmission of Neighbor Solicitation messages in milliseconds.

Options for this type of message include Link Layer source address, link Maximum Transmission Unit and prefix information. [86]

Neighbor Solicitation messages are sent to a node to either verify or determine its Link Layer address. In the latter case, the packet is sent to a multicast address. The Link Layer address of the originating node must be transmitted if the destination is a multicast address. [86]

Neighbor Advertisement messages are either sent in response to a solicitation or unsolicited message. The latter can be used as an unreliable channel for fast propagation of information through the network.

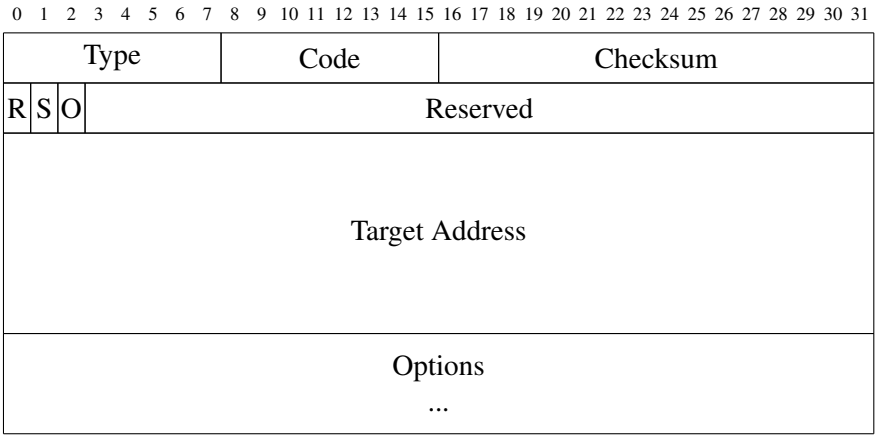

Figure 3.7: Format of a Neighbor Advertisement message

Typically, messages of this type are addressed to the originator of the solicitation or to the all-nodes multicast address.

The *R* flag is set if the originating node is a router, while the *S* flag indicates that this packet was generated in response to a solicitation. The *O* flag is used to signal that information contained within this advertisement should override existing cache entries. If not set, recipients of this message that already have an entry for this node in their Neighbor Cache, will not update their entry, but nodes that do not have such an entry will add one with the information contained in this packet.

Note that if the solicitation that triggered the generation of this packet was sent to a multicast address the Link Layer address of the responding node must be contained in the answer. Otherwise an infinite recursion could occur since the peer nodes do not have the necessary entries in their Neighbor Cache. [86]

Redirect packets are used by routers to inform nodes that a better first hop to their destination is available. To that end, they include the *Destination Address* of the packet triggering this redirect and the *Target Address* of the first-hop preferable to the originator. This mechanism can also be used to indicate that the target is actually on the same link. This is achieved by setting Target Address and destination to the same value. [86]

Options are again presented in a type-length-value style. Detailed information on the precise formats of all options can be found in [86].

Neighbor Unreachable Detection is one of the new features introduced by IPv6. It improves packet delivery efficiency of a network or partitions of a network in case of router failure, by keeping track of reachability information. In addition, it improves communication with nodes that have changing Link Layer addresses and minimizes loss of packets sent to mobile nodes.

Depending on the type of connection and the role of the neighbor, different tactics are applied should reachability be lost. For example, if the node in question is a router, an attempt to switch to a different router might be appropriate, whereas if the node is the final destination of the packet, reperforming address resolution might yield the desired result.

State information on the reachability of a neighbor is kept in the Neighbor Cache and updated in one of two ways, either by sending a Neighbor Solicitation message or by gleaning information from an upper-layer protocol. For example, acknowledgements in a Transmission Control Protocol connection confirm that the node in question received the data previously sent. When possible, upper-layer information should be used, since this method incurs no additional cost. [86]

Considering the extent of the protocol, a large range of possible threats has to be examined. For reasons of simplicity only a general overview will be provided here. Details relevant to this work will be discussed where the need arises.

The main threats with relevance to this protocol can be divided into three categories.

- Denial Of Service
  For example, a node could prevent other nodes from joining a network by blocking their

address configuration. This can be achieved by replying to every solicitation sent by that node, indicating that all addresses on the network are taken.

- Address Spoofing
  Can be performed in a similar way to old IPv4 attacks on the Address Resolution Protocol.

- Router Spoofing
  An attacker could advertise his node as router and send spoofed Router Advertisements for all legitimate routers, indicating that they are not to be used. As a consequence, all packets, including link-local packets, would have to travel through the rouge router. Neighbor Unreachable Detection has no way to detect such blackholing if the attacker correctly answers all solicitation messages.

All these threats are discussed in great detail in [72].

It can be concluded that the Neighbor Discovery Protocol has a multitude of security issues, some of which can be mitigated easily, while others are buried deep in the concepts employed. The original specification of the protocol relied on IPSec to close the gaps and handle security, but as this is not an option in modern days IPv6 stacks' alternatives had to be found. To this end Secure Neighbor Discovery was developed. [86] [74]

3.2.3 Secure Neighbor Discovery

As a response to the infeasibility of securing the Neighbor Discovery Protocol through mechanisms of IPSec, a different approach had to be proposed. Secure Neighbor Discovery aims at providing mechanisms to prove address ownership, authenticate messages and establish trust anchors and certification paths. All of this without relying on IPSec. [77]

A huge real life obstacle for the use of IPSec as a means of securing neighbor discovery is that a multitude of nodes simply do not implement IPSec as part of their IPv6 stack although it was mandatory when the standard was created. In response to the wide spread behaviour of the industry the Internet Engineering Task Force changed the requirement status of IPSec in IPv6 nodes from mandatory to optional. [99]

In order to secure the neighbor discovery process, three major components are defined. A mechanism to prove ownership of an address via public-private key pairs without the use of a public key infrastructure, certification paths with common trust anchors between nodes to ensure the legitimacy of routers and a replay protection mechanism using timestamps, nonces and signatures. Each of these building blocks will be described in detail in the next paragraphs. [77]

For the transition period between the unsecured and the secured version of neighbor discovery, the standard lays down detailed rules which can be summarized as follows: Nodes using Secure Neighbor Discovery trust information they gain through secure means more than that gained via normal neighbor discovery and fall back to unsecured information only if no secured information is available. All messages that come through the secured protocol but fail the

verification process will either be discarded or treated as unsecured, depending on the context. Nodes only using the unsecured version of the protocol ignore all unknown options by default. Therefore, secure discovery does not interfere with their operation. [77] [86]

3.2.3.1 Address Ownership

In order to establish a binding between an IPv6 address and a node, the concept of Cryptographically Generated Addresses is employed. [77]

Cryptographically Generated Addresses are generated by replacing up to the 64 least significant bits of the IPv6 address with a hash value. In order to generate this value, three parameters are used. [78]

- The public key of the node generating the address. This key is generated by the node. As there is no outside infrastructure involved, this key is random in the sense that there is no definite binding of key to entity. Consequences of this are discussed later in this chapter.
- A 64-bit subnet prefix for which the address is generated.
- A security parameter, which is a three bit unsigned integer with a value range from 0 to 7. The cost of generating a new Cryptographically Generated Address depends exponentially on this parameter. It is also used to determine the strength of the generated address against brute force attacks.

From these parameters the data structure in Figure 3.8 is generated. [78]

The *Modifier* is an arbitrary 128-bit unsigned integer used to implement the hash extension performed during address generation. It aims at improving privacy by adding randomness. The *Collision Count* is an eight bit unsigned integer which must be either 0, 1 or 2. It is incremented during address generation to recover from potential collisions detected by the Duplicate Address Detection mechanism. The *Public Key* contains the variable length DER-encoded ASN.1 structure of the public key of the key-pair used in this address generation. This should be an RSA key of at least 384 bits. The *Extension Fields* are currently unused. [78]

After construction of this data structure, two hash values are generated using the SHA-1 algorithm. [78]

Hash1 is generated using the data structure as input. After calculation of the hash, the left-most 64 bit of the 160 bit hash are taken and the rest is discarded. This hash is used to generate the address to be used by the node. To this end, the first three bits are replaced by the security parameter and the "u" and "g" bits reserved by the IPv6 Addressing Architecture standard are set to 0, yielding the final cryptographically generated address. [78]

Hash2 is generated by calculating the SHA-1 hash over the same data structure with collision count and subnet prefix initialized to zeros. The left-most 112 bit are taken and again the remaining bits are discarded. This hash has to satisfy the condition that the first $16 *$

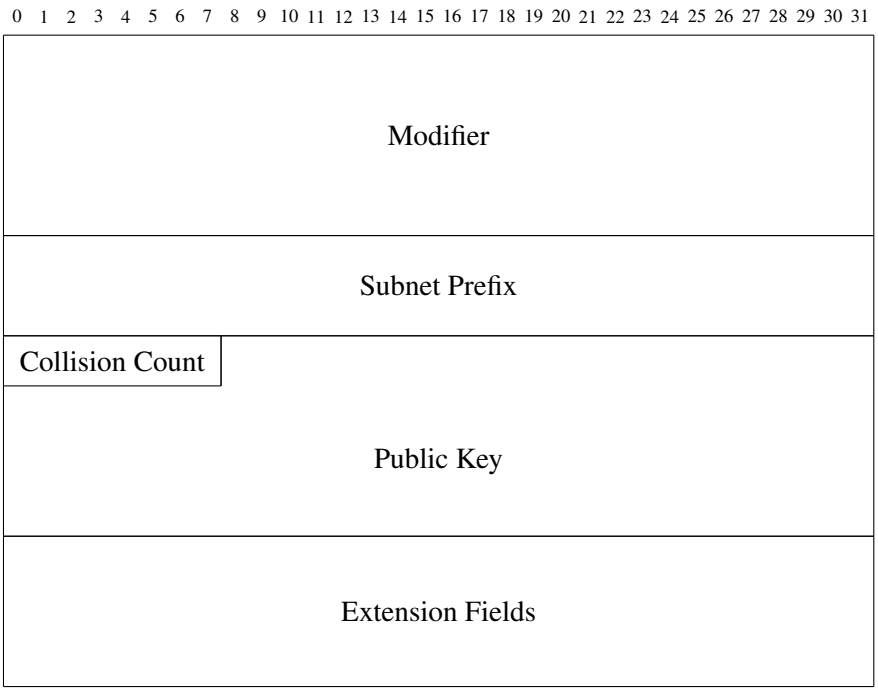

Figure 3.8: Data structure used by Cryptographically Generated Addresses

$security\_parameter$ bits have to be zero. This can for example be achieved by incrementing the modifier and calculating the hash until the condition is met. [78]

After the generation process is completed, the data structure is added as an option defined in the Secure Neighbor Discovery standard to all Neighbor Solicitation/Advertisement messages as well as to Router Solicitation messages unless they are sent from the unspecified address. Messages are then signed using the RSA-Option detailed later in this section. [78]

By calculating the hash values and comparing Hash1 with the address and checking if Hash2, the subnet prefix and the collision count conform to specified values, a node can establish that a public key is valid for and bound to a cryptographically generated address. This knowledge can then be used to validate signatures for this address. [77] [78]

Note that this only establishes that the node in question holds a key-pair valid for this address. No other information about the identity of the node can be gained via this mechanism.

The RSA Signature Option can be considered the workhorse of the Secure Neighbor Discovery Protocol. It allows the attachment of public key based signatures to messages used in neighbor discovery. If using the Secure Neighbor Discovery Protocol, Neighbor Solicitation/Advertisement, Router Advertisement and Redirect messages must contain this option. The option can be omitted in Router Solicitation messages, if and only if the source address is the unspecified address. [77]

Figure 3.9 shows the RSA Signature Option where *Key Hash* are the most significant 128 bits of the SHA-1 hash of the public key used for this signature. This key can be known to the

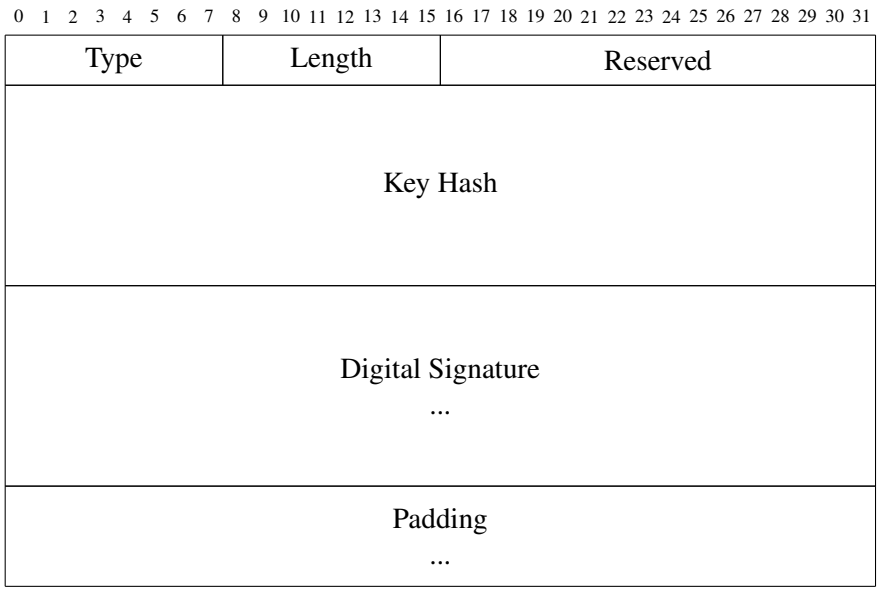

Figure 3.9: Layout of the RSA Signature Option

receiver either through an entry in its signature cache, or if the sender is new to the network through the Cryptographically Generated Address option in the same message. [77]

The *Digital Signature* is a variable length field. It contains the PKCS#1 version 1.5 signature generated with the sender's private key. PKCS#1 is a family of standards providing definitions of and recommendations for the implementation of the RSA Algorithm as well as mathematical properties of keys and operations used in the process of encrypting or signing. [41] [77]

The signature itself is constructed over these octets: [77]

- The 128 bit message type tag of Cryptographically Generated Addresses.
- The source address of the packet
- The destination address of the packet
- The 8 bit Type and Code fields as well as the 16 bit Checksum field of the Internet Control Message Protocol header of the packet
- The Neighbor Discovery Protocol header starting at the octet after the Checksum field of the control message header up to but not including the neighbor discovery options
- All Neighbor Discovery Protocol options preceding this option

A receiver of a message signed this way must ignore any options after the first RSA Signature Option. If a trust anchor model, as defined in the next section, is used, a valid path between the sender's key and the trust anchor must exist. However, if the receiving node is not configured to use a trust anchor, the verification can be limited to the Cryptographically Generated Address approach, even if the sender has used a trust anchor. [77]

Receiving nodes may drop packets containing RSA Signatures silently for any number of reasons, including an apparent attack aiming to exhaust the nodes resources. [77]

3.2.3.2 Router Legitimacy

The Secure Neighbor Discovery Protocol defines a basic mechanism to authorize nodes as routers and grant them the right to advertise subnet prefixes. This is achieved via a public key based approach similar to the one applied on the Web for SSL secured connections. The major difference is that certification paths can initially not be verified by the node that wants to establish trust since it can not communicate off-link. The following paragraphs will provide an overview of the required provisioning and infrastructure as well as insights into the workings of this part of Secure Neighbor Discovery. [77]

To legitimize a node as router, a node newly connecting to a link checks if the router candidate can provide a valid certificate. A certificate is considered valid if the node performing certification can construct a path from the provided certificate to a trust root. [77]

Trust roots can either be implemented via a globally rooted public key infrastructure or through a more decentralized approach in which each node is preconfigured with a set of public keys corresponding to its trust roots. As an example, a node could be configured with keys from its own organisation, the user's Internet Service Provider and all Internet Service Providers that have roaming agreements with the user's home Internet Service Provider. Due to the backward compatibility of the Secure Neighbor Discovery Protocol, nodes would still function in networks not using router authentication. [77]

Certificates can provide two kinds of authorisation: [77]

- A node is authorized to act as router. It belongs to a set, with possibly only one entry, of trusted routers and all routers within this set have the same authorisation.

- As an option, a router may be authorized to advertise a set of subnet prefixes, while other routers are authorized to advertise different, but possibly intersecting, subnet prefix sets.

Other parameters transmitted from the router to a host like the router's own address or prefix lifetimes are not covered by this mechanism. [77]

Certificates used in Secure Neighbor Discovery can be used for authorisation in other contexts, but great care should be taken if that is the case. Among other things, it has to be guaranteed that the provided authorization information is appropriate for all processes and that no unwanted side effects can occur. As an example, if a certificate is used for authorization in both Secure Neighbor Discovery and a routing protocol, the null prefix (::/0), used in secure discovery to indicate that a router might advertise any prefix, could lead to routing or verification problems. [77]

The Certificate Format used to authorize routers is defined by the X.509 standard version 3. A certificate should contain at least one extension of such a certificate as defined in the Request For Comments 3779. These extensions allow for the binding of a block or range of Internet Protocol addresses and/or Autonomous System numbers to a certificate. [77] [73] [90]

In order to construct a certification path the node trying to be authorized as a router sends a certificate chain to the newly connected node. This certification path is then, after initial checks concerning the "well-formedness", checked against trust anchors known to the node. Should these checks succeed, the node accepts the router. If subnet prefixes are authorized as well, the subnets in the certification path must have a subset relationship from anchor to router, that is the subnets of a certificate farther away from the anchor must be a subset of the subnets of the certificates closer to the anchor. [77]

If no such path can be established, the node can send a Certification Path Solicitation Message, see Figure 3.10, to the node trying to authorize itself as a router.

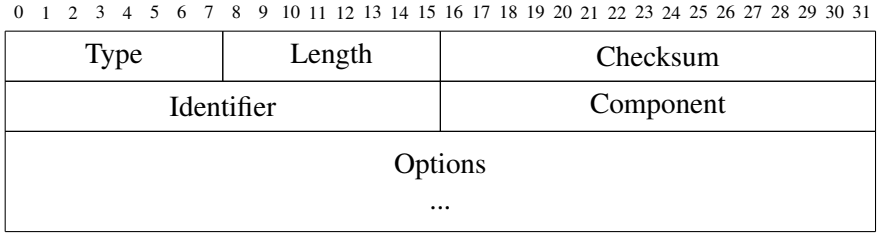

Figure 3.10: Layout of a Certification Path Solicitation message

Where the *Identifier* should be a random number. It is used to match solicitations to advertisements. *Component* denotes the element of the path needed by the sender of the packet. If *Component* is set to 65535 the sender needs the complete path. [77]

The only currently valid option is the Trust Anchor Option, in which the sender specifies one or more trust anchors it is willing to accept. [77]

Usually this message is sent either to the All-Routers multicast address, the Solicited-Node multicast address or the hosts default router via unicast. [77]

Upon receiving such a message, the recipient will try constructing a certification path between itself and at least one of the trust anchors specified in the solicitation and respond using a Certification Path Advertisement as depicted in Figure 3.11. [77]

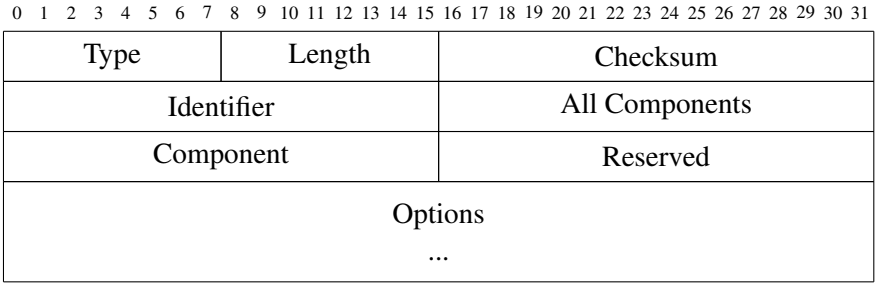

Figure 3.11: Layout of a Certification Path Advertisement message

All Components and *Component* are used to order packets on the path, since every certificate should be sent in a separate packet to avoid extensive fragmentation at the Internet Protocol layer. [77]

Currently, two options are defined for this message type:

- Trust Anchor is used to help the recipients of the message determine the usefulness for them and can be omitted.
- Certificate contains one certificate per option. The certificate of the trust anchor should not be transmitted.

This mechanism, called Authorization Delegation Discovery, circumvents the need for nodes to go off-link through a possibly malicious router by having the router prove itself via public key cryptography. [77]

3.2.3.3 Replay Protection

In order to protect the process of neighbor discovery against replay attacks, these are attacks in which pre-recorded information of a legitimate transaction can be used to circumvent protection mechanisms, Secure Neighbor Discovery introduces two new options. [77]

The Timestamp option is used to secure unsolicited messages, for example advertisements and redirects. [77]

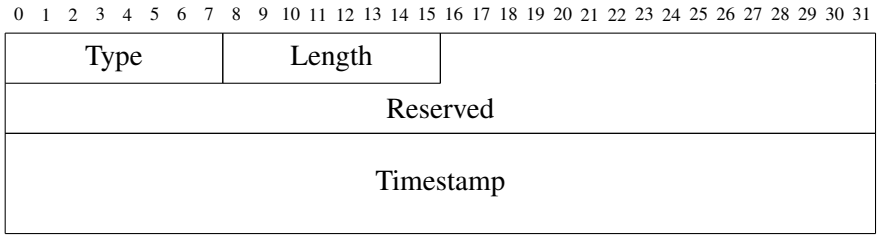

Figure 3.12: Timestamp Option

It contains the timestamp as a 64-bit unsigned integer value, indicating the seconds since the 1^{st} of January, 1970, 00:00 UTC. The format used is a fixed point format where the first 48 bits indicate full seconds and the remaining bits represent $1/64000$ fractions of a second. [77]

The Nonce option is used to secure answers to all solicited messages, with the possible exception of responses sent to multicast address. In the latter case, responding nodes may include the nonce or a timestamp, but including only a timestamp might result in the soliciting node ignoring the answer, if the clocks between the two nodes are not sufficiently synchronized. [77]

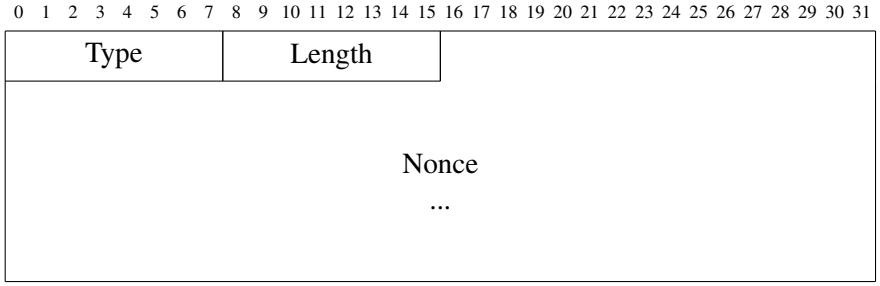

Figure 3.13: Nonce Option

The *Nonce Field* itself contains a random number selected by the sender of the solicitation. This length of the option must be a multiple of eight octets and the nonce itself must be at least six bytes long. [77]

3.2.3.4 Security Of This Protocol

The topic of security for this protocol is twofold. On one hand the obvious question is whether or not it succeeds in mitigating the threats present in the unsecured version of Neighbor Discovery. On the other hand it is well known in the computer science community that fixing a problem can introduce an arbitrary number of new ones. Therefore, possible new vectors introduced by Secure Neighbor Discovery have to be taken into account.

Fixing security problems of the Neighbor Discovery Protocol is the main purpose of this protocol and the standard gives an in detail overview of the threats mitigated by the use of Secure Neighbor Discovery, as well as the mechanisms countering the specific threats. In summary, most of the threats covered by [72] are addressed and countered. However, not everything is mitigated. Threats not addressed by this protocol include: [77]

- Brute fore and (Distributed) Denial Of Service attacks on routers.
- Router compromisation.
- Neighbor Discovery Denial Of Service, by flooding a router with crafted packets forcing it to perform discovery on a large amount of addresses.

While the first two points are of a more general nature and can be considered out-of-scope for this protocol, the third can easily be mitigated by rate limiting the discovery process. [77]

New issues arising with this protocol mostly fall into the category of resource exhaustion. Various safeguards are recommended, most of them including the monitoring and limiting of resource usage, especially in routers. For example, it is recommended that, if a large amount of Authorisation Delegation Discovery messages is received, a router should limit the number of discoveries it is engaged in and, by caching results for trust paths, decrease the work required to establish a path. [77]

In addition to these vectors, the security of Cryptographically Generated Addresses has to be taken into account. The short key length used makes these addresses vulnerable to integer-factoring attacks, that is the decomposition of a number into its non trivial divisors, but should currently provide enough protection against most Denial Of Service attacks. These and similar issues are discussed in depth in [78] and are generally concerned with the relative weakness of the employed cryptographic methods given the used parameters. [78]

It can be concluded that the Secure Neighbor Discovery Protocol is a huge leap forward from the unsecured version in the areas it addresses. Most of the issues introduced can be mitigated by simple means of upscaling of parameters for cryptographic functions, should the need arise.

In summary, this chapter shows that the attack surface has shifted when comparing IPv4 to IPv6. While some vectors have been added, others have disappeared. However, there is no evidence that there is a significant difference in the overall vulnerability between the protocols on the conceptual level.

CHAPTER 4

Tool Suites

This chapter consists of two sections. The first part deals with currently available toolsets for assessing the security situation of IPv6 implementations and networks. It will take a closer look at different dimensions of these suites, which will include functionality, ease of setup, documentation and perceived usefulness. In the second section, some components of the previously discussed toolsets will be selected and used to build a (semi-)automated testing tool for the laboratory part of this work. It will present details on the process of choosing exploits and tests as well as an in-depth review of how the chosen tests and exploits are implemented in the toolset. The chapter will then close by detailing the problems encountered along the road to (semi-)automation.

4.1 IPv6 Toolkit

The IPv6 Toolkit by SI6 Networks was first released on August, 24th 2013 with the current version being 1.5.3. According to their homepage, this collection of tools aims to be a general purpose security assessment, trouble shooting and resilience testing suite. They also claim to offer the most comprehensive scanning tool currently available for IPv6. It contains 13 tools for a variety of purposes, which will be described in the next pages. [18] [118]

tcp6 is a tool that can perform a huge variety of TCP based attacks. It does so by utilizing its ability to send arbitrary TCP segments.

rs6 can send router solicitation messages of arbitrary content to any connected network or host.

na6 sends crafted neighbor advertisement messages to the host or network being assessed.

addr6 analyses IPv6 addresses. It can identify the scope/type of an address as well as infer the type of Interface ID used by that address. Additionally, it can produce statistics for a set of IPv6 addresses.

ni6 allows assessment of nodes and networks with respect to node information messages.

ns6 will craft neighbor solicitation messages.

rd6 performs attacks based on Internet Control Message Protocol redirect messages.

jumbo6 is a tool in this suite used to test a node's implementation of jumbogram handling. A jumbogram is a packet carrying a payload larger than the protocols Maximum Transmission Unit. For IPv6 that is a datagram larger than 65535 octets. [63]
In contrast to other tools in this suite it offers no passive mode.

flow6 performs tests on the targets flow label generation policy.

icmp6 can send a multitude of Internet Control Message Protocol messages and error messages.

scan6 scans an IPv6 network using a number of advanced scanning techniques.

frag6 can be leveraged to assess IPv6 fragmentation capabilities and vulnerabilities of a node.

ra6 can send router advertisement messages of arbitrary content to any connected network, or host.

The information provided above was taken from the README included with the software, its set of manual pages, the corresponding help messages and the homepage of the toolset. [118] [136]

The toolset is easy to set up and ready to use in short time, using the provided makefile.

It provides very high flexibility and countless options for every tool. This very fine granularity of control over the final on-the-wire packet can make it a great tool for diagnosing highly sophisticated problems. It can also enable very subtle attacks to be performed with a high degree of precision and without the need to write separate tools for the job. This of course comes at the price of a seemingly cluttered command line and complex handling, resulting in a steep learning curve.

This being said, the documentation of this toolset is extensive and detailed. Each tool has a separate manual page detailing every option, sometimes down to implementation level. Usage examples are not only provided but explained and use cases are presented with most of them. At times Requests For Comments are referenced or other material for further reading is linked.

Scan6 is the most powerful feature of the IPv6 Toolkit. It provides multiple sophisticated scanning techniques to discover hosts in local and remote networks. [37] [104] [101]

The possible issues with this set of tools are twofold. On one hand, the high flexibility comes at the price of a complex interface, the impact of which is reduced by the extensive documentation. Still, the initial learning phase can prove rather time consuming. On the other hand, while selecting tools for the lab, the default behaviour of the tools, like tests run without options provided, seemed more of an afterthought than a planned feature.

In summary, this toolset is a highly flexible, well documented trouble shooting and security testing suite with the ability to perform advanced tests and attacks with a surprising amount of precision. It is very well suited for node discovery and is therefore a valuable asset in any test. Finding unknown problems or vulnerabilities with this toolset on the other hand is problematic. However, if the problem or vulnerability is specific and in a well defined section of the network or protocol, this toolset provides the means to create solutions and exploits without much overhead.

4.2 THC-IPV6

The THC-IPV6 describes itself as:

" A complete toolset to attack the inherent protocol weaknesses of IPV6 and ICMP6,[...]" [123]

The publicly available version 2.5 of this set of tools contains 67 programs designed to test various aspects of IPv6 in hosts as well as routers, tunnelling devices and firewalls. The next pages will give an overview of the functionality of THC-IPV6. The focus of the overview will rest on the tools used in the laboratory part of this thesis.

trace6 This is a simple tool that traces the route from one host to another.

covert_send6 This tool sends a given file inside a destination header.

fragmentation6 is a tool that performs implementation checks and attacks on host and firewalls between hosts. In total, 33 attacks and tests are performed, ranging from simple duplicate frames, via overlapping frames to multiple denial of service attacks like fragments with "holes" in between. That is a fragment that states its contents end at for example byte 10 and the following fragment claims to start at byte 12 of the reassembled payload.

randicmp6 sends all combinations for ICMP headers. This takes a long time to complete. Additionally, the most important basics of this tool are covered by fuzz_ip6.

redirsniff6 implants a route into the victim, which redirects all traffic to a given destination to a desired IP-Address. This is done to all traffic that flows through the victim that matches victim $\rightarrow$ target. The router which would handle the route must be known to make the handover. Additionally, wildcards can be used for victim or destination, in which case the tool will listen to network traffic and pick a host from that information. If the new-router/ -mac does not exist, this results in a denial of service.

dnsrevenum6 performs a fast reverse domain name system enumeration and is able to cope with slow servers. In an IPv6 context this can be very helpful for reconnaissance purposes.

dump_router6 dumps a list of all local routes and the information about them retrievable via router solicitation messages.

extract_networks6.sh prints all the networks found in a specified file.

fake_solicitate6 solicits an IPv6 address on the network, sending it to the all-nodes multicast address. This might be used to deny service to or black hole a single node. Black holing is a technique which could be leveraged in more advanced attack/defence scenarios. It is the process of silently dropping packets for a destination. Neither originator nor destination are informed that the packet is discarded. [67]

flood_solicitate6 floods the network with neighbor solicitation messages.

passive_discovery6 passively sniffs the network and dumps all client IPv6 addresses detected. Note that in a switched environment better results are obtained by additionally starting parasite6. This, however, will adversely impact the performance of the network.

flood_mld26 floods multicast listener discovery version 2 messages.

dnsdict6 enumerates a domain for domain name system entries. It uses a dictionary file if supplied or a built-in list otherwise. This tool is based on dnsmap by gnucitizen.org. [19]

fake_advertise6 advertises an IPv6 address on the network with the sender's media access control address as link-layer address if no other behaviour is specified. This advertisement is sent to the all-nodes multicast address if no target address is set. The address advertised is the node's address if no different address is specified.

ndpexhaust26 floods the target /64 network with TooBig error messages. This tool version is a lot more efficient than ndpexhaust6.

covert_send6d writes covertly received content to file. This is the counter part to covert_send6.

fake_mld6 advertises, adds or deletes a node in a multicast group. It can also query for listeners to a multicast address.

dos-new-ip6 prevents new IPv6 interfaces from coming up by sending answers to duplicate address checks (Duplicate Address Detection). This results in a denial of service for new IPv6 devices.

implementation6d identifies test packets sent by the implementation6 tool. It is useful to check which packets passed a firewall.

fake_router6 announces a node as a router and tries to become the default router. If a non-existing link-local or mac address is supplied, this results in a denial of service.

toobig6 implants the specified Maximum Transmission Unit into the path to the target.

sendpeesmp6 sends secure neighbor solicitation messages forcing the target to verify a lot of cryptographically generated addresses and RSA signatures. This version of the tool uses multiple processes for message generation.

fake_mld26 uses the Multicast Listener Discovery version 2 protocol but offers the same basic functionality as fake_mld6. Only a subset of what the protocol is able to do is implementable via the command line. Advanced features are available via the toolsets packet creation library.

fuzz_dhcps6 is a fuzzing tool to test Dynamic Host Configuration Protocol servers acting on the IPv6 network of an interface.

kill_router6 announces that a target router is going down to delete it from the routing tables. If you supply a '*' as router-address, this tool will sniff the network for any router announcement packets and immediately send the kill packet.

flood_rs6 floods the local network with Router Solicitation packets.

sendpees6 sends secure neighbor solicitation messages forcing the target to verify a lot of cryptographically generated addresses and RSA signatures.

flood_router6 floods the local network with router advertisements. Fragment, destination and hop-by-hop headers can be added to bypass router advertisement security guards. [95]

detect-new-ip6 detects new IPv6 addresses joining the local network. If a script is supplied, it is executed with the detected IPv6 address as first and the interface as second command line option.

exploit6 is a tool that tries to exploit the target host with multiple IPv6 vulnerabilities published on the Common Vulnerability and Exposure database maintained by Mitre. Common Vulnerabilities and Exposures is a dictionary of publicly known security vulnerabilities and exposures focusing on information systems. [11]

fake_dhcps6 is a tool to fake Dynamic Host Configuration Protocol version 6 servers.

alive6 is a simple host enumeration tool.

flood_redir6 floods the local network with Redirect Messages. Fragment, destination and hop-by-hop headers can be added to bypass simple filters.

node_query6 sends an Node Query message to the target and dumps the replies to the standard output.

implementation6 is a tool that tests for various header implementation details. It does so by sending multiple headers or header sequences followed by an echo request and determines standard conform behaviour based on the reply.

parasite6 is an "Address Resolution Protocol spoofer" for IPv6, redirecting all local traffic to the attackers own system by answering falsely to Neighbor Solicitation messages.

fake_router26 announces the host it is called on or a given host as a router and attempts to establish it as the default router. If a non-existing link-local or mac address is supplied, this results in a denial of service.

detect_sniffer6 tests if systems on the local LAN are sniffing. Works against Windows, Linux, OS/X and *BSD. If no target is given, the link-local-all-nodes address is used, which however does not always works.

6to4test.sh tests if the IPv4 target has a dynamic 6to4 tunnel active.

fake_mldrouter6 announces, deletes or solicits a multicast listener device router.

fuzz_ip6 is a fuzzing tool able to fuzz the fields of a multitude of Internet Control Message Protocol messages such as neighbor or router advertisements. Additionally, it can fuzz fields in headers like fragmentation, router alert and destination.

inverse_lookup6 performs an inverse address query, to get the IPv6 addresses that are assigned to a MAC address. Note that only few systems support this yet.

rsmurf6 smurfs the local network of the victim. This depends on an implementation error, currently only verified on some versions of Linux. Targeting "ff02::1" will result in a complete denial of service for the target LAN.

firewall6 performs various access control list bypass attempts to check implementations. It defaults to using TCP ports, but provides an option to switch to UDP. For all test cases to work, Echo Request/Reply messages to the destination must be allowed.

flood_advertise6 floods the local network with neighbor advertisements.

flood_router26 floods the local network with router advertisements. Each packet contains 25 prefix and route entries.

thc-ipv6-setup.sh is a shorthand for the most used settings in the context of this toolset.

redir6 implants a route into the victim, which redirects all traffic addressed to a defined node to the desired address. This is done to all traffic that flows through the victim that matches victim $\rightarrow$ target. The router which would handle the route must be known to make the handover. Additionally, a hop-limit can be specified. If the new-router/-mac does not exist, this results in a denial of service.

fake_dns6d is a fake Domain Name System server that serves the same IPv6 address to any lookup request. You can use this together with parasite6 if clients have a fixed Domain Name System server. This is a very simple server. It does not implement multiple queries in a packet, or "complex" lookups like MX or NS.

fake_mipv6 sends a fake mobile IPv6 message to the mobile IPv6 home agent. If the agent is configured to accept these messages without the use of IPSec, all packets for the defined victim will be delivered to the care-of-address of the attackers choice.

denial6 performs various denial of service attacks on a target. This, naturally, generates huge loads on the target host, which could cause it to crash.

ndpexhaust6 randomly sends Echo Requests to IP Addresses in the target network.

flood_mld6 floods the local network with Multicast Listener Discovery reports.

dnssecwalk tries to perform a zone transfer and checks for inconsistencies in the database.

smurf6 smurfs the target with Echo Replies. The target of the Echo Requests is the link-local all-nodes multicast address if no other address is specified.

four2six sends an IPv4 packet to an IPv6 4to6 gateway. If a port is specified, a UDP packet is sent to that port, otherwise a version 4 Echo Request is sent.

flood_dhcpc6 is a dynamic host configuration client flooder. It can be used to deplete the IP address pool a server is offering. If the pool is very large, this is a rather useless endeavour.

fake_pim6 sends fake protocol independent multicast packets. It can send hello commands with an optional designated router priority, as well as *join* and *prune* commands. Join and prune need a multicast group, a target address and the address of neighboring protocol independent multicast routers.

fake_dnsupdate6 sends a spoofed dns update to a specified host.

thcsyn6 floods the target address/port with TCP-SYN packets. If "x" is supplied as port, it will be randomized.

flood_mldrouter6 floods the local network with Multicast Listener Discovery router advertisements.

dos_mld.sh implements a denial of service attack on the multicast infrastructure. A multicast address can be specified, which will be targeted first. In time all multicast traffic will stop.

dump_dhcp6 is a Dynamic Host Configuration Protocol information tool. It dumps a list of available servers and their setup.

inject_alive6 is a tool which answers to keep-alive requests on Point-to-Point over Ethernet and 6in4 tunnels. For Point-to-Point over Ethernet it also sends keep-alive requests.

address6 converts a Media Access Control or IPv4 address to an IPv6 address and defaults to link-local if no prefix is given. When given an IPv6 address, it prints the corresponding Media Access Control or IPv4 address if possible. If there are multiple results, all will be displayed.

thcping6 crafts an Internet Control Message Protocol/Transmission Control Protocol/User Datagram Protocol packet with user defined IPv6 or extension header options.

extract_hosts6.sh prints the host parts of IPv6 addresses found in a file.

The information about these tools is taken from the manual pages delivered with them, their help messages, the README included with the distribution and experimentation done during the course of the lab. [139]

During testing, the toolset presented itself as an extensive base for performing security analysis on an IPv6 network, of which the attacker has little to no previous knowledge. It offers a large amount of tools, which can help with reconnaissance and information gathering as well as "ready to go" attacks on hosts and infrastructure devices as well as services.

The included packet generation library, at first glance, enables a versed programmer to easily write the tools needed to perform more complex tasks or build a new tool on the fly. However, due to time constraints an in-depth analysis of this functionality is outside of the scope of this thesis. Should closer inspection of the library at a later point in time show that it is insufficient for the tasks at hand, there are a multitude of other packet generators around, which can replace its function in an attack scenario. One example would be Scapy, a flexible packet manipulation, generation and analysis tool [114].

The greatest drawback of THC-IPv6 is most likely the lack of a well structured documentation. The manual pages provide a nice general summary, and the help texts give hints on how to use the tools. However, detailed information on attacks used must be taken either from the source code of the tool or from a network dump.

Still, installation of the toolset proves to be very straight forward and setting up an attacking computer can be done in a very short amount of time.

In summary, this is a well crafted set of attack and reconnaissance tools that offer ready to use solutions for a multitude of situations. Even disregarding the packet generation library, the amount and quality of out-of-the-box functionality makes this toolset invaluable to any security test on a network level.

4.3 Test Suite

In order to manage the testing activities and homogenize the testing process and result reporting, the tool sets described in the first part of this chapter were examined and a subset of tests and exploits was selected. These were then automatized using *fabric*, which is a python framework mainly used for application deployment and automation of administrative tasks on local or remote systems. [14]

In this section, the process of selecting tests and exploits will be documented, the problems that arose in this process will be discussed, and details of the inner workings of the chosen tests and exploits will be summarized.

For the setup used during this part of the laboratory, please refer to Section 5.1. Once all exploits were chosen and the automation tool was finished, automated tests were performed.

The results of the tests as well as their discussion can be found in Chapter 6.

4.4 Choosing Exploits

To make an informed choice of tests and test parameters as well as to gain experience with the used tool, the first step was reviewing all available documentation, including possibly relevant presentations, blogs, mailing lists and news groups.

Aside from documentation, which has already been mentioned in the previous section, there are, at the time of writing, no relevant sources of interactive nature to be found within a reasonable amount of time.

There are some presentations available through the homepages of the toolkits, for example [42], but they have not proved to be very useful for the task at hand.

One interesting source of information on the IPv6 Toolkit can be found in form of a video of a talk held in 2013. In this video, Fernando Gont, the toolkit's main author, gives some insight into the tools and their intended use, as well as into the toolkit in general and his plans for the future. [34]

At the beginning of the presentation, Gont mentions one of his ideas behind the tools which helps to explain their apparent lack of out of the box functionality. He states that the tools are not intended to be used in a "run and crash" manner, but rather serve as an aid in "I have this packet I want to send and see what happens" situations. [34] This piece of information proved to be of importance with respect to selecting tests, since it basically meant that all tests using tools from this toolkit would have to be "built" during the development process of the test suite.

One consequence of this is that a very high workload would be generated in the search of possible exploits and tests, before any work could progress. In addition, every test case would have to be fine tuned individually, which can also be a very time consuming task. Finally since there would be no peer review for these tests, the possibility of errors made during test case creation would be higher.

Due to these observations, the decision was made to start building the test suite using THC-IPv6 and, depending on the number and kind of tests possible with it, if necessary fill in the information gaps by designing appropriate tests using IPv6 Toolkit.

4.4.1 Manual testing

Since the focus at this stage is on the tools a very simple testing network was set up. The diagram of which can be seen in Figure 4.1

Figure 4.1: The setup of the network for manual testing.

In this setting "Eric" is the attacking host, while "Heisenbug" is an out-of-the-box installation of Fedora 20. Both hosts are virtualized using the techniques described in Chapter 5.

After installation, a simple ping6 confirmed connectivity between hosts, but there was some trouble with the IPv6 Toolkit. See Section 4.6 for details.

The final configuration of the attacking host was:

- *OS:* Ubuntu 12.04.5
- *Kernel:* 3.2.0-67
- *Network Interfaces:*
  - *eth0:* IPv4/IPv6 interface bridged to physical network with Internet connectivity. 1Gb wire speed.
  - *eth1:* IPv6 interface connected to the host undergoing test. Configured using Stateless Address Autoconfiguration. 1 Gb wire speed.
- *Number of CPU's:* 1
- *RAM:* 2048 MB

The tested node was supplied with the following configuration:

- *OS:* Fedora 20
- *Kernel:* 3.14.9-200
- *Network Interfaces:*
  - *eth0:* IPv6 interface connected to the host performing the test. Configured using Stateless Address Autoconfiguration. 1 Gb wire speed.
- *Number of CPU's:* 1

- *RAM:* 768 MB

After the first cycle of running all programs included in the THC-IPv6 tool set seventeen were in the loop for closer inspection. Seven of these were selected for the automation process. The chosen test and details on their concepts and workings will be provided in the following section. In total, the selected tests and exploits provide over 70 data points per tested host, ranging from vulnerabilities over well known exploits to progress of implementation.

This was deemed sufficient and IPv6 Toolkit was at this point excluded from the automated testing program.

However, to be able to provide an overview of the programs and features included in IPv6 Toolkit, some manual tests were performed.

4.4.2 Automation

For the process of automation, two options were considered.

The first was pyUnit, the python relative of JUnit, originally intended as unit testing framework, which is a set of tools and commands used to repeatedly perform small to medium sized tests on a piece of software. [45] It was taken into consideration, since from the perspective of the task at hand it seemed to have all the abilities essential to get the job done. Tests can be defined once and run on different hosts without much overhead. While this is true, the programmatic realisation seemed rather unwieldy and offered little flexibility. Therefore, pyUnit was quickly abandoned in search for a better alternative.

This alternative was found in the second framework, fabric. Fabric is a suite that provides secure shell based interaction with the command lines of remote hosts, as well as the ability to run local commands. Its main purpose is the automation of maintenance and administration tasks of software and remote computers. It is easy to define tasks which fabric will then perform on a set of data, most often a list of hosts with their respective network addresses. The power to execute commands remotely gives great flexibility and the local command execution guarantees fast integration of command line tools like THC-IPv6. For these reasons, fabric was chosen as the automation tool for this project. [14]

After the tests were chosen, implementation began. Initial difficulties with output formats and the seperation of functionality into functions, were soon overcome and work progressed at a reasonable pace. The output of all used tools is parsed and aggregated into a simple JSON object, which contains the name of the test, whether it succeeded or not, and where appropriate the original output.

At the end of the final development iteration, the testing suite was able to perform all selected tests on an arbitrary number of hosts in a semi-automatic fashion. If the test suite detects that a host is down after an attack, that is it does not respond to Echo Requests, the tests are halted and the user is asked to check on the host under test.

The decision to remain at the semi-automatic stage, and therefore exclude features like programatic control of virtual machines and in-tool evaluation of graphic user interface respon-

sivenes on tested hosts was made due to time constraints. These features may, however, be added in future work.

4.5 Details of chosen Exploits

This section presents the chosen tests and exploits and explains, what they test and how they work.

4.5.1 denial6

This tool tests two vectors known to cause denial of service in IPv6 systems.

Large hop-by-hop header with router-alert and filled with unknown options poses a threat as described in Section 3.2.1.3 by using a router alert option. The large hop-by-hop header and the unknown options could act as possible evasion mechanism or could trigger vulnerabilities in badly written IPv6 stacks.

Large destination header filled with unknown options is, according to the author of the tool set, a test case taken from his own experience with network stacks and their possible vulnerabilities.

4.5.2 exploit6

Tests a total of four exploits against targets, two of which have entries in the Critical Vulnerabilities and Exposures database. The other two have been programmed by the author of the tool set in order to test for erroneous implementations of packet processing.

Very Large Ping, with 6 checksum combinations: In this test, the author aims at exposing vulnerabilities in the parsing of very large Echo Requests. A failure to parse or a resulting denial of service could indicate an exploitable vulnerability in the implementation. In addition, the test checks for behaviour associated with comparison and calculation of checksums.

Large Ping, with 3 checksum combinations: This test aims at the same weaknesses as the previous one, but sends slightly smaller packets. It seems reasonable to assume that some nodes would simply drop Echo Requests exceeding a certain size. It can be assumed, that this test extends the coverage of the above test to such nodes.

CVE-2003-0429: Bad Prefix Length: This vulnerability is also described on Security Focus under ID 7880. The underlying vulnerability is a buffer overflow in a network analysis tool commonly used by network administrators. The tool was named ethereal when the vulnerability was discovered and is today known as wireshark. The overflow is triggered by parsing malicious prefix length fields of IPv4 or IPv6 packets. Since information on the problem is sparse in all sources, the author of this tool notes that the implementation is unsure. [116]

CVE-2004-0257: Send too big message followed by TCP SYN: Is a vulnerability which was originally present in OpenBSD and NetBSD. It causes denial of service by sending a packet reducing the Maximum Transmission Unit to a very small value and then sending a TCP SYN packet. This causes an unhandled exception in the kernel of the affected system leading to a complete system failure. [117]

4.5.3 fragmentation6

This tool is designed to test multiple dimensions in one go. It starts by running tests which send fragmented packets designed to test if they pass a firewall. Then it continues by running tests designed to determine the state of implemented features and finally it runs fragment based denial of service attacks against the target node. In total, thirty-three tests are performed. Eight of which target firewalls, ten aim at implementation state and fifteen try to disable the target.

To evaluate results of firewall and implementation tests, the tester would have to run a traffic capture device at appropriate locations in the network and review the generated output. The denial of service part is assessed by the tool using the Echo Request/Reply mechanism.

4.5.4 fuzz_ip6

fuzz_ip6 is a fuzzer able to generate packets with random data inserted at different locations within the packet. Additional to fuzzing values of parts of the packet, this tool can also fuzz headers and their content. It can do so with or without adding a transport layer to the packets it sends.

For this thesis, we used this tool to its full extent. This comprises fuzzing the following packets and headers:

- ICMPv6 Echo Request
  - one-shot fragmentation
  - IP header
  - jumbo packet header
  - source routing header
  - destination header
  - hop-by-hop header
  - router alert header
- ICMPv6 Neighbor Solicitation
  - hop-by-hop header
  - router alert header
- ICMPv6 Neighbor Advertisement
  - hop-by-hop header

  - router alert header
- ICMPv6 Router Advertisement
  - hop-by-hop header
  - router alert header
- Multicast Listener Report
  - hop-by-hop header
  - router alert header
- Multicast Listener Done
  - hop-by-hop header
  - router alert header
- Multicast Listener Query
  - hop-by-hop header
  - router alert header
- Multicast Listener v2 Report
  - hop-by-hop header
  - router alert header
- Multicast Listener v2 Query
  - hop-by-hop header
  - router alert header
- Node Query
  - hop-by-hop header
  - router alert header

Running all these tests generates approximately 50.000 packets. Before and after each sort of packet the host under test is probed with a regular Echo Request. Should no reply be received, the test is marked as failed.

4.5.5 implementation6

This tool runs 55 tests to determine the extent of implemented features with concern to various options, header orders and fragmentation. Its does so by simply sending an Echo Request preceded by the dimension under test. If no Echo Reply is received, the tested node fails the test.

The tool tests the following dimensions:

Hop-by-hop header implementation is tested for standard adherence by first sending an empty header, then a header containing enough zeros to be fragmented and then by chaining multiple headers of this type, all of them without payload.

Destination option header implementation is tested in the same way as hop-by-hop header implementation.

Fragmentation is tested by:

- sending a correctly fragmented packet
- sending a correct one-shot fragmentation packet, that is a normal packet sent as "fragmented" packet by adding the fragmentation header and setting the "last fragment" field to true
- sending multiple packets with different overlapping fragments
- sending a fragmented message with missing fragments

Source-routing is tested via routing header type 0, which is deprecated and should be ignored, as well as via unauthorized mobile source routing.

Chaining the tool also chains multiple headers together to evaluate the host under test. Chained tests include the chaining of:

- mobile source routing - source routing
- fragmentation - source routing
- hop-by-hop - fragmentation - source routing
- destination - fragmentation - source routing

and permutations thereof.

Solicitation is tested for mobile prefixes, node information and certificates, as well as for neighbors, which are tested forward and inverse with multiple hop-limit values.

Authentication Header and Encrypted Security Payload implementations are tested for correct handling of zero headers with Internet Control Message Protocol and Transmission Control Protocol as upper layer payload.

jumbo option on packets with length zero and a length too small to be classified as jumbo.

multicast on local link behaviour is tested with multiple packets.

length and checksum field mismatches conclude the test set.

4.5.6 sendpees6

This tool generates a huge amount of secure neighbor solicitation messages. This results in load on the target since it has to verify all incoming cryptographically generated addresses and RSA signatures.

4.5.7 sendpeesmp6

This is essentially an extended and multi process version of sendpees6. This tool and its non-multi process version were selected to test the ability of nodes to perform under load. On hosts that do not implement a rate limiting schema for their Secure Neighbor Discovery, one or both of these tools should result in a Denial Of Service.

4.5.8 smurf6

This is a tool which implements a local smurf attack, utilizing the local all-nodes multicast address for amplification. It was chosen in order to test the resilience of nodes to this type of denial of service, as well as to determine which hosts will respond to the packets sent to initiate this kind of attack.

4.6 Encountered Problems

On initial setup, the attacking host was running Ubuntu 14.04. IPv6 network connectivity was confirmed via ping6 and THC-IPv6 worked just fine. Upon starting the manual testing process with IPv6 Toolkit two errors were encountered:

- Error while performing Neighbor Discovery for the Destination Address
- Error while learning Source Address and Next Hop

Various posts on blogs and forums suggested that these errors originate in a change of how newer versions of the Linux kernel treat IPv6 traffic. Testing the toolkit on the host machine and a physical network yielded expected behaviour.

To try and solve this problem, a rollback to Ubuntu 12.04.5 with kernel version 3.11 was performed. On receiving the same error messages, the Kernel was rolled back to version 3.2. The hosting system and the attacking node were now running the same operating systems. After tweaking multiple parameters there was still no change in behaviour.

Upon disabling the internet-facing interface on the attacking host, the errors ceased and reactivating it did not convert the system back to an erroneous state. Rebooting the system did. Checking the output of some diagnostic tools available through the operating system revealed the problem, which seems situated in the kernels routing table.

Whenever eth0 was the first entry for link local addresses in the routing table the errors were thrown. The problem was fixed by writing a simple script modifying the routing table.

A detailed bug description will be sent to the creators of IPv6 Toolkit shortly after publication of this work, should the problem still persist at that time.

CHAPTER 5

Laboratory Setup and Data Acquisition

In this chapter, the structure of the network used for testing will be outlined and the rationale behind choices made during its setup will be given. The selection process for hosts will be documented and the hosts under tests will be presented. Afterwards, the methods for data acquisition not previously described will be highlighted. The chapter will close with a discussion of problems encountered during the process of host selection and network setup.

5.1 Setup of Testing Network

In order to test and evaluate the current state of affairs in IPv6 security, a controlled environment had to be created. To this end a virtualized network was constructed. Additionally, for testing Android devices, a small physical setup had to be created.

The machine hosting the test network runs Ubuntu 12.04.5 LTS as its operating system and provides 11512 MB of RAM combined with an AMD FX^{TM}-6100 Six-Core Processor.

Oracle Virtual Box version 4.1.12 provided virtualisation and the guest extensions which were installed on all guest systems for which they were available ensured that the memory footprint of the guests was within operational parameters of the host.

In addition to the automation toolset described in the previous chapter, some common network monitoring tools and techniques were used to ensure correct results. These included, among others, wireshark, tcpdump and iftop.

To ensure that the test results are as untainted by IPv4 side effects as possible, a guests only network was created in Virtual Box and every machine was configured with 2 network interface cards. One connected to the internal IPv6 only network, one bridged to a physical IPv4 and IPv6

network with Internet connectivity. On all operating systems that provided the option, IPv4 was completely disabled on the network interface connected to the internal network.

Prior to testing, the package and update management systems on all machines were run to ensure that their respective operating systems are at the current state of stable development. To ensure reproducibility of results, a snapshot of the virtual machines was taken after the updates were performed.

During testing, the network interface card connected to the internet-connected network was removed from the settings of the guest machines.

Figure 5.1 provides a network schematic. Hosts are named according to their operating system with the exception of "Eric" which in our scenarios is the attacking host, running Ubuntu 12.04.5 LTS.

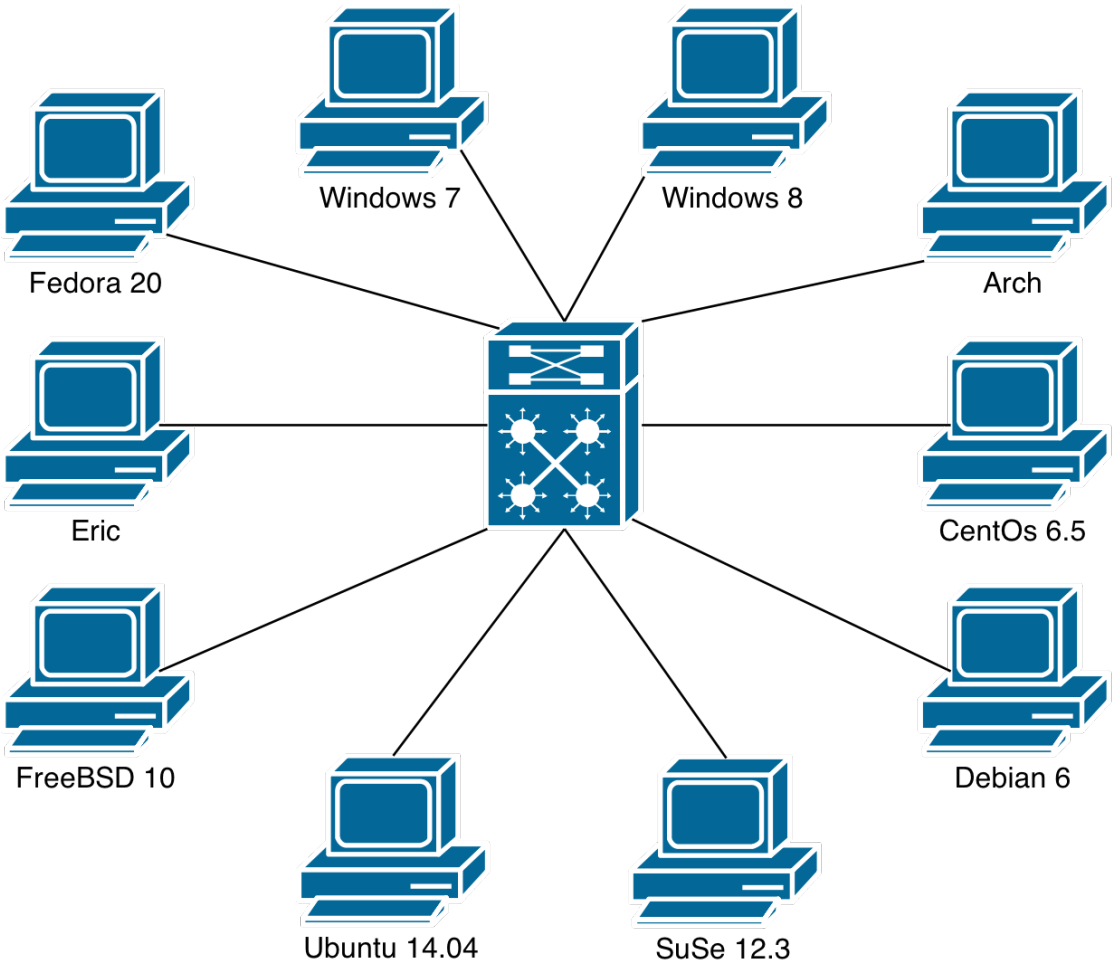

Figure 5.1: Setup of the virtual testing network.

Due to problems discussed in Section 5.4, in addition to this virtual network a second physical network had to be created to test Android devices. The layout of this network is visualized

in Figure 5.2.

“Eric” is once again the attacking host, in this case an HP EliteBook 840G running Ubuntu 14.04 LTS. “Jane” is a host running surveillance tasks during the test run, because network interface card limitations on “Eric” caused certain tests to not yield correct results during fully automated test runs.

A technicolor TG788A1vn running firmware version 10.2.5.5 was used as a bridge connecting the devices. Although this device does not advertise any special IPv6 related capabilities in the documentation available to us, all additional features present were disabled for the duration of the test.

The hardware platforms running Android version 4.1.2 and 4.3.1/4.4 were a GooPhone I5s and a Samsung Galaxy S 3, respectively. The GooPhone ran the vendor supplied firmware, while the SGS 3 was set up using CyanogenMod 10.2.0 and CyanogenMod 11.

CyanogenMod is a popular open source firmware distribution available for a multitude of devices. [1]

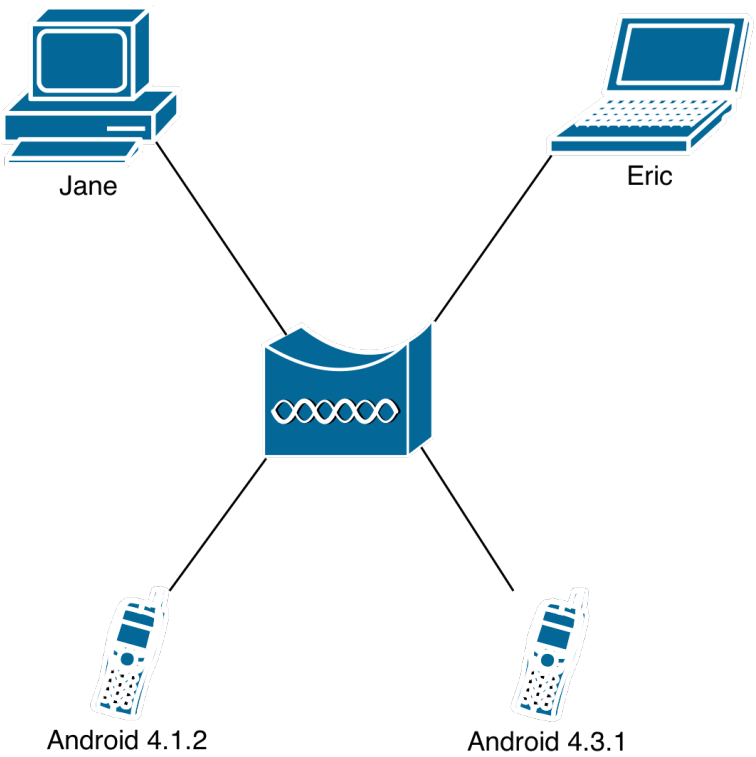

Figure 5.2: Setup of the physical testing network.

5.2 Choice Of Hosts

In total, a number of twelve hosts were tested. Seven flavours of *nix, two versions of Windows and three versions of Android.

The *nix host under test were running:

- Debian 6.0.10
- Ubuntu 14.04
- Arch
- OpenSuse 12.3
- CentOs 6.5
- FreeBSD 10
- Fedora 20

all of which were downloaded from their respective download pages.

The used *nix distributions were chosen for a number of reasons.

Debian was selected due to its large base of high profile users and because it is one of the Linux distributions with lots of “children”. This means that a lot of distributions are based on Debian. Additionally it is a rather old distribution. [20] [126] [119]

Ubuntu is an example of *nix “generations”. It is based on Debian and still benefits from the work of Debian packages and their respective maintainers. This is well documented. For example, in the process of developing new versions of Ubuntu there is a date which signifies the “DebianImportFreeze”. Before this date new versions of packages are automatically imported into the repositories of the new version of Ubuntu. It was chosen because it grants the option of comparing it to its “parent” but mainly for its large popularity. [13] [38]

Arch was chosen as an independent distribution. It features no classic release cycle as most other distributions but works under a rolling release model. [2] This means, that new developments are continually pushed to users, with no need to upgrade between major versions. In a rolling release software, the user is always up-to-date on the latest changes, but might suffer from instabilities due to previously unknown or unfixed problems.

OpenSuse is a derivative/child/successor of SuSe which in turn forked of Slackware, a Linux distribution originally released in 1993. While SuSe today aims at commercial and corporate customers, OpenSuse focusses on bottom-up community driven collaborative development. While OpenSuse offers a rolling release update schema the default is a classic release cycle. [5] [6]

CentOS is a derivative of Red Hat Enterprise Linux. It aims at providing functional compatibility with Red Hat Enterprise Linux, while keeping its packages free of vendor branding and associated artwork. [3]

FreeBSD is special in this list, because it is the only Unix derivative. It was derived from BSD, the Unix version developed at the University of California, Berkeley. The project maintains its own list of security vulnerabilities found in FreeBSD which, considering the lifespan of the project, is surprisingly short. [17] [4]

Fedora is a child of Red Head, not to be confused with Red Hat Enterprise Linux. The project presents itself as value-driven and promotes being "First". That means they commit to a rapid release cycle and try to provide the most "advanced" experience in all aspects of their system. [40] [15]

Between them these versions of *nix systems cover a large base of philosophies and attitudes within the spectrum of software creators that comprises the *nix community. This should result in a relevant overview of general state of readiness within the *nix community.

On the side of Microsoft operating systems, Windows 7 and Windows 8 were scrutinized. Both were downloaded via the Microsoft Developer Network, accessible via the account issued by the Technische Universität Wien.

The version choice for Windows-based operating systems was rather complicated, due to the fact that little to no information on sales numbers, number of registered installs, or similar data useful for determining "market shares" is published by Microsoft.

Data provided by third parties such as, for example, NetApplications, which is a company providing web analytic services, is unreliable due to two major reasons. First, they do not report the sources for their data, which means that neither the mode of collection can be judged nor can the impact of possible biases be weighed. Additionally, the most likely methods of data acquisition, for example the User-Agent string of a web request, can easily be modified by the client.

More reliable methods like OS-Fingerprinting, a means of identification which uses the behaviour of the targets operating system in response to corner case scenarios, would be feasible through the use of current scanning software, like ZMap. However, it is outside the scope of this work, and has its own problems and limitations. [127]

Taking an overview of multiple third party information providers, Windows 7 and Windows 8 seem to be on top of the market. [39] [10] [38]

Windows XP still seems to have a large market share but was left out of consideration for this thesis, since it has reached its official end of life/end of support. [120]

Android was tested using versions 4.1.2, 4.3.1 and 4.4. The first of which was in the form of vendor supplied firmware, while the latter two were retrieved via the download page of Cyanogenmod.

Versions 4.1.x to 4.3.x of Android have been designated codename "Jelly Bean" by Google and represent a total of 54.2% of all active Android devices. [8] Android 4.1.2 was selected because it is within the version range of 4.1.x, which at 26.5% market share clocks in as the largest group not only within our subgroup but in total.

Android 4.3.1 is included due to the fact that it is the underlying system in Cyanogenmod 10.2. That is the "latest stable release" of Cyanogen before they switched to a rolling release model [12] and was downloaded more than 775000 times according to ClockworkROM man-

ager, which is the top hit for ROM managers for Android based devices according to the Google Play store.

Android 4.4 codenamed “KitKat” is the current version of the Android operating system, and is run by 20.9% of active Android devices. [8]

5.3 Data Acquisition

The acquisition of test results was handled in part by the automation tool written for this lab.

This was done in two ways:

- Exploit implementations providing their own feedback on success, for example via exit codes, were evaluated according to the response they gave.
- For all other exploits, an Echo Request was sent to the host after the attack or where appropriate after a “cool down” period, a time frame that allows the exploit to compromise the functionality of the host. Failure to respond results in the evaluation of the exploit as successful.

In addition to this, some test runs were recorded using wireshark and the resulting network dump was evaluated manually. The goal was to gain a deeper insight into the working of the exploits and to verify the automated results on the basis of random samples.

While completely recording all test runs would have been possible, it was not done, since the data gained would not have provided additional insights. Should the need arise to gather this data, all that is necessary is the restoration of the virtual machines to the appropriate snapshot and running the automation tool against the host while recording.

Some exploits were verified manually in a live fashion. This includes all exploits aiming to degrade system performance, since in most operating systems “higher” levels of functionality, for example the graphic user interface, will degrade faster than network communications, when the system is put under heavy load. Test cases that resulted in the automated system reporting a successful exploitation were also verified manually. In case of verification, the extent of the success was recorded and the system restored to a functional state by means of a reboot.

5.4 Encountered Problems

5.4.1 Data Acquisition

Initially, the process of data acquisition was supposed to be fully automated. This, however, proved to be unfeasible in the given time frame and with the tools at hand. Especially performance degradation proved hard to measure. While parameters like CPU-load and memory usage can give clues as to how many resources the system is currently dedicating to a single or number of processes, it is still hard to predict at which point load actually impedes performance in a way noticeable by the user. Another point is that measuring network interface throughput

and thereby evaluating the user's ability to access the Internet without impediment could have an adverse effect on other parts of the tests, due to the additional stress placed on the network interface device.

These and other similar considerations led to the conclusion that a human aided approach for result collection is the most useful one in this scenario.

5.4.2 Android

The initial concept for this laboratory was to build a virtual testing network which includes all hosts under test and one attacking node. To this end, integration of emulated Android devices was planned.

In the process of setting up the lab, this was abandoned. While connecting the emulator to the virtual network proved difficult, but possible, getting uninitiated transmissions past the emulator's internal mechanisms proved to be unachievable in the given time frame.

Connecting through network redirection, as described in the Android Emulator guide provided on the Android Developer Platform, was unsuccessful. The connection to the virtual network was achieved by running the emulator inside a Virtual Machine and connecting that machine to the virtual network. [125]

Initiating all communication from the device under test was considered, but the idea was discarded in favour of the more time-efficient approach of setting up a physical testing network.

5.4.3 Virtualisation

While building the laboratory, the decision was made to virtualize as many machines as possible. This introduces the potential for false positives due to the virtualisation technology itself.

However, since today many systems, especially servers, are virtualized, this bias was judged to be acceptable, if the tests were conducted in more than one virtualization environment. To this end, all described tests were repeated using KVM with an identical setup. The machines were exported from VirtualBox, imported into KVM and the network setup was duplicated. The results were identical for both technologies with the exception of slight differences in timing and delays.

In case of successful exploitation, other currently running systems were checked to determine whether the test case had impacted them as well. This was used as an indicator for the overall status of the virtualized environment.

CHAPTER 6

Results

In this chapter, information on the processing of raw data will be given. To this end, the steps to canonize results, integrate observations and measurements and generate interpretable representations are presented.

The outcome of this process will then be analyzed and discussed in the context of overall security of a system. Additionally, assumptions will be made on the reasons of found problems. Their likelihood will be the topic of further discussion.

6.1 Result Processing

This section will given an overview of techniques and tools used to manipulate the raw results and extract meaningful representations for visualisation and further analysis.

Results of the laboratory were divided into three partitions. One is the well parsable output of the automated testing tool, the second is the free text observation notebook created during the process of testing to record behaviour hard to test or encode in the automated tool. The third is the dump of network traffic acquired during testing.

6.1.1 Automated Testing Tool

The automated testing tool generates text files containing the results of a run in JSON format. In order to ensure that further processing and manual review can be done in a smooth fashion, it was confirmed that the results are sorted.

For initial viewing of data and possible further statistical computations, the JSON files were parsed using the code in Listing 6.1.

The resulting lists were then added to a *Pandas Dataframe*. Pandas is a BSD-licensed open source data analysis library for the python language. It provides data structures and data analysis tools. Pandas is part of the SciPy ecosystem, which contains multiple packages for various forms of and tasks in scientific computing. [115] [44]

```
def get_results(path):
    d = json.load(open(path))
    val = []
    for item in sorted(d.iteritems()):
        val.append(item[1]['passed'])
    return val
```

Listing 6.1: Extracting results for Pandas

After establishing the dataframe, the results were filtered for tests that failed in some nodes, but succeeded in others. This was accomplished using the code found in Listing 6.2. The results that diverge were considered particularly interesting because they might differentiate progress or philosophy of implementation and could provide a good starting point for in-depth discussion of the overall state of readiness of IPv6.

```
def diverging(x):
    if all(x) or not any(x):
        pass
    else:
        return x
```

Listing 6.2: Finding differences

6.1.2 Laboratory Notebook

The notebook was cleared of unnecessary comments, reread and then incorporated into the result section of this work. This resulted in inclusion of information which would otherwise have been lost due to technical limitations and time constraints.

6.1.3 Network Dump

The network dump was manually analyzed using wireshark. Through this manual analysis an insight into details of exploits and their interaction with the node under attack was gained. Additionally, the dump was used to confirm the results of the automated test tools by means of random sample verification.

6.2 Results

In this section, the actual results of the testing will be presented and discussed. The found problems and vulnerabilities can roughly be divided into three subcategories: denial of service, problems found during fuzzing and issues identified while testing for behaviour with concern to implementation details.

6.2.1 Denial of Service

Denial of Service Attacks aim at rendering a service, for example a website or router, unavailable to its intended users. Two general types can be discerned, distributed and non-distributed. While in distributed denial of service attacks multiple nodes attack the service, in the non-distributed variant only one host is considered the attacking node. Processes that induce a small load on the attacker, for example generation of packets the target has to cryptographically verify, and large loads on the victim, for example the verification of said packets, can be considered prime vectors for such attacks.

Windows 7 performed rather poorly in all tests related to this subclass of attacks. While the network layer stayed functional during the testing and did not show noticeably larger latency when responding to Echo Requests, the graphical user interface was rendered unresponsive by most of the attacks. It did not recover within a three minute time frame after the end of a successful attack. Trying to call the Task Manager or otherwise interact with the machine via its terminal elicited no response. The machine had to undergo a hard reset in order to regain functionality.

During the time Windows 7 was down, all other virtual machines did not show any signs of degradation. Additionally, sending packets between the other machines worked fine. Both these points strongly suggest that the problem is indeed within the operating system or the network driver.

As all other virtual machines used the same network driver and the behaviour was not visible on all tested hosts, it is likely that the network driver is not the problem.

A test on real hardware, performed about 6 months after the original laboratory tests, did show an increase in latency times, but the graphical user interface was not rendered unresponsive. Whether this is due to improvements introduced in the time between the tests or other factors, could not be determined.

Android hosts showed some interesting behaviour. All versions under test did not react to denial of service using Secure Neighbor Discovery, but they did show unresponsiveness when Router Alert or Destination Header based attacks were used. Depending on the header and version of Android, latency from the beginning of the attack until the stop of communication on the network layer ranged from instantaneous to around 2 seconds. Two observations could be made. The network layer recovered almost immediately after the end of the attack and the graphic user interface was not affected in the slightest way on any of the devices.

6.2.1.1 Router Alert or Destination Header

This test generates pressure on the destination node as well as the path by running two scenarios.

First by sending packets with the hop-by-hop option followed by a Router Alert header all nodes on the path acting as routers will be forced to parse the upper layer content. To make this relevant, multiple random options are added to the packet, before finishing off with an Echo Request. For the second test, an Echo Request with a Destination Header is sent, in order to

force the receiving node to parse a long chain of random options attached to this header.

These tests showed no effect on any of the operating systems with the exception of Android.

In all tested versions of the Android system, the first attack resulted in a loss of network communication. While version 4.3.1 and 4.4 lost connectivity almost immediately, version 4.2.1 remains able to communicate for about two seconds. The difference in response times to the attack could possibly be explained by differences in hardware, but no definitive tests were conducted with regard to that hypotheses.

The second attack also resulted in a denial of service on all Android versions. Again Android 4.2.1 lost connectivity about one to two seconds later than the other versions.

Assuming that Android implements some sort of rate limiting, their approach seems to fail when dealing with these kinds of packets. If, however, there is no rate limiting employed the reasons for failure of the network stack are obvious. As mentioned, the user interface did not show any response to the higher loads.

6.2.1.2 Secure Neighbor Discovery

In this test, the node tested was put under pressure by sending a large amount of Secure Neighbor Discovery packets.

Specifically, Neighbor Solicitation messages were used, which results in the potential of the target performing a huge number of expensive cryptographic operations to verify the Cryptographically Generated Address and possibly the RSA signature. [78] [77]

When looking at performance of the user interface, two host systems stood out in terms of their respective slowdowns and recovery times. Arch Linux starts behaving sluggish after about thirty seconds, but remains responsive and recovers in just under one minute after stopping the attack. OpenSuSe also suffers from increased response times, which seem about as long as those of Arch, but it takes around ninety seconds to recover after the attacks have stopped.

The command used for testing response times was a directory listing of / using the *ls -a* command. Execution took place directly in the terminal and time measurements were performed using a timer on the system hosting the virtual machines.

The decision to use rough intervals for time measurement was based on a number of reasons. First, since the systems under test are virtualized, the execution times do not only depend on the scheduler of the system itself, but additionally on the underlying host. Second, although it would have been possible to measure "exact" response times for commands, operating under the assumption that the impact of virtualisation is negligible, comparing them across all operating systems seemed nonsensical without base lines for execution since absolute comparisons would not isolate the impact of higher system load. And finally, the perceived quality of service of an operating system, that is the way the user perceives the performance, seemed the more important measure. To gain valid results on this last point, a different design including at least one group of subjects would have been necessary. Although interesting, this is definitely out of the scope of this thesis.

6.2.2 Fuzzing

As discussed in Section 3.1.3, fuzzing is a technique for finding possible vulnerabilities within a piece of software by connecting a generator yielding (semi-)randomly generated values to the programs input and monitoring it for faults.

6.2.2.1 Multicast Listener Report

During testing, a problem arose when requesting a Multicast Listener Report version 1 from a FreeBSD machine with fuzzed parameters. The following conditions could be observed:

- The FreeBSD host does not respond to ICMPv6 Echo Requests
- Other hosts do not respond to ICMPv6 Echo Requests
- All terminal interfaces are still responsive

After this failure the following procedures were observed, and their results documented:

Routing tables and interface information on hosts were inspected and showed no anomalies.
FreeBSD was rebooted. This reboot did not effect the overall network. Communication was still not possible via IPv6.
An additional host which had not been running at the time of the test was booted. No behavioural difference in the network could be observed and no communication between hosts could be established.
All hosts were then shut down and rebooted. This procedure re-established communication between hosts. Testing on FreeBSD was continued starting with the first test after the failure case.

It seems reasonable to assume that the network stack of FreeBSD interacted with the networking mechanisms of Virtual Box in an unforeseen way to produce this situation. Further investigation of the issue, although very interesting, was deemed out of scope. In this context it might be interesting to note that a method to test the IPv6 capabilities of the virtualisation mechanism in isolation would be a great tool to further diagnose this problem.

6.2.2.2 Router Advertisements

When fuzzing Neighbor Discovery Protocol Router Advertisement messages, Windows 7 and Ubuntu 14.04 lost network connectivity and did not regain it after the test.
In addition, when testing the host running Ubuntu using only this test case, it generated side effects on other hosts online in the network. After the test, other hosts had multiple addresses configured on the interface connected to the test network.
Initially this was attributed to fuzzing of the prefix field, but further testing makes this explanation seem less plausible. After the first occurrence of this effect all hosts, except Ubuntu, were rebooted and the test case was run against a different machine. Examination of all hosts showed that no additional addresses had been configured. This procedure was repeated three times, in

order to check if the problem might originate in the randomness of fuzzing. Each consecutive run yielded the same result, with no additional addresses.

Restarting Ubuntu and performing the test again led to a regression to the original problem, where additional addresses were found on the other machines.

6.2.2.3 Android

The results of the automated fuzzing test on Android should be viewed with caution. From version 4.3.1 onwards the failed tests most likely failed due to large variations in the response times to Echo Requests.

For example, in version 4.3.1 during the fuzz test of Router Advertisements, response times of over 42000 milliseconds with occasional lost packets could be observed on the external monitoring station.

To evaluate the impact of these results the test cases were re-run. While the system was under load, user interface and network performance were monitored manually. During this test no noticeable degradation in service quality of the network layer could be observed. The graphical interface interaction was identical to normal conditions and browsing the Web also showed no perceivable slowdown.

It is conceivable that the networking stack just could not keep up with the total load and decided to prioritize packets based on some internal metric. This would be in accordance with the suggestions of the Internet Engineering Task Force concerning rate limiting as described in Chapter 3. This hypothesis is also backed by the observation that packets requiring more processing also resulted in longer delays and only very heavy loads resulted in packet loss. The latter could indicate a second level of rate limiting. While the first level postpones the reply and stores the packet for later processing, the second level regulates the maximum storage space allocated for the queueing of packets.

6.2.3 Implementation

Tests in this category mostly work using the same mechanics. An Echo Request with different headers and options is sent to a node. If the node replies with an Echo Reply, it is assumed that the headers have been parsed correctly, otherwise the test fails. The used toolkit performs over 50 tests. Those deemed most relevant in a security context are presented in detail, while the others are summarized.

6.2.3.1 Node Information Query

This additional type of Internet Control Message Protocol message is the exception in this set of tests as it does not follow the Echo Request/Reply mechanism described in the beginning of this section.

Node Information messages are designed to enable asking a certain IPv6 node for information like its hostname or its fully qualified domain name. This can help gaining valuable information in a “serverless” environment. [85]

Of all tested operating systems only FreeBSD supports this mechanism.
The reasons for this could be manyfold. Other operating system and network stack designers could think that the additional risk of information disclosure is not worth taking, especially considering the environments their systems will mostly work in. Additionally, a new denial of service vector is introduced as this mechanism could be used to amplify traffic.

Given that the denial of service vector can be mitigated by load balancing, information disclosure can be limited with the correct use of scopes and replay attacks are defended against by using a nonce.

Another likely reason is that there is no demand for this service large enough to warrant its integration.

6.2.3.2 Echo Request From Local Multicast Address

In this test, case an Echo Request was sent to the nodes and its origin was set to the local all-nodes multicast address.

Replying to such a packet can pose a potential threat, since it has the potential to generate huge loads on a network. Every vulnerable node will send one reply per request and every node on the network will have to parse all replies. A packet multiplication factor proportional to the number of vulnerable hosts is achieved. While this might not be the most attractive vector in terms of multiplication potential, if used in combination with others, it can become relevant.

Only two operating systems, CentOS and Debian, replied to this message under their default settings.

6.2.3.3 Fragmented and Source-Routed Echo Request From Local Multicast Address

In this similar scenario, some very interesting changes occur. Only Android versions 4.3.1 and 4.4 respond to this packet.

Considering that these two operating systems did not respond to the packets of the previous test, it indicates some form of separate processing of fragmented packets that does not follow the same procedures as the processing of unfragmented packets. This results in potentially problematic behaviour. This is especially interesting as it might open up vulnerabilities that only exist if the parsed packet is fragmented.

The fact that CentOs and Debian drop this packet is possibly because it includes a source routing option and an Echo Request.

Summarizing the rest of the implementation tests, it can be concluded that there still is work to do for some corner cases of the protocol on almost all varieties of IPv6 stacks. How relevant these corner cases are in day-to-day operations could be the topic of future work in the field of IPv6 research.

From a security perspective, the failures are handled gracefully, opening no evident gaps.

6.3 Summary

The main question posed in the context of this thesis is if IPv6 is fit to deploy with respect to security. Or in other words, what would happen, from a security point of view, if the community decided that from today onwards, IPv4 would be disabled on all machines.
To answer these questions, multiple points were considered throughout this thesis.

Concerning attack vectors, it can be summarized that while some new angles appeared, others were lost due to the changes in technologies and concepts. Overall, the gains and losses on this front seem to cancel out.

The overall behaviour of the hosts tested in the laboratory part of the thesis met expectations. Some problems remain, mostly in the sector of Router Advertisements, but most systems handled really well, even under load and extreme conditions.

Denial of service attacks still pose a problem especially with mobile hosts, like phones, that lack the computing power of modern day computers. It was surprising to see that these devices handled denial of service attacks gracefully in most situations and only lost network connectivity, a fact that depending on circumstances might even go unnoticed.

Moving into the concrete limitations of this thesis and its laboratory we see that routers and firewall have been excluded from the setup. This was done for several reasons.

When building the laboratory, one key question was whether or not to include firewalls. Literature research and an overview of recent conferences showed that there is already some momentum in this part of the field as for example evidenced by [147] [124] and [33]. This together with the cost of acquiring the required equipment led to the decision to include more hosts and to forego the testing of firewalls.

Looking at routers the situation is similar, although the tests and publications are mostly produced by vendors in the form of books and white papers, for example [141] and [30]. There is also some independent research in the field. [128]

Building a more complex testing setup and including more types of nodes and services would provide a good starting point for future work.

With Windows hosts the built-in firewall had to be disabled in order to enable Echo Request/Reply pairs. Working around this by creating an on-host evaluation mechanism could further improve the output of the presented tests.

Additionally, the test cases were not checked for correctness. This means that in some cases results could be false positives. For the purpose of this thesis, this does not bias the results since all tested hosts were submitted to the same tests and comparisons. However, checking and extending the test cases could also present an opportunity for further improvement of this work and its results.

CHAPTER 7

Conclusion

In conclusion, it can be said that there are still hurdles to overcome, but that will always be the case. Every new technology has its problems and some of them are hard if not impossible to control. Overall, the IPv6 protocol performed well in the tests. When combining the results of this thesis with the tests done by peers, IPv6 does not appear to be less secure than IPv4. The attack vectors might have shifted, but a well configured node does not show a significant attack surface. Local denial of service attacks are still a problem, just as they have been in IPv4, but as far as can be judged today not to a larger extent than in IPv4.

The protocol evolves, holes are fixed and the standards are adapted to reality and best practices, as evidenced by the evolution from Neighbor Discovery Protocol to Secure Neighbor Discovery Protocol.

In the opinion of the author, IPv6 is ready to be deployed on a large scale. Yes, there will be incidents and some unforeseen problems will arise, but there is only so much you can do in a sandbox. It is time to take IPv6 out of the lab and into the real world. IPv6 will become the next big step in networking and the sooner we start adapting, the better off we will be in the long run.

The extension of the automated testing framework to include currently not covered parts of IPv6, for example Mobile IPv6, provides large areas for future research. Additionally, the problems found in Section 6.2.2.2, could provide an interesting starting point for further research as no light could be shed on the root cause of this problem.

Bibliography

[1] About - CyanogenMod. http://wiki.cyanogenmod.org/w/About. Accessed: 2014-08-16.

[2] About Arch Linux. https://wiki.archlinux.org/index.php/Arch_Linux. Accessed: 2014-10-06.

[3] About CentOs. http://www.centos.org/about/. Accessed: 2014-10-06.

[4] About FreeBSD. http://www.freebsd.org/about.html. Accessed: 2014-10-06.

[5] About OpenSuse. http://en.opensuse.org/openSUSE:Why_openSUSE. Accessed: 2014-10-06.

[6] About SuSe. https://www.suse.com/company/. Accessed: 2014-10-06.

[7] Amsterdam Internet Exchange Statistics. https://ams-ix.net/technical/statistics. Accessed: 2014-11-22.

[8] Android Developers - Android Version Distribution. https://developer.android.com/about/dashboards/index.html?utm_source=ausdroid.net. Accessed: 2014-08-18.

[9] Brief History Of The Internet. http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet. Accessed: 2014-11-21.

[10] Computer operating systems: global market share 2012-2014. http://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems-since-2009/. Accessed: 2014-09-29.

[11] CVE - Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/. Accessed: 2014-08-11.

[12] CyanogenMod 11.0 M6 Release. http://www.cyanogenmod.org/blog/cyanogenmod-11-0-m6-release. Accessed: 2014-09-29.

[13] DebianImportFreeze. https://wiki.ubuntu.com/DebianImportFreeze. Accessed: 2014-10-03.

[14] Fabric Homepage. http://www.fabfile.org/. Accessed: 2014-08-16.

[15] Foundations. https://fedoraproject.org/wiki/Foundations. Accessed: 2014-10-07.

[16] Free Pool of IPv4 Addresses Depleted. https://www.nro.net/news/ipv4-free-pool-depleted. Accessed: 2014-11-24.

[17] FreeBSD Security Advisories. https://www.freebsd.org/security/advisories.html. Accessed: 2014-10-06.

[18] GitHub fgont/ipv6toolkit. https://github.com/fgont/ipv6toolkit/releases. Accessed: 2014-08-14.

[19] GNUCitizen - dnsmap. http://www.gnucitizen.org/blog/new-version-of-dnsmap-out/. Accessed: 2014-08-10.

[20] GNU/Linux Distribution Timeline. http://futurist.se/gldt/wp-content/uploads/12.10/gldt1210.svg. Accessed: 2014-10-03.

[21] Gont's web site. http://www.gont.com.ar/. Accessed: 2014-11-20.

[22] How much IPv6 is there? http://chrisgrundemann.com/index.php/2009/how-much-ipv6-is-there/. Accessed: 2014-11-21.

[23] IANA - Internet Protocol Version 6 Address Space. http://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml. Accessed: 2014-11-21.

[24] IANA Home. http://www.iana.org/numbers. Accessed: 2014-11-24.

[25] ICMP and Security in IPv6. http://blogs.cisco.com/security/icmp-and-security-in-ipv6/. Accessed: 2014-11-22.

[26] IEEE 802.11 Wireless LANs. http://standards.ieee.org/about/get/802/802.11.html. Accessed: 2014-11-24.

[27] IEEE 802.1D Spanning Tree. http://standards.ieee.org/about/get/802/802.1.html. Accessed: 2014-11-24.

[28] IEEE 802.3 Ethernet. http://standards.ieee.org/about/get/802/802.3.html. Accessed: 2014-11-24.

[29] IPv6 Enabled Networks. http://v6asns.ripe.net/v/6?s=_ALL. Accessed: 2014-11-22.

[30] IPv6 First-Hop Security Concerns. http://www.cisco.com/web/about/security/intelligence/ipv6_first_hop.html. Accessed: 2015-01-09.

[31] IPv6 Multicast Address Space Registry. http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xhtml. Accessed: 2014-11-23.

[32] IPv6 Routing Header Security. http://www.secdev.org/conf/IPv6_RH_security-csw07.pdf. Accessed: 2014-12-18.

[33] IPv6 Security and Firewalls - si6networks. http://www.si6networks.com/presentations/ipv6kongress/mhfg-ipv6-kongress-ipv6-security-assessment.pdf. Accessed: 2015-01-09.

[34] IPv6 Toolkit News (IPv6 Hackers Meeting 1 - July 2013). https://www.youtube.com/watch?v=YYEPuZVrHFc. Accessed: 2014-08-09.

[35] Mailing List - IPv6 Hackers. http://www.ipv6hackers.org/mailing-list. Accessed: 2014-11-20.

[36] Measurements|World IPv6 Launch. http://www.worldipv6launch.org/measurements/. Accessed: 2014-11-22.

[37] Network Reconnaissance in IPv6 Networks. http://tools.ietf.org/html/draft-ietf-opsec-ipv6-host-scanning-04. Draft to replace RFC 5157.

[38] Operating system market share. http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0. Accessed: 2014-09-29.

[39] OS Statistics. http://www.w3schools.com/browsers/browsers_os.asp. Accessed: 2014-09-29.

[40] Overview Fedora Project. https://fedoraproject.org/wiki/Overview. Accessed: 2014-10-07.

[41] PKCS #1: RSA Encryption Standard. ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-1.asc. Accessed: 2014.12.31.

[42] Presentation: Attacking the IPv6 Protocol Suite. https://www.thc.org/papers/vh_thc-ipv6_attack.pdf. Acccessed: 2014-08-16.

[43] Press Release ICANN IPv4 Depletion. https://www.icann.org/en/system/files/press-materials/release-03feb11-en.pdf. Accessed: 2014-11-24.

[44] Python Data Analysis Library. http://pandas.pydata.org/. Accessed: 2014-10-15.

[45] PyUnit - the standard unit testing framework for Python. http://pyunit.sourceforge.net/. Accessed: 2014-08-16.

[46] Request for Comments (RFC) Pages. http://www.ietf.org/rfc.html. Accessed: 2014-08-10.

[47] RFC 1122 - Requirements for Internet Hosts – Communication Layers.

[48] RFC 1546 - Host Anycasting Service.

[49] RFC 1726 - Technical Criteria for Choosing IP The Next Generation (IPng).

[50] RFC 1858 - Security Considerations for IP Fragment Filtering.

[51] RFC 1883 - Internet Protocol, Version 6 (IPv6) Specification. Obsoleted by 2460.

[52] RFC 1930 - Guidelines for creation, selection, and registration of an Autonomous System (AS).

[53] RFC 1981 - Path MTU Discovery for IP version 6.

[54] RFC 1992 - The Nimrod Routing Architecture.

[55] RFC 2131 - Dynamic Host Configuration Protocol.

[56] RFC 2205 - Resource ReSerVation Protocol (RSVP).

[57] RFC 2402 -IP Authentication Header.

[58] RFC 2406 - IP Encapsulating Security Payload (ESP).

[59] RFC 2460 - Internet Protocol, Version 6 (IPv6) Specification.

[60] RFC 2526 - Reserved IPv6 Subnet Anycast Addresses.

[61] RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1.

[62] RFC 2644 - Changing the Default for Directed Broadcasts in Routers.

[63] RFC 2675 - IPv6 Jumbograms.

[64] RFC 2711 - IPv6 Router Alert Option.

[65] RFC 2827 - Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.

[66] RFC 2828 - Internet Security Glossary.

[67] RFC 2923 - TCP Problems with Path MTU Discovery.

[68] RFC 3302 - Traditional IP Network Address Translator (Traditional NAT).

[69] RFC 3306 - Unicast-Prefix-based IPv6 Multicast Addresses.

[70] RFC 3315 - Dynamic Host Configuration Protocol for IPv6 (DHCPv6).

[71] RFC 3513 - Internet Protocol Version 6 (IPv6) Addressing Architecture.

[72] RFC 3756 - IPv6 Neighbor Discovery (ND) Trust Models and Threats.

[73] RFC 3779 - X.509 Extensions for IP Addresses and AS Identifiers.

[74] RFC 3791 - SEcure Neighbor Discovery (SEND).

[75] RFC 3879 - Deprecating Site Local Addresses.

[76] RFC 3956 - Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address.

[77] RFC 3971 - SEcure Neighbor Discovery (SEND).

[78] RFC 3972 - Cryptographically Generated Addresses (CGA).

[79] RFC 4007 - IPv6 Scoped Address Architecture.

[80] RFC 4081 - Security Threats for Next Steps in Signaling (NSIS).

[81] RFC 4193 - Unique Local IPv6 Unicast Addresses.

[82] RFC 4291 - IP Version 6 Addressing Architecture.

[83] RFC 4294 -IPv6 Node Requirements.

[84] RFC 4301 -Security Architecture for the Internet Protocol.

[85] RFC 4620 - IPv6 Node Information Queries.

[86] RFC 4861 - Neighbor Discovery for IP version 6 (IPv6).

[87] RFC 4862 - IPv6 Stateless Address Autoconfiguration.

[88] RFC 4941 - Privacy Extensions for Stateless Address Autoconfiguration in IPv6.

[89] RFC 5095 - Deprecation of Type 0 Routing Headers in IPv6.

[90] RFC 5280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.

[91] RFC 5340 - OSPF for IPv6.

[92] RFC 5453 - Reserved IPv6 Interface Identifiers.

[93] RFC 5902 - IAB Thoughts on IPv6 Network Address Translation.

[94] RFC 6071 -IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap.

[95] RFC 6105 - IPv6 Router Advertisement Guard. Updated by 7113.

[96] RFC 6177 - IPv6 Address Assignment to End Sites.

[97] RFC 6275 - Mobility Support in IPv6.

[98] RFC 6398 - IP Router Alert Considerations and Usage.

[99] RFC 6434 - IPv6 Node Requirements.

[100] RFC 6554 - An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL).

[101] RFC 6583 - Operational Neighbor Discovery Problems.

[102] RFC 6618 - Mobile IPv6 Security Framework Using Transport Layer Security for Communication between the Mobile Node and Home Agent.

[103] RFC 7136 - Significance of IPv6 Interface Identifiers.

[104] RFC 7217 - A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC).

[105] RFC 7321 -Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH).

[106] RFC 7346 - IPv6 Multicast Address Scopes.

[107] RFC 7371 - Updates to the IPv6 Multicast Addressing Architecture.

[108] RFC 768 - User Datagram Protocol.

[109] RFC 791 - Internet Protocol.

[110] RFC 792 - INTERNET CONTROL MESSAGE PROTOCOL.

[111] RFC 793 - TRANSMISSION CONTROL PROTOCOL.

[112] RFC 826 - An Ethernet Address Resolution Protocol.

[113] RFC-Editor Webpage. `www.rfc-editor.org`. Accessed: 2014-11-11.

[114] Scapy. `http://www.secdev.org/projects/scapy/`. Accessed: 2014-08-14.

[115] SciPy.org. `http://www.scipy.org/`. Accessed: 2014-10-15.

[116] Security Focus - BugID 7880. `http://www.securityfocus.com/bid/7880`. Accessed: 2014-08-21.

[117] Security Focus - BugID 9577. http://www.securityfocus.com/bid/9577. Accessed: 2014-08-21.

[118] SI6 IPv6Toolkit. http://www.si6networks.com/tools/ipv6toolkit/index.html. Accessed: 2014-08-14.

[119] Software distributions based on Debian. https://www.debian.org/misc/children-distros. Accessed: 2014-10-03.

[120] Support for Windows XP has ended. http://www.microsoft.com/en-us/windows/enterprise/end-of-support.aspx. Accessed: 2014-09-29.

[121] Teardrop Denial Of Service Attack. http://www3.physnet.uni-hamburg.de/physnet/security/vulnerability/teardrop.html. Accessed: 2014-12-15.

[122] The CORBA Specification. http://www.omg.org/spec/CORBA/. Accessed: 2014-11-24.

[123] The Hackers Choice THC-IPV6 Homepage. https://www.thc.org/thc-ipv6/. Accessed: 2014-08-09.

[124] TROOPERS14 - Testing IPv6 Firewalls with ft6. https://www.troopers.de/wp-content/uploads/2013/11/TROOPERS14-Testing_IPv6_Firewalls_with_ft6-Oliver_Eggert.pdf. Accessed: 2015-01-09.

[125] Using the emulator. http://developer.android.com/tools/devices/emulator.html#emulatornetworking. Accessed: 2014-10-07.

[126] Whos's using Debian. https://www.debian.org/users/. Accessed: 2014-10-03.

[127] ZMap - The Internet Scanner. https://zmap.io/. Accessed: 2014-09-29.

[128] J. G. Bejar. Ipv6 security analysis. Technical report, School of Information Studies - Syracuse University, 2014.

[129] S. Bellovin. Problem areas for the ip security protocols. In *USENIX Security*, 1996.

[130] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, 1986.

[131] J. V. Carreira, D. Costa, and J. G. Silva. Fault injection spot-checks computer system dependability. *Spectrum, IEEE*, 36(8):50–55, 1999.

[132] A.R. Choudhary and A. Sekelsky. Securing ipv6 network infrastructure: A new security model. In *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, pages 500–506. IEEE, 2010.

[133] J. P. Degabriele and K. G. Paterson. Attacking the ipsec standards in encryption-only configurations. In *IEEE Symposium on Security and Privacy*, pages 335–349, 2007.

[134] M. Drmota, B. Gittenberger, G. Karigl, and A. Panholzer. *Mathematik für Informatik*. Heldermann, 2007.

[135] P. J. Gmytrasiewicz and E. H. Durfee. Decisiontheoretic recursive modeling and the coordinated attack problem. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems| AIPS92*, pages 88–95, 1992.

[136] F. Gont. *IPv6 Toolkit - Linux Manual Pages*, 2014. Provided by: `http://www.si6networks.com/tools/ipv6toolkit/ipv6toolkit-v1.5.3.tar.gz`.

[137] M. Sutton; P. Amini; A. Greene. *Fuzzing: Brute Force Vulnerability Discovery*. Addison Wesley Professional, 2007.

[138] S. Kumar. Smurf-based distributed denial of service (ddos) attack amplification in internet. In *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on*, pages 25–25. IEEE, 2007.

[139] A. B. Gonzalez M. Moya. *thc-ipv6(8) - Linux Manual Page*, 2014. Retrieved from: `http://www.thc.org/releases/thc-ipv6-2.5.tar.gz`.

[140] A Moravejosharieh, H Modares, and R Salleh. Overview of mobile ipv6 security. In *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, pages 584–587. IEEE, 2012.

[141] E. Vyncke S. Hogg. *IPv6 Security*. Cisco Press, 2009.

[142] A. Takanen. *Fuzzing for software security testing and quality assurance*. Artech House, Boston, 2008.

[143] A. S. Tanenbaum. *Computer Networks, 4-th Edition*. Prentice Hall, 2003.

[144] H Tschofenig and D Kroeselberg. Mobile IPv6 Security Framework Using Transport Layer Security for Communication between the Mobile Node and Home Agent. 2012.

[145] R.P. Van Heerden, I.M. Bester, and I.D. Burke. A review of ipv6 security concerns. 2012.

[146] J. Voas. Fault injection for the masses. *Computer*, 30(12):129–130, 1997.

[147] J. Weber. IPv6 Security Test Laboratory. Master's thesis, Ruhr-University Bochum, Germany, 2013. `http://blog.webernetz.net/2013/05/06/ipv6-security-master-thesis/`.