OS fingerprinting with IPv6

GIAC (GCIA) Gold Certification

Author: Christoph Eckstein, christoph.eckstein@samapartners.com
Advisor: Antonios Atlasis

Accepted: September 21st, 2011

Abstract

Over the next years to come IPv6 will eventually replace IPv4 in private and public networks. This paper attempts to describe upcoming challenges and limitations as well as new methods of OS fingerprinting with the shift to IPv6. This includes performing network scans and finding live hosts to fingerprint. The focus of this paper will be on describing the changes made within the protocol headers in IPv6 and the consequences for OS fingerprinting. This covers known fingerprinting method with new IPv6 header fields and new methods of OS fingerprinting enabled with the IPv6 protocol. Protocol examinations also include IPv6 extension headers introduced with IPv6. Current fingerprinting and scanning tools will be listed and checked for IPv6 fingerprinting support. Finally, possible countermeasures to prevent OS fingerprinting on IPv6 protocol level will be mentioned.

1. Introduction

In real life human fingerprints are used as a method of identification. As of today no two fingerprints were found to be alike, hence fingerprints are an excellent way to positively identify a person beyond reasonable doubt. Just like a human fingerprint has its unique characteristics, an operating system has its unique implementation of communication protocols by which it can be identified. In this context, OS (Operating System) fingerprinting is the analysis of certain characteristics and behaviors in network communications in order to remotely identify an OS and its version without having direct access to the system itself (Allen, 2007). Like in real life, fingerprints are compared to a database of known identities. Captured system communications characteristics and behaviors need to be compared to a database of known operating systems. OS fingerprinting is a useful and important tool for both security professionals (called 'white-hats') and crackers (called 'black-hats'). Not only can it be used to get a good overview of systems on a network, but it can reveal vulnerable systems that need to be secured or might be promising targets to attack.

The main reason why OS fingerprinting is feasible refers back to the RFCs responsible for communications protocols like IP (IPv6), ICMP, TCP and UDP only describing the expected normal behavior. They define how the protocols should be used and should work, but they do not explain in detail how certain parameters should be chosen or how to handle unexpected or incorrect flag combinations. For instance, the default values for the TTL (Time to live) in the IPv4 or the ISN (Initial Sequence Number) in the TCP protocol are not defined. This lack in specification leaves room for interpretation by OS vendors when implementing these protocols in their operating systems. Therefore, different operating systems tend to show different characteristics in communication and handling of unexpected or incorrect flag combinations.

OS fingerprinting is an important technique for both white-hats and black-hats. Not only is OS fingerprinting an excellent technique for black-hats to discover vulnerable systems, but it is also very valuable to security professionals to be able to discover, manage and control network resources and identify vulnerable systems.

For black-hats it is not only essential to find live hosts on a targeted network, but to identify their operating systems. This way the black-hat can use the appropriate exploits and intrusion techniques for the identified operating systems. For instance, if a targeted webserver is known to run a Linux operating system it would not be promising to try exploits and attacks crafted for an IIS webserver. Furthermore, an OS fingerprint may reveal that a targeted system is missing specific security patches or service packs which fix known vulnerabilities. With that knowledge black-hat might be able to target those specific vulnerabilities and gain control over the system with little effort (Allen, 2007).

Security professionals may utilize OS fingerprinting to manage and secure networks and systems. White-hats or administrators can use OS fingerprinting to map out and identify all systems existent in their managed network segment. The information gained may be used to identify unpatched or vulnerable systems or unauthorized and rouge devices connected to the network. In addition, other network devices such as printers and switches have to be identified, managed and updated as well (Nerakis, 2006).

OS fingerprinting is a complex and extensive subject. There are different methods of OS fingerprinting, both technical and non-technical. These include port scanning, banner grabbing, active stack fingerprinting, passive stack fingerprinting and social engineering. Moreover, stack fingerprinting analysis can be made on different layers of the TCP/IP reference model and utilizing various protocols. Although other protocols like TCP or UDP are mentioned in this paper and are used more extensively for OS fingerprinting, this paper focuses on OS fingerprinting based on the IPv6 protocol (Nerakis, 2006).

This paper looks at some IPv6 fingerprinting methods derived from IPv4 and some newly enabled ones. Additionally, some of the methods are tested to demonstrate the possibilities of OS fingerprinting with IPv6. But the limits of this paper do not allow for analyzing and testing of all IPv6 OS fingerprinting methods.

2. The Change from IPv4 to IPv6

2.1. Protocol Header changes in IPv6

This chapter gives a short overview of the IPv6 protocol. The introduction of the IPv6 protocol is not only in response to the exhaustion of IPv4 address space, but an evolution of the IPv4 protocol in terms of improving existing features and adding new ones. Improvements in the IPv6 protocol include (Network Working Group, 1995):

- Expanded address space.
- Extended routing (more levels of addressing hierarchy, simple auto-configuration of addresses).
- Improved scalability of multicast routing.
- Simplified header (lesser header fields compared to IPv4 to lower processing costs, dropped header fields are now available as optional extension headers).
- Support for optional extension headers (allows for faster processing because extension headers are not examined by routers, allows for arbitrary length of IPv6 header).
- Support for authentication and privacy through encryption.
- Support for source routes (Source Demand Routing Protocol (SDRP)).
- Quality of service capabilities.

These improvements also allow for foreseen and unforeseen successes in simplicity, resiliency and flexibility of the IPv6 protocol (Murphy & Malone, 2005).

In order to better understand the effect the change from IPv4 to IPv6 has on OS fingerprinting, the following figures describe the differences and similarities of these protocols.

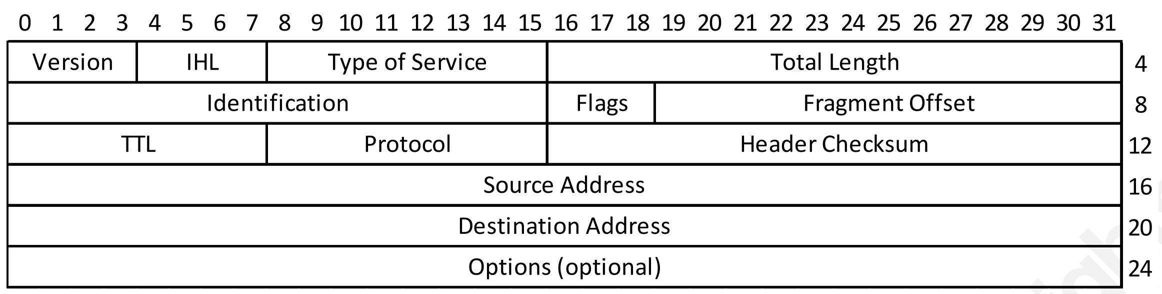


Figure 1 - IPv4 Protocol Header (SANS Institute, 2011a)

Figure 1 - IPv4 Protocol Header shows the IPv4 protocol header as specified in the RFC 791 (University of Southern California, 1981). The Figure 2 - IPv6 Protocol Header shows the new IPv6 protocol header (Network Working Group, 1998). Although the 'Source Address' and 'Destination Address" header fields are four times as long in IPv6 compared to IPv4, the entire protocol header without options is only twice as long. This is because some of the header fields from IPv4 were discarded in IPv6 and somewhere moved to an extension header. That speeds up the routing process as routers between the source and destination don not need to process extension headers.

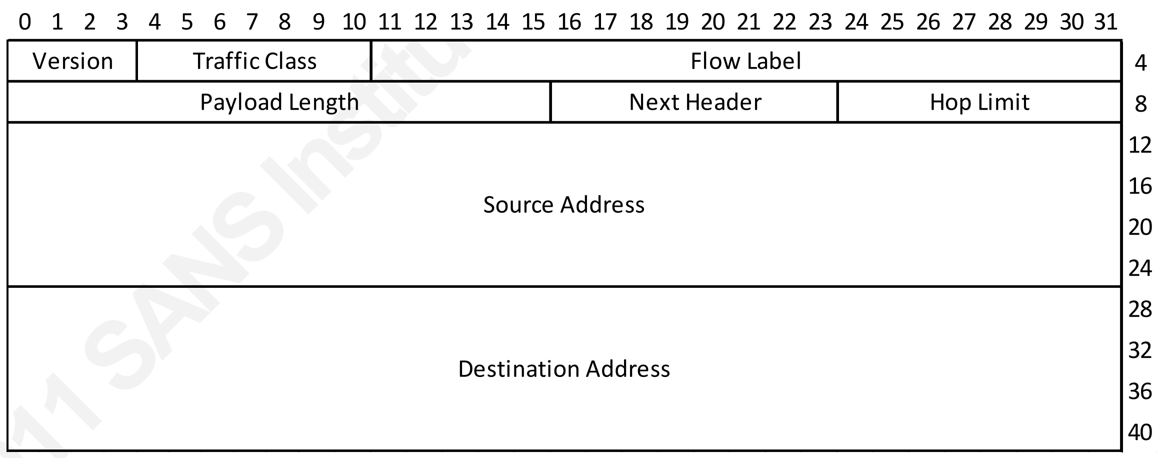


Figure 2 - IPv6 Protocol Header (SANS Institute, 2011b)

Next, this paper is going to examine some specific header fields in the IPv6 protocol. There are only three header fields that have the same name as before; these are 'Version', 'Source Address' and 'Destination Address'. The source and destination address fields gained in length. The version header field is the only field that is identical to the IPv4 protocol. The version field specifies the IP protocol version and therefore caries the information how to process the packet. Table 1 – Modified header fields in IPv6 give an overview over the IPv6 and corresponding IPv4 header fields as well as new features introduced in IPv6.

IPv6	**IPv4**	**Description**
Traffic Class / Flow Label	Type of Service	This field may be used by a host to label those packets for which it is requesting special handling by routers within a network, such as non-default quality of service or "real-time" service.
Payload Length	IHL (Internet Header Length) / Total Length	Length of the remainder of the packet following the IPv6 header, in octets.
Next Header	Protocol	Identifies the type of header immediately following the IPv6 header.
Hop Limit	TTL	Decremented by 1 by each node that forwards the packet. The packet is discarded if Hop Limit is decremented to zero.

Table 1 – Modified header fields in IPv6

As only the absolute necessary header fields are placed in the default header of the IPv6 protocol, all other additional functionality is moved to the optional extension headers. Table 2 - Extension headers in IPv6 shows all available extension headers (Hermann-Seton, 2002).

IPv6	**Description**
Hop-by-Hop Options Header	Options that need to be examined by all devices on the path.
Destination Options Header	Options that need to be examined only by the destination of the packet.
Routing Header	Methods to specify the route for a datagram (used with Mobile IPv6). The Routing header is used by an IPv6 source to list one or more intermediate nodes (or topological clusters) to be "visited" on the way to a packet's destination.
Fragment Header	The Fragment header is used by an IPv6 source to send payloads larger than would fit in the path MTU to their destinations. The Fragment Header replaces and moves the fragmentation capabilities of IPv4 to the optional extensions headers. Therefore the Fragment Header is not examined by routers and fragmentation must be handled by the end points.
Authentication Header (AH)	The Authentication header is used to provide authentication and integrity assurance for IPv6 packets.
Privacy Header (ESP)	The Privacy Header or Encapsulating Security Payload (ESP) seeks to provide confidentiality and integrity by encrypting data to be protected and placing the encrypted data in the data portion of the Privacy Header.

Table 2 - Extension headers in IPv6

In addition to the header fields that were changed or added in IPv6, the header field 'Checksum' was discarded completely. This speeds up the routing process as no checksum needs to be checked and calculated by every router the packet passes. The integrity of packets is normally checked by upper layer protocols like TCP, so this does not negatively affect the reliability of a communication.

2.2. Challenges in finding live hosts with IPv6

This paper assumes that the targeted live host for fingerprinting is known. But in reality a new challenge with IPv6 is that it is not as easy to find live hosts to fingerprint as with IPv4. Scanning a subnet with IPv4 might be done in a matter of hours, but it might take days or weeks with IPv6 as the address space is much wider. Furthermore, assuming that the number of live hosts will not raise the same amount as the address space increases with the transition to IPv6, it will be much more difficult to find live hosts. This is mainly because most addresses are not used yet, although there are techniques to reduce the IPv6 search space (Chown, 2005). For instance, a typical IPv4 subnet has 8 bits, a typical IPv6 subnet 64 bits reserved for host addressing. At a scanning speed of one probe per second the IPv4 subnet scan would take less than 5 minutes to complete, whereas the IPv6 subnet would take more than 5 billion years to complete (6net, 2008).

3. Existing OS fingerprinting methods from IPv4

3.1. IPID generation and fragmentation

The new fragmentation extension header in IPv6 reuses known IPv4 header fields and therefore does not offer new functionality. The IPv4 header fields 'Identification' (packet ID), 'Flags' and 'Fragment Offset' were moved to the IPv6 fragmentation extension header. While the basic functionality of fragmentation stays the same in IPv6, the relocation of the fragmentation header fields enables new ways of usage for OS fingerprinting. This is because fragmentation is now handled by the source and destination rather than intermediate routers to speed up packet delivery times (Network Working Group, 1998). In IPv4 fingerprinting, tools avoid fragmentation because it is done by intermediate routers which results in identifying a router rather than the sending operating system (Lyon, 2011a). In IPv6, fragmentation is now exclusively done by the source and destinations hosts. This means the fragmentation extension header is not processed and manipulated by intermediate routers and therefore can be used to identify the source OS. In IPv6 routing devices or routers along the travel path of a packet do not need to support fragmentation anymore and will simply discard any packets too large for the MTU of the next network segment (Nerakis, 2006).

Although fragmentation fingerprinting is generally avoided in IPv4, one flag is used for fingerprinting (Lyon, 2011b). That is the "DF" or "don't fragment" flag of the IPv4 header. This is only set by the sending host and not manipulated by intermediate routers, hence it can be used for fingerprinting. But the DF flag is the only IPv4 header field that was removed in the IPv6 fragmentation header as it is not necessary for packet delivery any more. This means the DF flag is not available for fingerprinting in IPv6. But as fragmentation is handled by source and destinations hosts in IPv6, all other fragmentation extension header field could now be used for fingerprinting (Trowbridge, 2003).

The IPv6 fragmentation header is specified in RFC 2460 and adds three modifications to handle error conditions that may arise when reassembling fragmented packets. First, the time to complete the reassembly for a packet is limited to 60 seconds within the reception of the first fragment. If not all fragments are received within this timeframe, all fragments must be discarded. Furthermore, if the first fragment has been received, an ICMP Fragment Reassembly Time Exceeded response message should be sent back. Secondly, the fragment offset in the fragmentation extension header is specified in 8-octet units, relative to the start of the fragment of the original packet. This means that, if a fragment is not the last fragment and the length of the fragment is not a multiple of 8 octets, the fragment must be discarded. In such an event an ICMP Parameter Problem, Code 0 error message should be sent in response. Finally, if the length and offset of a fragment are such that the total length of the reassembled packet would exceed 65,535 octets, all fragments should be discarded (DITCHE, 2005).

Extensive tests should be performed to examine if the OS vendors comply with the aforementioned recommendations. If any of them do not, this could be a good indication that might be used for OS fingerprinting.

3.1.1. IPID generation

The IPID is a unique identifier for every packet in a connection. It is used as reference for every single fragment of a packet when reassembling it. Many operating systems utilize a system wide counter for IPID generation. Other, more advanced operating systems randomize the IPID. Of these two variations the system wide counter may produce predictable IPIDs which could be used to identify an operating system (Trowbridge, 2003).

Some fingerprinting tools like Nmap even try to find patterns in the IPID generation of an OS (Nerakis, 2006).

3.1.2. Overlapping fragments

The handling of overlapping fragments could be used for OS fingerprinting in IPv4, although avoided because intermediate routers handle fragmentation (Lyon, 2011b). Overlapping fragments means that two consecutive fragments carry the same bytes of the final IP packet. For instance, the last two bytes of the first fragment represent the same two bytes as the first two bytes of the second fragment. In this case the OS TCP/IP stack has to decide how to reassemble the final packet. Depending on the implementation of the OS some might tend to override the bytes from the earlier fragment with the bytes from the later fragment and vice versa. Such differences can lead to OS fingerprinting (Nerakis, 2006).

As far as the IPv6 fragmentation handling is concerned, this was initially specified in the RFC 2460. This RFC allows overlapping fragments, but it does not specify how to process them (DITCHE, 2005). This shortcoming is addressed in the RFC 5722, which intends to update the IPv6 fragmentation handling. Specifically, RFC 5722 explicitly requires overlapping fragments to be silently discarded, eliminating any effects of such type of attacks (Krishnan, 2009). However, extensive tests should be performed to examine if the OS vendors comply with RFC 5722, or if they still implement RFC 2460 regarding fragmentation.

3.2. TTL or Hop Limit

The Time-to-Live or short TTL header field in the IPv4 protocol is used as a counter to prevent packets from bouncing around the internet or between routers indefinitely. The TTL is set to a specific value by the sending host and decremented by 1 by every router along the travel path. As soon as the TTL reaches 0, the packet is dropped, regardless of whether the target host was reached or not. The RFC 791, which specifies the IPv4 protocol, does not require an explicit value that should be used as initial value for the TTL (University of Southern California, 1981). As a result, different operating systems use different initial TTL values. These deviations between operating systems can be used to fingerprint them (Nerakis, 2006).

In the IPv6 protocol the TTL header field is renamed to hop limit, but the basic functionality stays the same and therefore can still be utilized to fingerprint operating systems. However IPv6 introduces the stateless address autoconfiguration (Thomson & Narten, 1998). Address autoconfiguration allows a host to obtain parameters like IPv6 prefix, link local MTU and hop limit on network initialization when a router is present. The router sends router advertisements to hosts on the local network in order to set certain parameters. By this, all operating systems regardless of differences in their TCP/IP stack implementation can obtain the same initial hop limit. Therefore, OS fingerprinting based on the initial hop limit may not be feasible in networks using address autoconfiguration (6deploy, 2011).

Still, in some cases like a home local area network where no router is present, differences in the initial hop limit as seen in IPv4 are expected between operating systems.

3.3. ICMP port unreachable message

The ICMP protocol in IPv4 allows OS fingerprinting based on the payload of port unreachable messages (Postel, 1981). In order to be able to associate the port unreachable message with the original message that triggered it, the first part of the originating message starting with the IPv4 header is included in the payload of the response. As the RFC 1122 only specifies the minimum required bytes of the original packet to include in the ICMP port unreachable response, different operating systems tend to include different amount over this minimum (Braden, 1989). Therefore, ICMP port unreachable messages are a good source of OS fingerprinting in IPv4 (Nerakis, 2006).

The basic principle of port unreachable messages stays the same in ICMPv6. The ICMP protocol is moved to an extension header of the IPv6 protocol and some ICMP codes are changed, but the header fields stay the same. Operating systems include different amounts of data from the original packet in ICMPv4 error messages, although the RFC specifies otherwise. This leads to the assumption that ICMPv6 implementation may do the same and provide OS specific fingerprints (Nerakis, 2006).

4. OS fingerprinting methods enabled by IPv6

4.1. IPv6 Extension Headers

The new concept of optional extensions headers replaces the IPv4 options. These extension headers might enable new methods of fingerprinting hosts. In this section we will look at some of these extension headers and identify header fields that might possibly be manipulated to stimulate unexpected responses. The extent of this paper does not allow testing all these possibilities. Until then all described possibilities do not represent actual proven fingerprinting methods. Nevertheless, this section will provide an overview of what might be feasible with the IPv6 extension headers.

There are a small number of available extension headers at the moment, each of which is identified by a distinct next header value. The currently implemented extension headers are (Ahmed & Asadullah, 2009):

- Hop-by-Hop options header
- Routing header
- Fragment header
- Destination options header
- Authentication header
- Privacy header

The extensions headers are only processed by the endpoints of a connection with exception of the Hop-by-Hop extension header. The Hop-by-Hop options header is examined by every node or router along the packets travel path.

In the following, some extensions headers are listed and possible fingerprinting methods enabled by them. This is just a short summary and does not explain the header fields or packet manipulation possibilities in detail.

4.1.1. Destination Options Header

The destination options header is specified in RFC 2460. The destination options header consists of the header fields “next header” (8-bit), “header extension length” (8-bit) and “options” (variable length). The options header field contains one or more Type-Length-Value (TLV) options. One TLV consists of the fields “option type” (8-bit), “options data length” (8-bit) and “options data” (variable length) (Network Working Group, 1998). The Destination Options Header may be used with an unrecognized destination type to fingerprint a targeted OS. To allow this the destination option type should be set to an unrecognized value for a destination header. By analyzing the response sent by targeted host, a distinction between different operating systems might be feasible (Ahmed & Asadullah, 2009).

4.1.2. Routing Header

The routing header is specified in RFC 2460. The routing header consists of the header fields “next header” (8-bit), “header extension length” (8-bit), “routing type” (8-bit), “segments left” (8-bit) and “type-specific data” (variable length) (Network Working Group, 1998). As the RFC does not specify how to handle unexpected or malformed routing headers, we can try the following manipulations to the routing header in order to stimulate a response that might be used to identify a specific operating system (Ahmed & Asadullah, 2009).

- Unrecognized routing type

  For this test the routing type field should be set to an unrecognized value, the segments left field should be set to “1” and the data field set to the “::0” address. The RFC 2460 specifies that in that case the processing target host has to make a decision based on the value of the segments left field. If the segments left field is set to “0” the host should ignore the routing header and proceed to the next header. Otherwise an ICMPv6 error message should be generated (Nerakis, 2006).

- Unrouted address

  The routing type field should be set to the default type, the segments left field should be set to “1” and the data field set to the “::0” address. The RFC 2460 specifies that the first address to be visited by the packet is the one in the IPv6 header. That address or host then has to send the packet to the next address specified in the routing extension header. As this address is considered unrouted, we might want to examine the responses by different operating systems (Nerakis, 2006).

- Incorrect extension header length

  The routing type field should be set to the default type “0”, the segments left field should be set to “2” and the data field set to the “::0” address. The RFC 2460 specifies that the receiver should discard the packet, if the segments left field is greater than the routing addresses in the routing extension header. A corresponding ICMPv6 error message should be generated (Nerakis, 2006).

4.1.3. Fragment Header

The fragmentation header is the replacement for the fragmentation options in the IPv4 header. See chapter 3.1 for fragmentation fingerprinting methods.

The RFC 2460 recommends an order for the extension headers to appear. The order recommended is:

- IPv6 header
- Hop-by-Hop options header
- Destination options header
- Routing header

- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header
- Upper-layer header

Furthermore, the RFC 2460 specifies that all extension headers, except for the destination options header, should only occur ones. The destination options header might occur twice, once before the routing header and again once before the upper-layer header. Both the order recommendation and occurrence specification of the extension headers might offer ways to manipulate packets to stimulate unexpected responses. As the order is merely a recommendation, different operating system might process various order combinations differently (DITCHE, 2005).

4.2. MTU discovery

As the fragmentation of packets is done by endpoints in IPv6 rather than by intermediate routers, the sending host first has to discover the MTU (maximum transmission unit) for the path to the destination host. IPv6 offers the path MTU discovery protocol specified in RFC 1981. The question arises how different operating systems implement or use the MTU discovery protocol. Different operating systems might choose a different default MTU for the first connection. Some might choose a low transmission unit to avoid fragmentation and the need for MTU discovery. Others might try to discover the MTU on every connection. All these possibilities might be used to fingerprint operating systems as vendors tend to implement RFCs differently (McCann, Deering, & Mogul, 1996).

4.3. Neighbor Discovery Protocol (NDP)

The Neighbor Discovery Protocol (NDP) is introduced with IPv6 as a replacement for the IPv4 ARP (Address Resolution Protocol). It is specified in the RFC 1122 and handles several tasks. These tasks are address autoconfiguration of nodes, discovery of other nodes on the same link, including determining the link layer addresses of other nodes, detection of duplicate addresses, locating available routers and DNS

(Domain Name System) servers and discovering the address prefix (Braden, 1989). The NDP is quite a simple protocol and implementation between OS vendors might not vary and therefore not allow OS fingerprinting based on the NDP. However, tests and analysis show that despite the simplicity there are differences between operating systems. Although there are some differences in the implementation between operating systems, these are not enough to precisely tell them apart or even distinguish between OS versions (Beck, Festor, & Chrisment, 2007).

5. IPv6 fingerprinting hands-on examples

This chapter provides some hands-on examples of OS fingerprinting with IPv6. For this purpose some of the methods described in the previous chapter will be tested. The following tests only utilize active fingerprinting methods. Successful passive fingerprinting methods will need a larger network that produced enough network traffic to capture sufficient and suitable packets.

5.1. The test environment

In order to illustrate the application of some of the fingerprinting methods described in the previous chapter, we set up a test environment. The test environment includes the following four operating systems.

Fingerprinting source

- Linux Ubuntu 11.04 (Canonical Ltd., 2011)

Fingerprinting targets

- Microsoft Windows 7 64-bit (6.1, Build 7601) (Microsoft Corporation, 2011)
- Linux Ubuntu 11.04 (Canonical Ltd., 2011)
- PC-BSD 8.2 (iXsystems Inc., 2011)

The OS selection does not cover a wide range of systems and different versions, but it does include one derivative of Microsoft Windows, Linux and Unix. This should be sufficient for the scope of this paper.

All systems are connected via local network only using the IPv6. For performing the fingerprinting and analyzing the network traffic the tools Scapy (version 2.1.0) (Biondi, 2011), traceroute (version 2.0.15 for Linux) and Wireshark (version 1.4.6) (The Wireshark team, 2011) are used.

The following figure shows the test environment setup with the fixed local IPv6 addresses configured for all hosts.

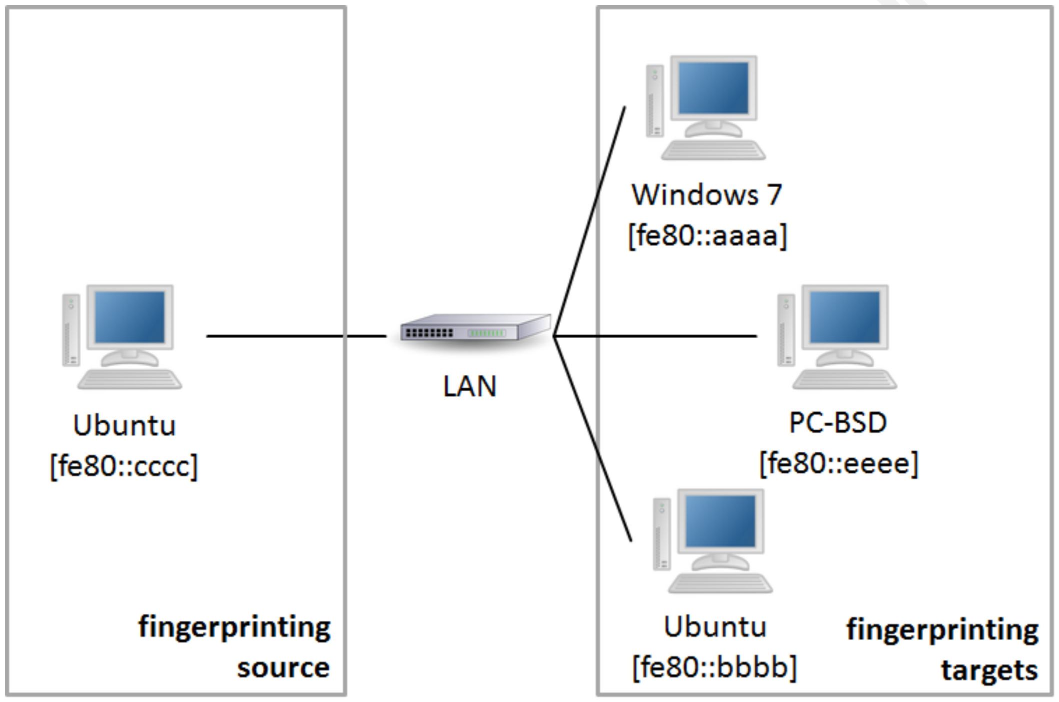


Figure 3 – OS Fingerprinting test environment

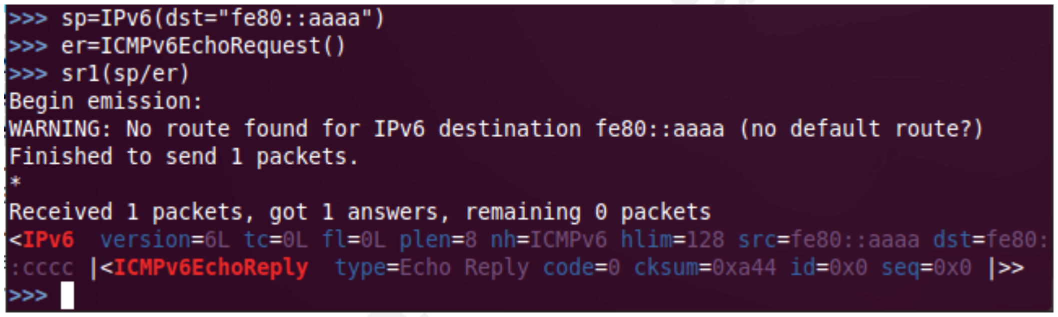
For these tests all the systems and IP addresses are known. This means all targeted IP addresses are known to be a running host, although there might be no response. Furthermore, as all systems are on the same local area network and no router needs to be passed, thus the hop limit will never be decremented. This is important for the hop limit fingerprinting tests.

5.2. Initial hop limit test

In order to test the initial hop limit of a system, the targeted hosts need to send an IPv6 packet. As the hop limit must be set for every IP packet, almost every stimulus that will trigger a response will work. In this case an ICMP echo request packet is chosen to stimulate a response. We craft the ICMP echo request packet using Scapy.

5.2.1. Windows 7

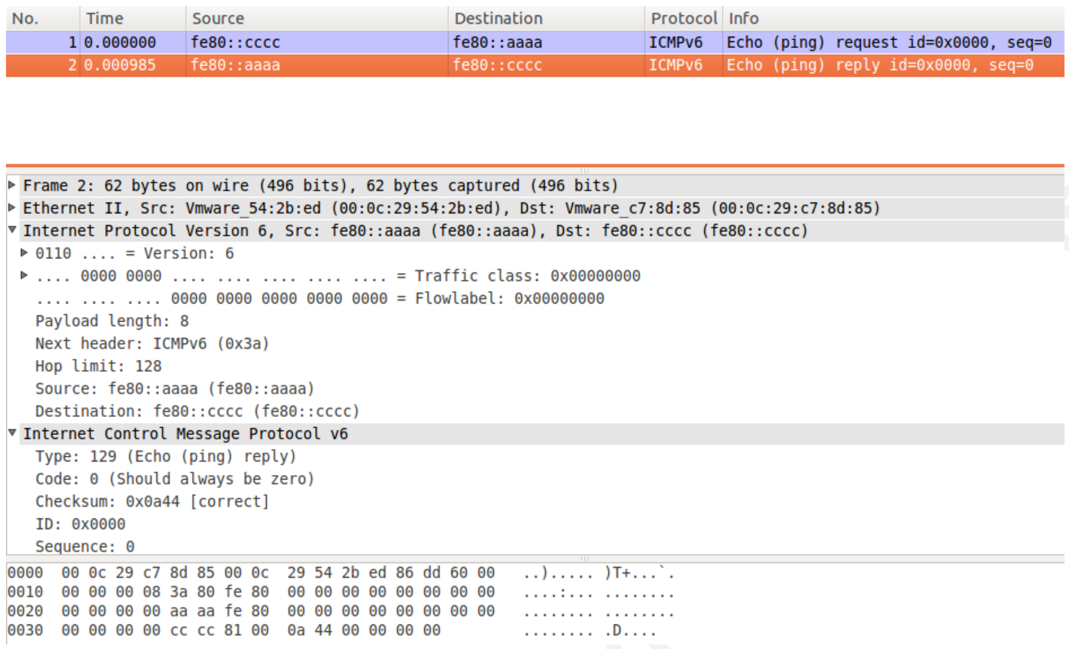
Figure 4 - ICMPv6 echo request against Windows 7 shows how we build an ICMPv6 echo request packet in Scapy. We first create a stimulus packet ("sp") as an IPv6 packet. The target host in this example is the Windows 7 host with the IPv6 address of 'fe80::aaaa'. Next we create an ICMPv6 echo request ("er"). Finally, we send the ICMPv6 echo request using the "sr1()" command. The screenshot also shows parts of the ICMPv6 response packet.



```
>>> sp=IPv6(dst="fe80::aaaa")
>>> er=ICMPv6EchoRequest()
>>> sr1(sp/er)
Begin emission:
WARNING: No route found for IPv6 destination fe80::aaaa (no default route?)
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IPv6  version=6L tc=0L fl=0L plen=8 nh=ICMPv6 hlim=128 src=fe80::aaaa dst=fe80:
:cccc |<ICMPv6EchoReply  type=Echo Reply code=0 cksum=0xa44 id=0x0 seq=0x0 |>>
>>> 
```

Figure 4 - ICMPv6 echo request against Windows 7

The warning can be ignored in this case as we are on a local network and know the host is alive. As we can see, we get one packet in response to our echo request. Figure 5 - Wireshark echo response output from Windows 7 shows the Wireshark output generated by the echo request and echo reply.



```
No.   Time      Source       Destination  Protocol Info
    1 0.000000  fe80::cccc   fe80::aaaa   ICMPv6   Echo (ping) request id=0x0000, seq=0
    2 0.000985  fe80::aaaa   fe80::cccc   ICMPv6   Echo (ping) reply id=0x0000, seq=0
```

```
▶ Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
▶ Ethernet II, Src: Vmware_54:2b:ed (00:0c:29:54:2b:ed), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▼ Internet Protocol Version 6, Src: fe80::aaaa (fe80::aaaa), Dst: fe80::cccc (fe80::cccc)
  ▶ 0110 .... = Version: 6
  ▶ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 8
    Next header: ICMPv6 (0x3a)
    Hop limit: 128
    Source: fe80::aaaa (fe80::aaaa)
    Destination: fe80::cccc (fe80::cccc)
▼ Internet Control Message Protocol v6
    Type: 129 (Echo (ping) reply)
    Code: 0 (Should always be zero)
    Checksum: 0x0a44 [correct]
    ID: 0x0000
    Sequence: 0
```

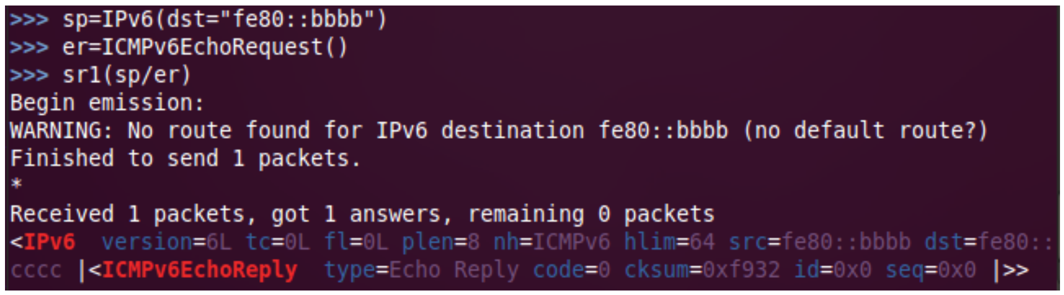
```
0000  00 0c 29 c7 8d 85 00 0c  29 54 2b ed 86 dd 60 00   ..)..... )T+...`.
0010  00 00 00 08 3a 80 fe 80  00 00 00 00 00 00 00 00   ....:... ........
0020  00 00 00 00 aa aa fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 81 00  0a 44 00 00 00 00         ........ .D....
```

Figure 5 - Wireshark echo response output from Windows 7

The first packet we see in the list is the ICMPv6 echo request send from our probing host to the target host with the IPv6 address of 'fe80::aaaa'. The second packet is the echo response send by the targeted host. In the details below we can see that the 'Hop limit' is set to '128' by the targeted host. As the test environment is set up as a local area network, we know the hop limit is the initial hop limit set by the operating system, in this case Windows 7.

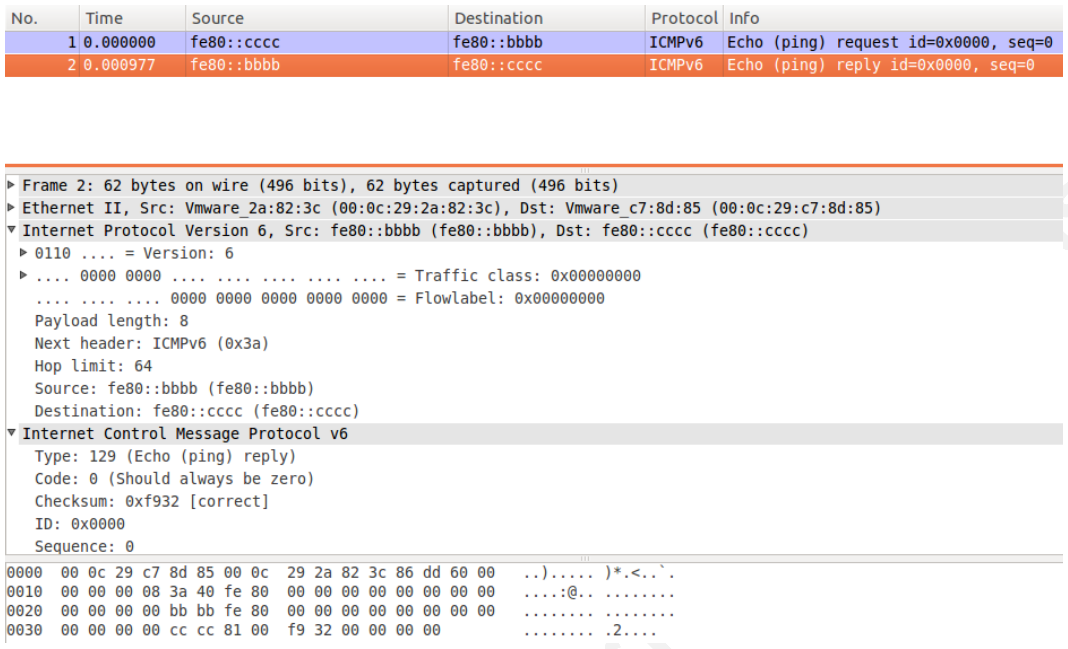
5.2.2. Ubuntu 11.04

We run the same test against the Ubuntu system with the parameters shown in Figure 6 - ICMPv6 echo request against Ubuntu 11.04.



```
>>> sp=IPv6(dst="fe80::bbbb")
>>> er=ICMPv6EchoRequest()
>>> sr1(sp/er)
Begin emission:
WARNING: No route found for IPv6 destination fe80::bbbb (no default route?)
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IPv6  version=6L tc=0L fl=0L plen=8 nh=ICMPv6 hlim=64 src=fe80::bbbb dst=fe80::
cccc |<ICMPv6EchoReply  type=Echo Reply code=0 cksum=0xf932 id=0x0 seq=0x0 |>>
```

Figure 6 - ICMPv6 echo request against Ubuntu 11.04



```
No.   Time      Source        Destination   Protocol  Info
  1   0.000000  fe80::cccc    fe80::bbbb    ICMPv6    Echo (ping) request id=0x0000, seq=0
  2   0.000977  fe80::bbbb    fe80::cccc    ICMPv6    Echo (ping) reply id=0x0000, seq=0
```

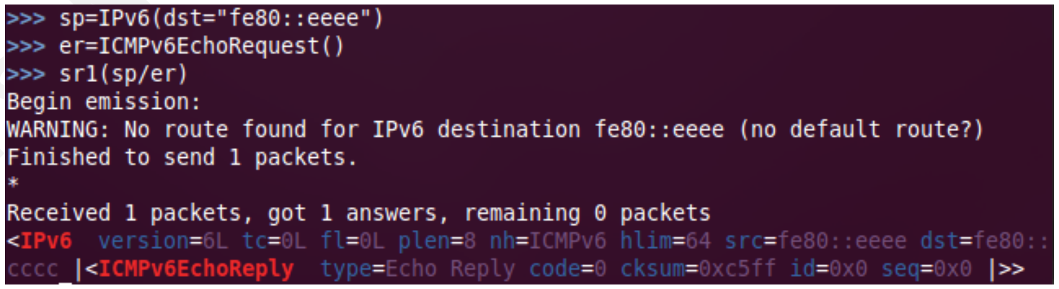
```
▶ Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
▶ Ethernet II, Src: Vmware_2a:82:3c (00:0c:29:2a:82:3c), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▼ Internet Protocol Version 6, Src: fe80::bbbb (fe80::bbbb), Dst: fe80::cccc (fe80::cccc)
  ▶ 0110 .... = Version: 6
  ▶ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 8
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source: fe80::bbbb (fe80::bbbb)
    Destination: fe80::cccc (fe80::cccc)
▼ Internet Control Message Protocol v6
    Type: 129 (Echo (ping) reply)
    Code: 0 (Should always be zero)
    Checksum: 0xf932 [correct]
    ID: 0x0000
    Sequence: 0
0000  00 0c 29 c7 8d 85 00 0c  29 2a 82 3c 86 dd 60 00   ..)..... )*.<..`.
0010  00 00 00 08 3a 40 fe 80  00 00 00 00 00 00 00 00   ....:@.. ........
0020  00 00 00 00 bb bb fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 81 00  f9 32 00 00 00 00         ........ .2....
```

Figure 7 - Wireshark echo response output from Ubuntu 11.04

In Figure 7 - Wireshark echo response output from Ubuntu 11.04 we see the same echo ICMPv6 echo request against the Ubuntu 11.04 host with the IPv6 address of 'fe80::bbbb'. As we can see, the Ubuntu host sets the initial hop limit to '64'.

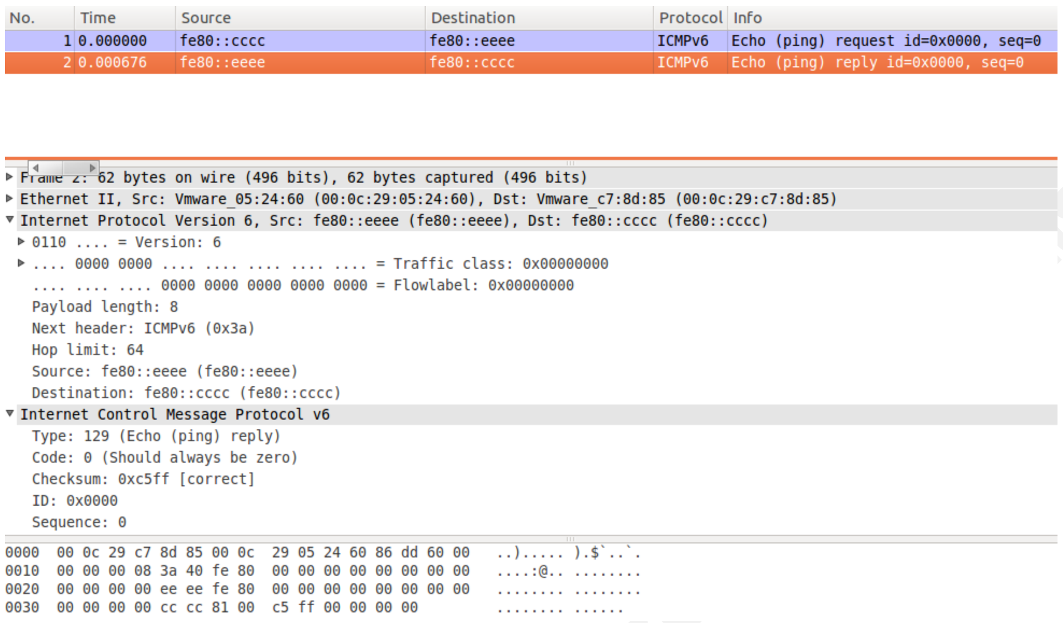
5.2.3. PC-BSD 8.2

Next, we run the ICMPv6 echo request against the PC-BSD host as shown in Figure 8 - ICMPv6 echo request against PC-BSD 8.2.



```
>>> sp=IPv6(dst="fe80::eeee")
>>> er=ICMPv6EchoRequest()
>>> sr1(sp/er)
Begin emission:
WARNING: No route found for IPv6 destination fe80::eeee (no default route?)
Finished to send 1 packets.
*
Received 1 packets, got 1 answers, remaining 0 packets
<IPv6  version=6L tc=0L fl=0L plen=8 nh=ICMPv6 hlim=64 src=fe80::eeee dst=fe80::
cccc |<ICMPv6EchoReply  type=Echo Reply code=0 cksum=0xc5ff id=0x0 seq=0x0 |>>
```

Figure 8 - ICMPv6 echo request against PC-BSD 8.2



```
No.   Time      Source      Destination  Protocol Info
    1 0.000000  fe80::cccc  fe80::eeee   ICMPv6   Echo (ping) request id=0x0000, seq=0
    2 0.000676  fe80::eeee  fe80::cccc   ICMPv6   Echo (ping) reply id=0x0000, seq=0
```

```
▶ Frame 2: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
▶ Ethernet II, Src: Vmware_05:24:60 (00:0c:29:05:24:60), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▼ Internet Protocol Version 6, Src: fe80::eeee (fe80::eeee), Dst: fe80::cccc (fe80::cccc)
  ▶ 0110 .... = Version: 6
  ▶ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 8
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source: fe80::eeee (fe80::eeee)
    Destination: fe80::cccc (fe80::cccc)
▼ Internet Control Message Protocol v6
    Type: 129 (Echo (ping) reply)
    Code: 0 (Should always be zero)
    Checksum: 0xc5ff [correct]
    ID: 0x0000
    Sequence: 0
```

```
0000  00 0c 29 c7 8d 85 00 0c  29 05 24 60 86 dd 60 00   ..)..... ).$`..`.
0010  00 00 00 08 3a 40 fe 80  00 00 00 00 00 00 00 00   ....:@.. ........
0020  00 00 00 00 ee ee fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 81 00  c5 ff 00 00 00 00         ........ ......
```

Figure 9 - Wireshark echo response output from PC-BSD 8.2

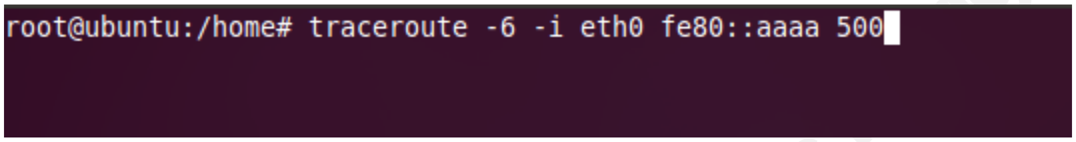
The Wireshark output for the PC-BSD hosts echo reply shows that the hop limit is set to ‘64’, the same as the Ubuntu host.

The ICMPv6 echo request tests show that operating systems use different initial hop limit values in their IPv6 implementation. Like the RFC 791 does not specify a certain initial TTL value for IPv4, the RFC 2460 does not specify a certain initial hop limit for IPv6 packets. In our tests, the Windows 7 host set an initial hop limit of ‘128’, whereas the Ubuntu and PC-BSD host set a value of ‘64’. This shows that operating system can be told apart by their initial hop limit. However, in this case it is only possible to differentiate between Windows and Unix/Linux based systems. To accurately identify the OS, more tests have to be done in order to find more fingerprints which help to tell the different Unix/Linux based systems apart.

5.3. ICMP port unreachable message

In IPv4 the amount of original data contained within the ICMP error messages is not constant and differs by operating system. These differences are used for OS fingerprinting (Nerakis, 2006). In IPv6, respectively ICMPv6, we might expect the same differences between operating systems. To test this behavior we utilize the Linux traceroute tool. Linux traceroute generates a serious of UDP packets to find out the route to a given target IP address. As the fingerprinting source and target are on the same local network and no programs are listening on UDP ports, we expect to see ICMP port unreachable messages as response to see on the network.

In this test scenario we use the following traceroute command to fingerprint the targeted systems.



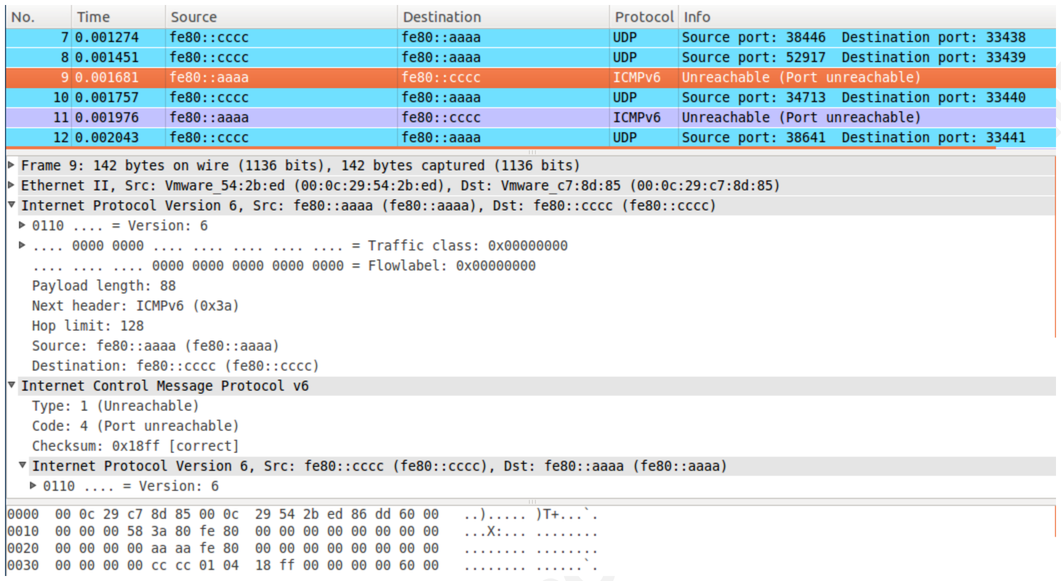
```
root@ubuntu:/home# traceroute -6 -i eth0 fe80::aaaa 500
```

Figure 10 - traceroute command

As we are using IPv6, the option “-6” must be added to the command line. Otherwise, traceroute would send an IPv4 packet. In order to tell traceroute the network interface to use, we specify it by adding “-i eth0”. This tells traceroute to use the eth0 interface, which is necessary, because we are using link local addresses. Next, we specify the target host we are trying to fingerprint. In this example it is the host with the address of “fe80::aaaa”. The last option defines the total packet length in bytes, excluding the Ethernet header, to use for traceroute. As we are looking for differences in the amount of original packet data contained in the response, we are continuously increasing this value. By this we hope to find the maximum payload each operating system returns. If these maximum payloads vary between the systems, we can use this information for OS fingerprinting.

5.3.1. Windows 7

This example shows one ICMPv6 port unreachable error message returned by Windows 7. In this case the IPv6 packet size (excluding the Ethernet header) of the UDP packets is 80 bytes. Shown on screenshot, traceroute sends a serious of UDP packets with different source and destination port variation to the target host. However, we are only interested in the payload length of the ICMPv6 port unreachable message. As we can see on the screenshot, the payload of the IPv6 packet that the Windows 7 host sends in response is ‘88’. This includes the ICMPv6 header, which is composed of 8 bytes. After subtracting the ICMPv6 header length of 8 bytes, the total amount of payload data within the ICMPv6 error message is 80 bytes. That is exactly the IPv6 packet size (excluding the Ethernet header) of the traceroute packets.

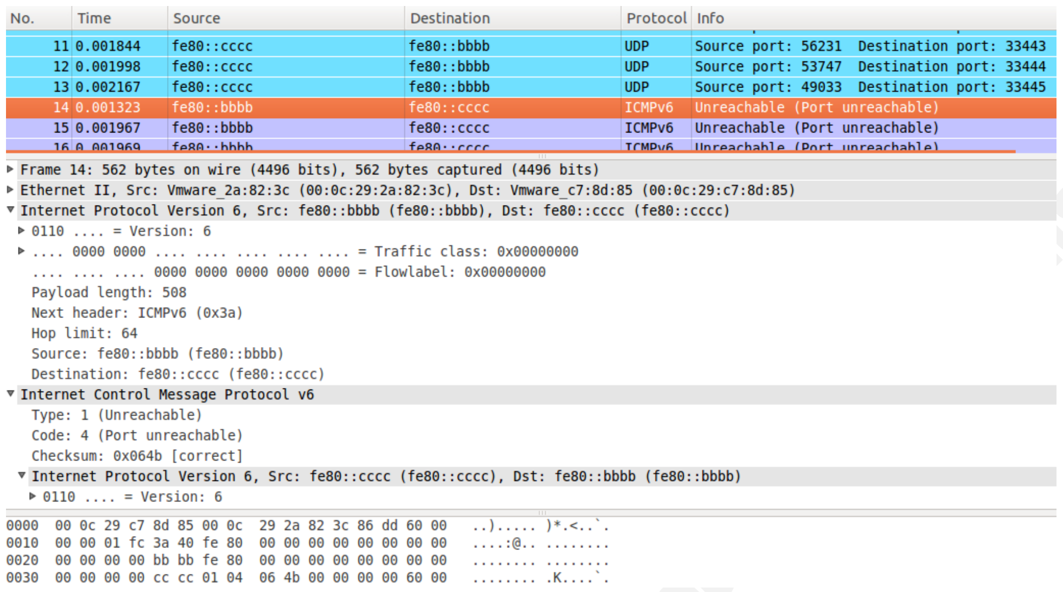


```
No.  Time      Source      Destination  Protocol Info
   7 0.001274  fe80::cccc  fe80::aaaa   UDP      Source port: 38446  Destination port: 33438
   8 0.001451  fe80::cccc  fe80::aaaa   UDP      Source port: 52917  Destination port: 33439
   9 0.001681  fe80::aaaa  fe80::cccc   ICMPv6   Unreachable (Port unreachable)
  10 0.001757  fe80::cccc  fe80::aaaa   UDP      Source port: 34713  Destination port: 33440
  11 0.001976  fe80::aaaa  fe80::cccc   ICMPv6   Unreachable (Port unreachable)
  12 0.002043  fe80::cccc  fe80::aaaa   UDP      Source port: 38641  Destination port: 33441

▶ Frame 9: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits)
▶ Ethernet II, Src: Vmware_54:2b:ed (00:0c:29:54:2b:ed), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▼ Internet Protocol Version 6, Src: fe80::aaaa (fe80::aaaa), Dst: fe80::cccc (fe80::cccc)
  ▶ 0110 .... = Version: 6
  ▶ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 88
    Next header: ICMPv6 (0x3a)
    Hop limit: 128
    Source: fe80::aaaa (fe80::aaaa)
    Destination: fe80::cccc (fe80::cccc)
▼ Internet Control Message Protocol v6
    Type: 1 (Unreachable)
    Code: 4 (Port unreachable)
    Checksum: 0x18ff [correct]
  ▼ Internet Protocol Version 6, Src: fe80::cccc (fe80::cccc), Dst: fe80::aaaa (fe80::aaaa)
    ▶ 0110 .... = Version: 6

0000  00 0c 29 c7 8d 85 00 0c  29 54 2b ed 86 dd 60 00   ..)..... )T+...`.
0010  00 00 00 58 3a 80 fe 80  00 00 00 00 00 00 00 00   ...X:... ........
0020  00 00 00 00 aa aa fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 01 04  18 ff 00 00 00 00 60 00   ........ ......`.
```

Figure 11 - ICMPv6 port unreachable message returned by Windows 7

5.3.2. Ubuntu 11.04

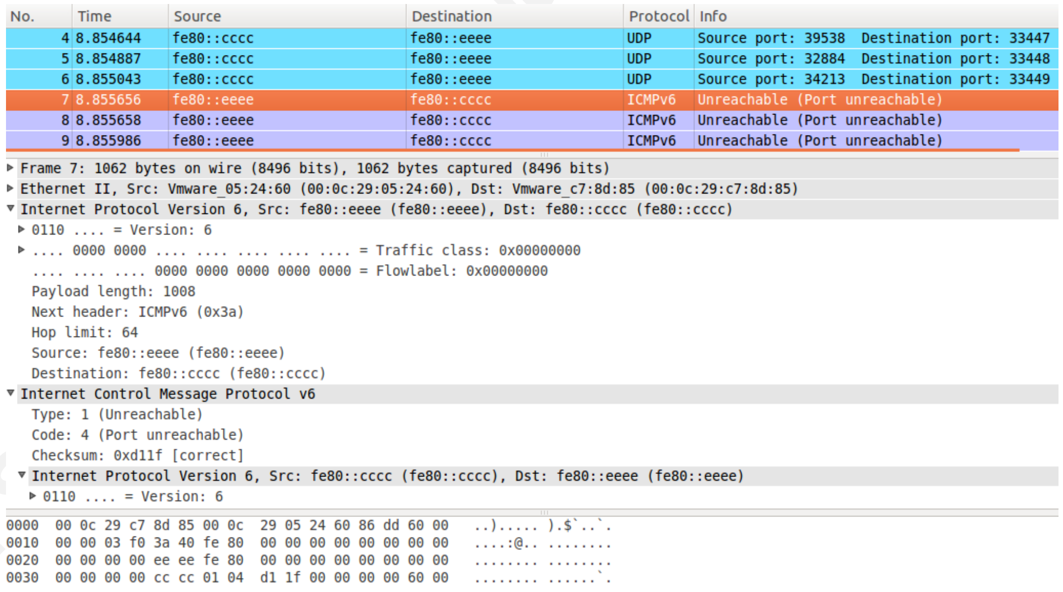
This example shows an ICMPv6 port unreachable message returned by Ubuntu 11.04. For this example we used an IPv6 packet size (excluding the Ethernet header) of 500 bytes for the UDP traceroute packets. As we can see the payload of the IPv6 packet returned by the Ubuntu host is 508 bytes. After subtracting the 8 bytes of the ICMPv6 header we get our 500 bytes of the original packet.



```
No.  Time      Source       Destination  Protocol Info
  11 0.001844  fe80::cccc   fe80::bbbb   UDP      Source port: 56231  Destination port: 33443
  12 0.001998  fe80::cccc   fe80::bbbb   UDP      Source port: 53747  Destination port: 33444
  13 0.002167  fe80::cccc   fe80::bbbb   UDP      Source port: 49033  Destination port: 33445
  14 0.001323  fe80::bbbb   fe80::cccc   ICMPv6   Unreachable (Port unreachable)
  15 0.001967  fe80::bbbb   fe80::cccc   ICMPv6   Unreachable (Port unreachable)
  16 0.001969  fe80::bbbb   fe80::cccc   ICMPv6   Unreachable (Port unreachable)

▸ Frame 14: 562 bytes on wire (4496 bits), 562 bytes captured (4496 bits)
▸ Ethernet II, Src: Vmware_2a:82:3c (00:0c:29:2a:82:3c), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▾ Internet Protocol Version 6, Src: fe80::bbbb (fe80::bbbb), Dst: fe80::cccc (fe80::cccc)
  ▸ 0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 508
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source: fe80::bbbb (fe80::bbbb)
    Destination: fe80::cccc (fe80::cccc)
▾ Internet Control Message Protocol v6
    Type: 1 (Unreachable)
    Code: 4 (Port unreachable)
    Checksum: 0x064b [correct]
  ▾ Internet Protocol Version 6, Src: fe80::cccc (fe80::cccc), Dst: fe80::bbbb (fe80::bbbb)
    ▸ 0110 .... = Version: 6

0000  00 0c 29 c7 8d 85 00 0c  29 2a 82 3c 86 dd 60 00   ..)..... )*.<..`.
0010  00 00 01 fc 3a 40 fe 80  00 00 00 00 00 00 00 00   ....:@.. ........
0020  00 00 00 00 bb bb fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 01 04  06 4b 00 00 00 00 60 00   ........ .K....`.
```

Figure 12 - ICMPv6 port unreachable message returned by Ubuntu 11.04

5.3.3. PC-BSD 8.2



```
No.  Time      Source       Destination  Protocol Info
   4 8.854644  fe80::cccc   fe80::eeee   UDP      Source port: 39538  Destination port: 33447
   5 8.854887  fe80::cccc   fe80::eeee   UDP      Source port: 32884  Destination port: 33448
   6 8.855043  fe80::cccc   fe80::eeee   UDP      Source port: 34213  Destination port: 33449
   7 8.855656  fe80::eeee   fe80::cccc   ICMPv6   Unreachable (Port unreachable)
   8 8.855658  fe80::eeee   fe80::cccc   ICMPv6   Unreachable (Port unreachable)
   9 8.855986  fe80::eeee   fe80::cccc   ICMPv6   Unreachable (Port unreachable)

▸ Frame 7: 1062 bytes on wire (8496 bits), 1062 bytes captured (8496 bits)
▸ Ethernet II, Src: Vmware_05:24:60 (00:0c:29:05:24:60), Dst: Vmware_c7:8d:85 (00:0c:29:c7:8d:85)
▾ Internet Protocol Version 6, Src: fe80::eeee (fe80::eeee), Dst: fe80::cccc (fe80::cccc)
  ▸ 0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 1008
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source: fe80::eeee (fe80::eeee)
    Destination: fe80::cccc (fe80::cccc)
▾ Internet Control Message Protocol v6
    Type: 1 (Unreachable)
    Code: 4 (Port unreachable)
    Checksum: 0xd11f [correct]
  ▾ Internet Protocol Version 6, Src: fe80::cccc (fe80::cccc), Dst: fe80::eeee (fe80::eeee)
    ▸ 0110 .... = Version: 6

0000  00 0c 29 c7 8d 85 00 0c  29 05 24 60 86 dd 60 00   ..)..... ).$`..`.
0010  00 00 03 f0 3a 40 fe 80  00 00 00 00 00 00 00 00   ....:@.. ........
0020  00 00 00 00 ee ee fe 80  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 cc cc 01 04  d1 1f 00 00 00 00 60 00   ........ ......`.
```

Figure 13 - ICMPv6 port unreachable message returned by PC-BSD 8.2

Figure 13 - ICMPv6 port unreachable message returned by PC-BSD 8.2 shows the same test against the PC-BSD host. Again, after subtracting the8 bytes of the ICMPv6 header we get our initial packet size of 1000 bytes, as set in traceroute.

To evaluate the maximum payload of the ICMPv6 error messages, we run a series of tests with growing packet sizes against each operating system. Table 3 - Bytes returned in ICMPv6 port unreachable error messages shows the evaluation of our tests. The first column is the packet size (excluding the Ethernet header) of the UDP packet series sent by traceroute. The second, third and fourth column represent the total bytes of payload in the ICMPv6 port unreachable message returned by the target hosts.

Total bytes sent	Windows 7	Ubuntu 11.04	PC-BSD 8.2
80	80	80	80
250	250	250	250
500	500	500	500
750	750	750	750
1000	1000	1000	1000
1250	1232	1232	1232
1500	1232	1232	1232

Table 3 - Bytes returned in ICMPv6 port unreachable error messages

As we can see in the test results, the amount of data from the original packet never exceeds 1232 bytes. This behavior is common for all three operating systems in this test. The conclusion of this test has to be that, unlike ICMP error messages in IPv4, the ICMPv6 error messages do not offer a fingerprint based on the payload of the original packet.

6. Fingerprinting tools and IPv6 support

As most fingerprinting techniques utilize the TCP and other upper layer protocols, the use of IPv6 is supported in most common network security tools. However, the actual usage of IPv6 protocol header information that could indicate a specific OS is limited.

Nevertheless, there are some tools that make use of IPv6 protocol characteristics to identify operation systems.

6.1. Nmap 5.51

The most well-known OS fingerprinting tool is Nmap. As of version 5.51 Nmap does support ping sweeps with IPv6, but it does not support automated OS identification based on IPv6. Automated probing and analysis are only supported for IPv4 at this time. The next, still unstable version of Nmap 5.59 (currently in beta status) will support automated fingerprinting based on IPv6 (Lyon, 2011a).

6.2. P0f

P0f is the most well-known passive OS fingerprinting tool. It passively detects OS types and versions, firewalls and the usage of NAT. As of now p0f works fine with IPv4, but does not support IPv6 (Beck, Festor, & Chrisment, 2007).

6.3. SinFP 4

SinFP 4 performs TCP active fingerprinting based on IPv6 to determine the remote OS. The tool itself works properly, though the fingerprint database is not very accurate. Thus, the OS identification may not always be very reliable. Besides, some of the tests could be considered as attacks by IDS (Beck, Festor, & Chrisment, 2007).

7. Preventing IPv6 OS fingerprinting

There are just as many techniques and tools for preventing OS fingerprinting as there are for performing OS fingerprinting. Preventing OS fingerprinting does not necessarily mean to prevent the disclosure of any information that might lead to identifying a specific OS, but it may also mean to alter or obfuscate information in order to simulate a different OS to the one actually used. Besides, OS fingerprinting tools rely on a number of different tests to conclusively identify an OS. By blocking or obfuscating some of these tests fingerprinting tools can be misled. (Zamboni & Kruegel, 2006). It must be stated that any kind of preventing OS fingerprinting or obfuscating the identity of an OS should not be considered a reliably security measure. Even so the true identity and version of an OS can be hidden, this cannot stop an attacker from trying every potential exploit for every OS. Therefore, preventing OS fingerprinting slows down hackers, but the only real security mechanism to rely on remains patching and hardening the OS (Allen, 2007).

As said before, there are in general many ways to prevent OS fingerprinting. For instance, welcome banners for FTP, SMTP and other services like Apache and IIS servers could be altered. A firewall or IPS could be used to filter and block outgoing traffic that might give away information to identify an OS. Or services like HTTP and SSH could be configured to use non-standard ports (Trowbridge, 2003).

In order to prevent OS fingerprinting in the case of the IPv6 protocol, either the TCP stack of the OS must be modified or a filtering device must manipulate packets passing through it. A filtering or packet manipulating device will slow down network traffic or even block it. Making modifications to the TCP stack of an OS might not only affect how network traffic appears, but also may have a negative impact on network performance (Allen, 2007).

8. Conclusion

After looking at known fingerprinting methods from IPv4 and newly enabled methods by the IPv6 protocol, we conclude that fingerprinting methods with the IP protocol did not fundamentally change. Some of the known fingerprinting techniques from IPv4 can still be used with IPv6, others are obsolete. Then again, IPv6 also enables new methods of OS fingerprinting which substitute the obsolete methods from IPv4. Overall, OS fingerprinting methods with the IP protocol are still limited with IPv6, hence OS fingerprinting still depends on upper layer protocols like TCP or FTP.

The change from IPv4 to IPv6 does bring one change worth mentioning. The address space is largely enhanced with the IPv6 128-bit IP addresses. As OS fingerprinting sometimes includes finding live hosts on a network first, a simple address range scan is too time consuming within a standard IPv6 subnet with 64-bit reserved for the host address.

This paper gives an overview of some of the OS fingerprinting methods available with IPv6. These methods still need to be intensively tested in an appropriate environment. Although this paper gives examples for some of the methods described, it does not and is not intended to cover OS fingerprinting in depth. The examples are based on three common operating systems in use today, but they barely cover all available operating systems. Likewise, the tests do not differentiate between different versions of a respective OS, which calls for further research. The objective is to test all the methods described with all major operating system including their different versions. This is not only necessary to confirm whether these methods are useful or not, but to build a database of as much IPv6 OS fingerprints as possible.

9. References

6deploy. (2011). *IPv6 Address autoconfigration stateless & stateful.* Retrieved September 08, 2011, from http://www.6deploy.eu/tutorials/080-6deploy_ipv6_autoconfiguration_mechs_v0_4.pdf

6net. (2008). *IPv6 Deployment Guide.* Javvin Press.

Ahmed, A., & Asadullah, S. (2009). *Deploying IPv6 in Broadband Access Networks.* John Wiley & Sons.

Allen, J. M. (2007, September 22). *OS and Application Fingerprinting Techniques.* Retrieved March 22, 2011, from http://www.sans.org/reading_room/whitepapers/authentication/os-application-fingerprinting-techniques_32923

Beck, F., Festor, O., & Chrisment, I. (2007, March). *IPv6 Neighbor Discovery Protocol based OS fingerprinting.* Retrieved March 21, 2011, from http://hal.inria.fr/docs/00/18/48/51/PDF/RT-0345.pdf

Biondi, P. (2011, September). *Scapy*. Retrieved September 03, 2011, from http://www.secdev.org/projects/scapy/

Braden, R. (1989, October). *RFC 1122 - Requirements for Internet Hosts -- Communication Layers*. Retrieved August 24, 2011, from http://www.rfc-editor.org/rfc/rfc1122.txt

Canonical Ltd. (2011). *Ubuntu*. Retrieved September 01, 2011, from http://www.ubuntu.com/

Chown, T. (2005, 10 27). *IPv6 Implications for TCP/UDP Port Scanning.* Retrieved 05 19, 2011, from http://tools.ietf.org/pdf/draft-chown-v6ops-port-scanning-implications-02.pdf

DITCHE. (2005, September). *IPv6 Protocol (RFC 2460 DS).* Retrieved August 27, 2011, from http://www.6diss.org/workshops/saf/ipv6-protocol.pdf

Hermann-Seton, P. (2002). *Security Features in IPv6.* Retrieved August 2011, 23, from http://www.sans.org/reading_room/whitepapers/protocols/security-features-ipv6_380

iXsystems Inc. (2011). *PC-BSD*. Retrieved September 01, 2011, from http://www.pcbsd.org/

Krishnan, S. (2009, December). *RFC 5722 - Handling of Overlapping IPv6 Fragments*. Retrieved September 08, 2011, from http://tools.ietf.org/html/rfc5722

Lyon, G. (2011a). *Nmap*. Retrieved September 04, 2011, from http://nmap.org/

Christoph Eckstein, christoph.eckstein@samapartners.com

Lyon, G. (2011b). *Fingerprinting Methods Avoided by Nmap*. Retrieved September 08, 2011, from http://nmap.org/book/osdetect-other-methods.html

McCann, J., Deering, S., & Mogul, J. (1996, August). *RFC 1981 - Path MTU Discovery for IP version 6*. Retrieved August 24, 2011, from http://www.ietf.org/rfc/rfc1981.txt

Microsoft Corporation. (2011). *Windows 7 - Microsoft Windows*. Retrieved September 01, 2011, from http://windows.microsoft.com/en-US/windows7/products/home

Murphy, N. R., & Malone, D. (2005). *IPv6 Network Administration.* O'Reilly Media.

Nerakis, E. (2006, September). *IPV6 HOST FINGERPRINT.* Retrieved March 21, 2011, from http://faculty.nps.edu/xie/theses/06Sep_Nerakis.pdf

Network Working Group. (1995, January). *RFC 1752 - The Recommendation for the IP Next Generation Protocol*. Retrieved July 19, 2011, from http://www.ietf.org/rfc/rfc1752.txt

Network Working Group. (1998, December). *RFC 2460 - IPv6 Specification*. Retrieved July 19, 2011, from http://www.ietf.org/rfc/rfc2460.txt

Postel, J. (1981, September). *RFC 792 - INTERNET CONTROL MESSAGE PROTOCOL*. Retrieved August 25, 2011, from http://tools.ietf.org/html/rfc792

SANS Institute. (2011a). *TCP/IP and tcpdump Pocket Reference Guide.* Retrieved March 22, 2011, from http://www.sans.org/security-resources/tcpip.pdf

SANS Institute. (2011b). *IPv6 TCP/IP and tcpdump Pocket Reference Guide.* Retrieved March 22, 2011, from http://www.sans.org/security-resources/ipv6_tcpip_pocketguide.pdf

The Wireshark team. (2011, September). *Wireshark - Go deep.* Retrieved September 01, 2011, from http://www.wireshark.org/

Thomson, S., & Narten, T. (1998, December). *RFC 2462 - IPv6 Stateless Address Autoconfiguration*. Retrieved September 08, 2011, from ftp://ftp.ripe.net/rfc/rfc2462.txt

Trowbridge, C. (2003, July 16). *An Overview of Remote Operating System Fingerprinting.* Retrieved March 22, 2001, from http://www.sans.org/reading_room/whitepapers/testing/overview-remote-operating-system-fingerprinting_1231

University of Southern California. (1981, September). *RFC 791 - Internet Protocol*. Retrieved July 19, 2011, from http://www.ietf.org/rfc/rfc791.txt

Christoph Eckstein, christoph.eckstein@samapartners.com

Zamboni, D., & Kruegel, C. (2006). *Recent Advances in Intrusion Detection: 9th International Symposium, RAID 2006, Hamburg, Germany, September 20-22, 2006, Proceedings (Lecture Notes in ... Applications, incl. Internet/Web, and HCI).* Springer.